Learning Disjunctions: Near-Optimal Trade-off between Mistakes and “I Don’t Know”s

Erik D. Demaine* Morteza Zadimoghaddam*

Abstract

We develop polynomial-time online algorithms for learning disjunctions while trading off between the number of mistakes and the number of “I don’t know” answers. In this model, we are given an online adversarial sequence of inputs for an unknown function of the form $f(x_1, x_2, \ldots, x_n) = \bigvee_{i \in S} x_i$, and for each such input, we must guess “true”, “false”, or “I don’t know”, after which we find out the correct output for that input. On the algorithm side, we show how to make at most εn mistakes while answering “I don’t know” at most $(1/\varepsilon)^{2^{O(1/\varepsilon)}} n$ times, which is linear for any constant $\varepsilon > 0$ and polynomial for some $\varepsilon = c/\lg \lg n$. Furthermore, we show how to make $O\left(n \frac{\log \log n}{\log n}\right)$ mistakes while answering “I don’t know” $O(n^2 \log \log n)$ times. On the lower bound side, we show that any algorithm making $o(n/\log n)$ mistakes must answer “I don’t know” a superpolynomial number of times. By contrast, no previous lower bounds were known, and the best previous algorithms (by Sayedi et al. who introduced the model) either make at most $\frac{1}{3}n$ mistakes while answering “I don’t know” $O(n)$ times with linear running time per answer, or make $O(n/\log n)$ mistakes while answering “I don’t know” $O(n^2)$ times with exponential running time per answer. Our lower bound establishes optimality of the latter mistake bound, assuming a polynomial number of “I don’t know”s. The running time of our algorithms (per answer) are $(1/\varepsilon)^{2^{O(1/\varepsilon)}} n$ and $\tilde{O}(n^3)$, respectively, whereas the first previous algorithm mentioned above makes many mistakes, and the second one requires $\Theta(2^n)$ time per answer. The only previous polynomial-time algorithm with reasonable number of mistakes achieves a mistake bound of εn and an “I don’t know” bound of $O(n^{1/\varepsilon})$ which is super polynomial for any non-constant ε.

*MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St, Cambridge, MA 02139, USA. {edemaine,morteza}@mit.edu

1 Introduction

Minimizing mistakes. Supervised learning of functions is one of the central areas within machine learning and computational learning theory. In general, there is a hidden target function that maps an input vector to an output, and an algorithm aims to predict the outputs of various inputs. Specifically, the algorithm is given an online sequence of inputs, and as each input arrives, the algorithm makes a prediction of the output, and then is told the actual output. The classic *mistake bound model* of Littlestone [Lit88] counts the total number of incorrect predictions, and the goal is to minimize that number over a (typically infinite) adversarial input sequence.

In this setting, Littlestone studied the problem of learning a *disjunction*, that is, a function with n Boolean inputs $x_1, x_2, \ldots, x_n$ and a Boolean output defined by the disjunction (Boolean OR) of a subset of these variables and/or their negations. Such functions are our focus in this paper. In general, n mistakes are necessary and sufficient for learning disjunctions in the mistake bound model. Littlestone’s WINNOW algorithm [Lit88] achieves improved performance when the number r of relevant features (variables) in the representation of the target function is low: it makes $O(r \log n)$ mistakes.

Trading off with “I don’t know”s. Sayedi, Zadimoghaddam, and Blum [SZB10] introduced a new model for supervised learning of functions that allows the prediction algorithm to answer “I don’t know” ($\perp$), instead of predicting an output, a limited number of times. The idea is that the algorithm can make fewer mistakes by trading off with this second parameter, and such more-accurate predictions may be more useful in certain applications. This model can be seen as a kind of fusion between the mistake learning model and the Knows What It Knows (KWIK) model of Li, Littman and Walsh [LLW08], where the prediction algorithm must not make any mistakes (with high probability) but can instead say “I don’t know” ($\perp$) whenever it cannot make an accurate prediction, and the goal is simply to minimize the number of such answers.

In the hybrid model with both mistakes and “I

Algorithm	Mistakes	"I don't know"s	Time per query
[SZB10]	$n/3$	$3n/2$	$n^{O(1)}$
[SZB10, Proposition 1]	$n/\lg n$	$O(n^2)$	$\Theta(2^n)$
[SZB10, Proposition 2]	$n/2$	n	$n^{O(1)}$
[SZB10, Proposition 2] modified	εn	$n^{1/\varepsilon}$	$n^{O(1)}$
Algorithm 1 (§3)	εn	$O\left((1/\varepsilon)^{4^{1+1/\varepsilon}} n\right)$	$O\left((1/\varepsilon)^{4^{1+1/\varepsilon}} n\right)$
Algorithm 1 (§3)	$c\,n/\lg\lg n$	$O(n^{1+\varepsilon})$	$O(n^{1+\varepsilon})$
Algorithm 2 (§4)	$O\left(n\frac{\log\log n}{\log n}\right)$	$O(n^2 \log\log n)$	$\tilde{O}(n^3)$

Table 1: Summary of results from [SZB10] and this paper, for any $\varepsilon > 0$ and some constant c.

don't know" answers, Sayedi et al. develop an online algorithm for learning disjunctions that makes at most $n/3$ mistakes and $3n/2$ $\perp$ answers. Can the number of mistakes be further reduced? Using their ideas, it is possible to further reduce the number of mistakes to at most n/k mistakes at the cost of up to n^{k-1} $\perp$ answers, for any integer $k > 0$. Thus, a polynomial number of $\perp$ answers enable bounding the number of mistakes by εn, but only for constant $\varepsilon > 0$.

Stronger than this result, Sayedi et al. show that a simple majority algorithm learns any Boolean function on n variables while making at most $n/\lg n$ mistakes and $O(n^2)$ $\perp$ answers. (Simply set $k = n/\lg n$ in [SZB10, Proposition 1].)

Unfortunately, their latter algorithm which achieves the optimal trade-off between mistakes and $\perp$ answers requires exponential time to compute the prediction (as mentioned in [SZB10]). For each query, they consider all 2^n disjunction functions that might be the target, and perform some $O(1)$-time processing on each. Sayedi et al. also develop a polynomial-time algorithm for learning disjunctions that makes at most $n/2$ mistakes and n $\perp$ answers. Using their ideas, it is again possible to reduce the number of mistakes to at most n/k, at the cost of making up to n^{k-1} $\perp$ answers, for any integer $k > 0$.

Our results. We present online algorithms for learning disjunctions that are efficient in both a computational sense (making predictions in polynomial time) and an error sense (nearly optimally trading off between mistakes and "I don't know"s). Table 1 summarizes our results and those in [SZB10].

In Section 3, for any constant $\varepsilon > 0$, we present an algorithm that makes at most εn mistakes and at most $(1/\varepsilon)^{2^{O(1/\varepsilon)}} n$ $\perp$ answers, while running in $(1/\varepsilon)^{2^{O(1/\varepsilon)}} n$ time per query. In particular, the $\perp$ and time bounds are linear for any constant $\varepsilon > 0$ and polynomial for $\varepsilon \geq c/\lg\lg n$ for a specific constant c. In particular, for an appropriate $\varepsilon = c/\lg\lg n$, we learn a disjunction while making $O(n/\log\log n)$ mistakes and $O(n^{1+\varepsilon'})$ $\perp$ answers, while running in $O(n^{1+\varepsilon'})$ time per query.

In Section 4, we present an algorithm that makes $O\left(n\frac{\log\log n}{\log n}\right)$ mistakes and $O(n^2 \log\log n)$ $\perp$ answers, while running in $\tilde{O}(n^3)$ time per query.

These two algorithms are the first polynomial-time learning algorithms making $o(n)$ mistakes and a polynomial number of $\perp$ answers. Our algorithms work even if the adversary has complete knowledge of the algorithm, and designs the sequence of queries based on that information.

In Section 5, we prove that, for some sequences of queries, any learning algorithm must either make $\Omega(n/\log n)$ mistakes or make a superpolynomial number of $\perp$ answers. Thus, if the algorithm makes a polynomial number of $\perp$ answers, it must make at least $\Omega(n/\log n)$ mistakes. Our hardness results hold even for algorithms that have unbounded computational power, such as algorithms with exponential running time per query. Thus we prove the optimality of the mistake bound in the (exponential-time) majority algorithm of Sayedi et al. [SZB10] mentioned above. Our lower bound also shows that the trade-off between mistakes and "I don't know"s obtained by our efficient learning algorithms are nearly tight; in particular, our second algorithm is within a $\Theta(\log\log n)$ factor from the optimal mistake bound.

Techniques. In Section 3, we develop a learning algorithm that makes at most εn mistakes by maintaining a categorized knowledge base. We keep $\lceil 1/\varepsilon \rceil$ collections of subsets of variables such that the label of every subset in these collections is True, and the correlation between these subsets is low in a certain sense. In this way, we can answer new queries based on the new information we can get out of them. We show how to update these collections carefully to keep the desired properties needed by our algorithm.

In Section 4, we show how to improve the number of mistakes to $O\left(n\frac{\log\log n}{\log n}\right)$. In this case, we use linear constraints to represent the knowledge we have learned so far, resulting in a convex polytope. Then we use the algorithm of Lovász and Vempala [LV06] for estimating the volume of this polytope, to estimate the knowledge learned from a particular outcome.

In Section 5, we prove hardness results for algorithms that learn disjunction functions by generating a randomized sequence of queries, and set their correct answers based on algorithm's answers to ensure that the algorithm can get rid of only a small fraction of variables. We keep the size of query sets logarithmic to be able to set the labels of many queries to either True or False. This approach allows us to force the algorithm to make a mistake or say $\perp$.

2 Model

Our goal is to learn a target function f known to belong to the *target concept class* H_n, which consists of all disjunction functions on n Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$. A *disjunction function* $f : \{\text{True}, \text{False}\}^n \to \{\text{True}, \text{False}\}$ is the disjunction (Boolean OR) of a subset of the variables or their negations. In fact, we can assume that the function f is *monotone*, meaning that f is the disjunction of a subset of the variables (and not their negations). We can make this assumption because we can consider the negations as n additional variables, and thereby view f as monotone on at most $2n$ variables. This transformation potentially doubles the value of n; all bounds we state are in terms of the increased value of n in a monotone disjunction.

Thus the target function f we aim to learn is defined by a subset of variables, $R \subseteq X$, which we call *relevant variables*, combined by a disjunction. In other words, $f(Q)$ measures whether any relevant variables are True in the query q. Viewing R and Q as Boolean vectors of length n, f is a Boolean dot product (with AND replacing multiplication and OR replacing addition). On the other hand, we can think of the query Q as a subset of variables, $Q \subseteq X$, namely, which variables x_i are set to True. Then $f(Q)$ is True if and only if $Q \cap R \neq \emptyset$, i.e., the value of at least one of the relevant variables is True in the input query.

In our setting, a sequence of adversarial queries arrive online, and for each query Q, we must predict the label of Q, i.e., the function value $f(Q)$, or we can say $\perp$ ("I don't know"). Afterwards, we learn the correct label $f(Q)$ (and thus whether our prediction, if we made one, was correct). Our goal is to minimize both the number of mistakes and the number of $\perp$ answers. This bicriterion optimization can be modeled as being given an upper bound given on the number of mistakes, and subject to this constraint, we want to minimize the number of $\perp$ answers.

Algorithm 1 — predicting the label of a query Q

if the label of Q can be inferred **then** return the inferred label;
else if $|Q \cap P| > k$ **then** return True;
else if Q has some critical subset **then** return True;
else return $\perp$.
In all cases, update maintained information according to Q and its correct label.

3 Algorithm based on Critical Subsets

We present a learning algorithm that is efficient in terms of both error complexity and computational complexity:

THEOREM 3.1. *For any constant $\varepsilon > 0$, Algorithm 1 (presented below) learns the target disjunction function while making at most εn mistakes and $O((1/\varepsilon)^{4^{1+1/\varepsilon}} n)$ $\perp$ answers. The algorithm's running time per query is $O((1/\varepsilon)^{4^{1+1/\varepsilon}} n)$.*

In particular, we obtain the first polynomial-time learning algorithm for disjunctions that makes $o(n)$ mistakes and a polynomial number of $\perp$ answers:

COROLLARY 3.1. *For some $\varepsilon = \Theta(1/\log\log n)$, Algorithm 1 learns the target disjunction function with at most $O(n/\log\log n)$ mistakes, and $O(n^{1+\varepsilon'})$ $\perp$ answers. For this subconstant value of ε, the running time per query is $O(n^{1+\varepsilon'})$.*

Proof: Set $1/\varepsilon = \log_4(\varepsilon' \lg n/\lg\lg\lg n) = \Theta(\log\log n)$. $\square$

3.1 Overview.

Algorithm 1 summarizes our learning algorithm at a high level. We describe each of the steps now, and follow up with specific implementation details, data structures, etc., in Section 3.2.

Maintained information. Throughout the sequence of queries, the algorithm maintains two disjoint subsets $P^-, P^+ \subseteq X$ of variables. We maintain the invariant that every variable in P^- is guaranteed irrelevant ($P^- \cap R = \emptyset$), and every variable in P^+ is guaranteed relevant ($P^+ \subseteq R$). Let P denote the remaining, undetermined variables: $P = X \setminus (P^- \cup P^+)$.

The algorithm maintains additional information about certain queries with True labels (i.e., for which the disjunction is true). Let $k = \lceil 1/\varepsilon \rceil$. For each i with $2 \leq i \leq k$, the algorithm maintains a family A_i of subsets of variables such that each subset $S \in A_i$ has size i and $S \cap R \neq \emptyset$ (so the label for S is True for sure).

At the very beginning, the sets P^-, P^+, A_2, A_3, $\ldots$, A_k are all empty, and $P = X$.

Step 1: Inferable label. Now consider a query Q, viewed as the set of variables with True values. First the algorithm checks whether it can infer the label of Q based on the maintained information listed above. Formally, if $Q \cap P^+ \neq \emptyset$, or Q is a superset of one of the subsets in families $\{A_i\}_{i=2}^k$, then we answer True; and if $Q \subseteq P^-$, then we answer False. In either case, we are guaranteed to be correct by our invariants. Otherwise, we proceed to the next step.

Step 2: Large sets. If the query Q has at least k unknown variables, i.e., $|Q \cap P| \geq k$, then we can afford to simply answer "True". If this prediction turns out to be incorrect (i.e., the correct answer is False), then we learn that all (at least k) variables in $Q \cap P$ are irrelevant, so we add them to P^-.

Step 3: Critical subsets. The heart of our algorithm is the notion (and handling) of critical subsets. For each i with $2 \leq i \leq k$, let $\ell_i = k^{4^i}$. Call a set S *critical* with respect to family A_j, where $j > |S|$, if there are $\ell_{j-|S|+1}$ supersets of S in family A_j, i.e., $|\{T \mid T \in A_j, S \subseteq T\}| = \ell_{j-|S|+1}$. We maintain the invariant that the number of these supersets never exceeds $\ell_{j-|S|+1}$ (see Lemma 3.2 below) by detecting and handling critical sets.

Specifically, let $Q' = Q \setminus P^-$. If Q' has a critical subset $S \subseteq Q'$ with respect to some family A_i, then we answer True. If this prediction turns out to be incorrect (i.e., the correct answer is False), then we update the maintained information as follows. Let $j = i - |S| + 1$, and let $T_1, T_2, \ldots, T_{\ell_j}$ be the ℓ_j supersets of set S in family A_i. (If Q' has multiple critical subsets with respect to different families, we choose an arbitrary critical set S with respect to some family A_i.) For $1 \leq x \leq \ell_j$, let $T'_x = T_x \setminus S$. Note that T'_x has size $j-1$, and the label for T'_x must be True because we know that the label of $T_x \in A_i$ is True, and all variables in $S \subseteq Q$ are definitely not in R. So we can potentially insert the set T'_x into family A_{j-1}. We select a disjoint family of these T'_x sets by the following greedy algorithm: select any T'_k that has not been selected or discarded, discard any T'_ms that have nonempty intersection with T'_k, and repeat until every T'_k has been selected or discarded. If $j > 2$, then we add these selected sets to the family A_{j-1}. If $j = 2$, then each selected set contains a single variable which must be a relevant variable, so we add them to the set P^+. In Lemma 3.3 of Section 3.3 below, we prove that we always choose at least $\sqrt{\ell_j}/j$ sets to add to A_{j-1} (for $j > 2$) or P^+ (for $j = 2$).

Step 4: Don't know. If none of the above conditions hold, we answer $\perp$. If the correct label turns out to be False, we add the variables of Q to P^-. If the correct label turns out to be True, then we add Q to the family $A_{|Q|}$.

3.2 Implementation Details. Next we explain how to maintain P^+, P^-, P, and the families $\{A_i\}_{i=2}^k$ in a data structure, and for each query, how to compute our prediction and update the data structure. We maintain P, P^-, and P^+ by storing, for each variable, to which of the three sets it belongs. For each family A_i, $2 \leq i \leq k$, and for each Boolean variable x_j, $1 \leq j \leq n$, we store a linked list of sets in the family A_i that contain variable x_j. We prove in Lemma 3.2 below that the size of each of these linked lists is at most ℓ_i Because the same set appears in multiple linked list, we add links (pointers) from any copy of a set to the other copies.

When a query Q arrives, we can check in $O(n)$ time the three conditions: $Q \cap P^+ \neq \emptyset$, $Q \subseteq P^-$, and $|Q \cap P| > k$. To check whether Q is a superset of some subset in one of the families, we can take a variable in Q and scan over all linked lists of this variable and any family A_j, $2 \leq j \leq k$. If none of these conditions hold, we replace Q with $Q' = Q \setminus P^-$ because we know that the variables in P^- or P^+ are not in any of the sets in the families. The new set Q' has size at most k. For any of the at most 2^k subsets of Q, say S, and for any family A_i with $i > |S|$, we need to check whether S is critical with respect to A_i. We take an arbitrary variable $x_j \in S$, and check the linked list for variable x_j and family A_i. We can simply count the supersets of S in this linked list to check the criticality condition.

We categorize the update operations as follows. When we predict the label of a query correctly, we do not need any updates. In other cases, we do the following:

1. When a query contains a critical set S with respect to family A_i and its correct label is False, we find the ℓ_j supersets of S in family A_i by scanning the linked list of some variable in S and family A_i. We then remove S from these supersets, and in the new ℓ_j sets, we take a greedy approach to select at least $\sqrt{\ell_j}/j$ disjoint sets among them. Every time we select one of them, we just need to check all other sets to eliminate those that have some variables in common with the selected set.

2. When we add a set S' to some family A_i, we should remove all of its supersets in higher families $A_{i+1}, A_{i+2}, \ldots, A_k$. This removal can be done by taking an arbitrary variable in S' and scanning the linked lists of this variable and all families A_j with

$i+1 \le j \le k$.

3. Again when we add a set S' to family A_i, we find every subset $S'' \subset S'$ that has become critical with respect to family A_i after inserting S' to family A_i. There are at most 2^k subsets for S', and we can check this criticality condition by scanning and counting over the linked list of some variable in S'' and family A_i. If some set S'' is critical with respect to family A_i, we remove its supersets in families A_j with $i+1 \le j \le k$ by scanning some linked lists.

4. Whenever we add a variable to set P^- or P^+, we remove all sets that contain this variable in all families $\{A_i\}_{i=2}^k$. This is intended to simplify the analysis.

Given these implementation details, we can analyze the running time of our algorithm:

Lemma 3.1. *The total running time (answering the query and updating maintained information based on the received correct label) is $O((1/\varepsilon)^{4^{1+1/\varepsilon}} n)$.*

Proof: As mentioned in Section 3.2, checking the three conditions $Q \cap P^+ \neq \emptyset$, $Q \subseteq P^-$, and $|Q \cap P| > k$ can be done in $O(n)$ time. Checking whether Q is a superset of some subset in one of the families can be done in time $O(2^k \sum_{j=2}^k \ell_j) \le k^{4^{k+1}}$. To find a critical subset of Q (if there exists any), we check all subsets of $Q \setminus P^-$ (at most 2^k subsets). For each of them we scan at most $k-2$ linked lists of total size at most $\sum_{j=2}^k \ell_j$. So the total running time of this operation is at most $2^k \sum_{j=2}^k \ell_j \le k^{4^{k+1}}$.

For updating our families, when we insert some variable to P^- and P^+, we scan and remove at most $\sum_{j=2}^k j\ell_j \le k^{4^{k+1}}$ variables from various linked lists. The total time over, all queries, we spend for these types of update operations is at most $nk^{4^{k+1}}$. When a query contains a critical set S with respect to family A_i and its correct label is False, we find the ℓ_j supersets of S in family A_i by scanning the linked list of some variable in S, and family A_i which takes ℓ_i time. We then remove S from these supersets, and in the new ℓ_j sets, we take a greedy approach to select at least $\sqrt{\ell_j}/j$ disjoint sets among them. Every time we select one of them, we just need to check all other sets to eliminate the ones that have some variables in common with the selected set. This operation takes time $\ell_j^2 \le k^{4^{k+1}}$.

The last update operation is when we add a set S to some family A_i. We need to remove all its supersets. It can be done by taking an arbitrary variable in S, and scanning the linked lists of this variable and all families $\{A_j\}_{j=i+1}^k$ which takes time $\sum_{j=i+1}^k \ell_j \le_j\}_{j=i+1}^k$. The other operation, is to find every subset $S' \subset S$ that has become critical with respect to family A_i after inserting S to family A_i. There are at most 2^k subsets for S, and we can check this criticality condition by scanning and counting over the linked list of some variable in S' and family A_i. This takes at most $2^k\ell_i \le k^{4^{k+1}}$ time . In case, some set S' is critical with respect to family A_i, we remove its supersets in families $\{A_j\}_{j=i+1}^k$ by scanning some linked lists which takes time $\sum_{j=i+1}^k \ell_j \le k^{4^{k+1}}$. □

3.3 Analysis. In this section, we prove some invariant structural properties about the families of sets $\{A_i\}_{i=2}^k$. We then upper bound the number of mistakes and $\perp$ answers, by analyzing the number of sets in each family. Because we might remove some sets from a family, we count and upper bound the number of sets that we ever insert into a family during the algorithm.

First we show that the critical sets represent the maximum level of "density" in our families:

Lemma 3.2. *After updating our families based on a query and its label, for any set S and $2 \le i \le k$, the number of supersets of S in family A_i is not more than $\ell_{i-|S|+1}$.*

Proof: There are two cases that we add sets into families. The first case is when we answer $\perp$ and the correct label is True. In this case, every subset of the query is noncritical; otherwise we would have answered "True" instead of $\perp$. Adding this query set to a family might make some of its subsets critical. This insertion does not violate anything because it is just one set.

In the other case, the query contains a critical set S with respect to some family A_i, we answer "True", and the correct label is False. So we consider $\ell_{i-|S|+1} = \ell_j$ sets containing S in family A_i, and remove S from them. Then we select at least $\sqrt{\ell_j}/j$ disjoint sets among them, and insert them into family A_{j-1} (only for $j > 2$). Because they are disjoint, we just need to prove that there is no subset of the selected sets that is critical with respect to family A_{j-1} before the update operations. Suppose not, and one of the selected sets T'_x contains a set S' which is critical with respect to family A_{j-1} before we perform the update operations. But when this set became critical, we removed all its supersets with size more than $j-1$. So there should not exist a set $T_x \supset T'_x \supseteq S'$ in family A_i (note that $i > j$). So the selected sets do not contain any critical subset with respect to A_j. We conclude that inserting them does not violate any desired property. □

Now we have the right tool to prove that, when we answer True for a query that contains some critical set, we will always find at least $\sqrt{\ell_j}/j$ disjoint sets of size $j-1$:

LEMMA 3.3. *For any query Q that contains some critical subset S with respect to family A_i, if our prediction is not correct, the number of selected sets of size $j-1 = i-|S|$ from our greedy approach is always at least $\sqrt{\ell_j}/j$.*

Proof: Let $T_1, T_2, \ldots, T_{\ell_j}$ be the ℓ_j supersets of set S in family A_i. Define T'_x to be $T_x \setminus S$ for any $1 \le x \le \ell_j$. First of all, we prove that there cannot be more than $\sqrt{\ell_j}$ sets among these T' sets which share a common variable x_c for any $1 \le c \le n$. Suppose there are more than $\sqrt{\ell_j} \ge \ell_{j-1}$ (this is implied from the definition of ℓ_j's) sets in family $\{T'_x\}_{x=1}^{\ell_j}$ which share a variable x_c in addition to all variables in S. These $\ell_{j-1}+1$ sets contradict with Lemma 3.2 because they all share $|S|+1$ variables and they are all in family A_i, and the number of them is more than $\ell_{i-|S|-1+1} = \ell_{j-1}$.

At each step, we select an arbitrary set among T' sets, and remove all sets in family $\{T'_x\}_{x=1}^{\ell_j}$ that have nonempty intersection with this selected set. Note that each variable is shared by at most $\sqrt{\ell_j}$ sets, and each selected set has $j-1$ variables. So we remove at most $(j-1)\sqrt{\ell_j}$ sets for each selected set. We can imply that we select at least $\frac{\ell_j}{1+(j-1)\sqrt{\ell_j}} \ge \sqrt{\ell_j}/j$ sets. $\square$

Now we are ready to upper bound the total number of sets we insert into each family during algorithm. For $2 \le i \le k$, let D_i be the maximum sum of the numbers of sets that we insert in total to families $A_2, A_3, \ldots, A_i$ during algorithm.

LEMMA 3.4. *For $2 \le i \le k$, the maximum total number D_i of sets inserted into $\{A_j\}_{j=2}^{i}$ is at most $U_i = 3n(i! \prod_{j=2}^{i} \ell_j)/4$.*

Proof: We prove the lemma by induction on i. As the induction base, we prove an upper bound of $U_2 = 3n\ell_2/2$ on D_2. We consider the sets in family A_2 in two cases.

- The sets in family A_2 that have been removed at some point in the algorithm.
- The sets in family A_2 that never have been removed, and are present in A_2 until the end of algorithm.

Every time we add a variable x_j to P^- or P^+, we remove all sets in A_2 that contain x_j. There are at most ℓ_2 such sets, and we add at most n variables to P^- or P^+. This means that there are at most $n\ell_2$ sets in A_2 that we remove. On the other hand, every set that has been inserted into A_2, and never has been removed, are present at the end of algorithm. By Lemma 3.2, we know that for each variable x_j there are at most $\ell_{2-1+1} = \ell_2$ sets in A_2 containing it which means that there are at most $\frac{n\ell_2}{2}$ sets present in A_2 at the end of algorithm, i.e., the term 2 in the denominator of the ratio comes from the fact that each set in A_2 has two variables. We conclude that T_2 is at most $n\ell_2 + n\ell_2/2 = 3n\ell_2/2$.

Now for any $i > 2$, the total number of sets that we are present in family A_i at the end of the algorithm does not exceed $n\ell_i/i$ because every set in A_i has i variables, and every variable cannot be in more than $\ell_{i-1+1} = \ell_i$ sets in family A_i, this is an implication of Lemma 3.2. Now it suffices to bound the number of removed sets from A_i in total. Every time a variable is inserted into P^- or P^+, we might remove all sets containing it. Each variable is contained in at most ℓ_i sets in A_i, and this happens at most n times. So there are at most $n\ell_i$ sets removed from A_3 in this way. For every set S we add to family A_j, we might remove some sets from family A_i in two possible ways. We note that $|S| = j$ is less than i. We might remove a superset of S in A_i or inserting S to family A_j might make some subset $S' \subset S$ critical with respect to family A_j, and then we remove all supersets of S' in family A_i. In both cases, the subsets we remove from family A_i should have at least one common variable with set S. So the number of sets we remove from A_i because of S does not exceed $(i-1)\ell_i$. Because each variable in S is contained in at most ℓ_i sets in A_i (again by Lemma 3.2), and there are at most $i-1$ variables in S. We also know that the number of sets S that we insert to families $A_2, A_3, \ldots, A_{i-1}$ is at most U_{i-1} by induction. We conclude that D_i is at most $n\ell_i/i + n\ell_i + (i-1)\ell_i U_{i-1} + U_{i-1}$. By definition of U_{i-1} and ℓ_i, we have that D_i is at most $i\ell_i U_{i-1}$. So we can show that D_i is at most

$$\left(\frac{i!}{2}\prod_{j=3}^{i}\ell_j\right) D_2 = \frac{3}{4} i!\, n \prod_{j=2}^{i}\ell_j$$

$\square$

Finally we are ready to prove Theorem 3.1:

Proof of Theorem 3.1: We might make mistakes in two situations: (1) when the query has more than k variables in common with set P, and (2) when the query contains a critical set with respect to some family. In the first case, for every mistake we make, at least $k+1$ variables are added to P^-, so there are at most $n/(k+1)$ mistakes of this type. We just need to prove that the number of mistakes of the second type is at most $n/k(k+1)$.

We have $k-1$ families $A_2, A_3, \ldots, A_k$. We prove that, for each $3 \le j \le k$, the number of mistakes made because of a critical set S with respect to family A_i where $j = i - |S|$ is at most $n/(k-1)k(k+1) \le n/k^3$. Therefore there are at most $n/k(k+1)$ mistakes of the second type. Every time we make a mistake because of a critical set S with respect to family A_i, there are at least $\sqrt{\ell_j}/j$ sets added to family A_{j-1} (for $j > 2$) by Lemma 3.3 where $j = i - |S|$. Therefore we can make at most $jD_{j-1}/\sqrt{\ell_j}$ mistakes. By Lemma 3.4, we know that D_{j-1} is at most $3n((j-1)!\prod_{j'=2}^{j-1}\ell_{j'})/4$. Based on definition of ℓ_js, we conclude that the number of mistakes for this $j = i - |S| \ge 2$ is at most

$$jD_{j-1}/\sqrt{\ell_j} = \frac{3nj(j-1)!\prod_{j'=2}^{j-1}\ell_{j'}}{4\sqrt{\ell_j}}$$

$$\le n\frac{k^{j+\sum_{j'=2}^{j-1}4^{j'}}}{k^{4^j/2}} \le n\frac{k^{j+4^j/3}}{k^{4^j/2}} \le n/k^3.$$

We also note that for cases with $j = 2$, we insert at least $\sqrt{\ell_2}/2 \ge k^8/2 > k^7$ variables to P^+ every time we make a mistake. So the total number of mistakes of this type (for $j = 2$) is at most n/k^3 because we have at most n variables to insert to P^+. This completes the claim that the total number of mistakes in n/k.

To bound the number of $\perp$ answers, we just note that for any $\perp$ answer with label True, we insert a set into one of the families. This means that the number of these types of $\perp$ answers is at most $U_k = 3n(k!\prod_{j=2}^{k}\ell_j)/4 \le nk^{k+\sum_{j=2}^{k}4^j} < n(k^{4^{k+1}}-1)$ which is what we claim in this theorem. For each $\perp$ answer with label False, we insert some variable to P^- which means the number of them is at most n. This completes the proof of the upper bound on the number of $\perp$ answers.

The desired bound on the running time is proved separately in Lemma 3.1. $\square$

4 Algorithm based on Polytope Sampling

In this section, we present a relatively simple algorithm that makes fewer mistakes at the cost of more running time per query.

Theorem 4.1. *Algorithm 2 (presented below) learns the target disjunction function while making* $O\left(n\frac{\log\log n}{\log n}\right)$ *mistakes and* $O(n^2 \log\log n)$ $\perp$ *answers. The algorithm's running time per query is* $\tilde{O}(n^3)$.

Algorithm 2 tries to find the correct label of the query if it can be implied based on previous queries. Similar to Algorithm 1, we define sets of variables P and P^- (but not P^+). In addition to these three sets, we maintain a linear program (convex polytope) LP, which is the main new idea of this algorithm. The convex polytope LP has n variables $z_1, z_2, \ldots, z_n$, one for each boolean variable. We initialize the linear program with n constraints $z_i \in [0, 1+\varepsilon]$, where ε is defined to be $1/(2\lg n)$. We partition the queries into two types as follows, and update this convex polytope LP (by adding an extra linear constraint) only when a query of the second type comes.

Algorithm 2 — predicting the label of a query Q

1: **if** the label of Q can be inferred
   then return the inferred label;
2: **else if** $|Q \cap P| > \lg n$ **then** return True;
3: **else if** adding constraint $\sum_{x_i \in Q \cap P} z_i \ge 1$ reduces the volume of LP by a factor of at least n
   then return False;
4: **else if** adding constraint $\sum_{x_i \in Q \cap P} z_i < 1$ reduces the volume of LP by a factor of at least n
   then return True;
5: **else** return $\perp$.
6: In all cases, perform updates according to Q and its correct label.

During our algorithm, we need subroutines to find volumes of some parts of our convex polytope LP. We can use any of the several available algorithms to find the volume of or sample a random point from a convex polytope, but that of Lovász and Vempala [LV06] seems to be the fastest with running time $\tilde{O}(n^3)$. The running time of our algorithm is essentially coming from this sampling LP subroutine.

An input query Q has one of the following two types, each of which is processed in a specific way:

(a) If there are at least $\lg n$ variables in $P \cap Q$, the algorithm predicts the label of Q to be true. We do not update the convex polytope LP in this case. If we make a mistake, which means that the correct label of Q is false, we remove the variables of $Q \cap P$ from set P, and insert them into P^-.

(b) If $|P \cap Q| \le \lg n$, we define Q' to be $P \cap Q$. We have three possibilities in this case:

   – If adding the linear constraint $\sum_{x_i \in Q'} z_i \ge 1$ reduces the volume of LP by at least a multiplicative factor of n, we predict the correct label to be False. If this is the correct label, we perform no update operation. Otherwise the correct label is true, so we add the extra linear constraint $\sum_{x_i \in Q'} z_i \ge 1$.

   – If adding the linear constraint $\sum_{x_i \in Q'} z_i < 1$ reduces the volume of LP by at least a multiplicative factor of n, we predict the correct label to be True. If this is the correct label, we perform no update operation. Otherwise the correct label is False, so we add the extra linear constraint $\sum_{x_i \in Q'} z_i < 1$.

– If none of the above conditions hold (none of the two linear constraints are restrictive enough), we answer $\perp$. If the correct label is True, we add the extra linear constraint $\sum_{x_i \in Q'} z_i \geq 1$. Otherwise the correct label is False, so we add the extra linear constraint $\sum_{x_i \in Q'} z_i < 1$.

Any mistake for these type-b queries reduces the volume of LP by a factor of at least n, and any $\perp$ answer reduces the volume by a factor of $1/(1-1/n)$ (multiplying the volume by at most $1 - 1/n$).

To upper bound the number of mistakes and $\perp$ answers of our algorithm, we first prove bounds on the volume of the LP at different stages of the algorithm using the following lemma.

Lemma 4.1. *The volume of LP is $(1+\varepsilon)^n \leq 2^n$ at the start of the algorithm, it never increases, and it never falls below $\varepsilon^n = 1/(2 \lg n)^n$.*

Proof: The starting volume is $(1+\varepsilon)^n$ because we only have constraints $z_i \in [0, 1+\varepsilon]$ for each $1 \leq i \leq n$. The volume never increases because our only operation is to add extra constraints. The volume never goes below ε^n because the following set of points with volume ε^n are always feasible for LP.

We claim that LP has as a feasible solution every vector $z' = (z'_1, z'_2, \ldots, z'_n)$ with $z'_i \in [1, 1+\varepsilon]$ for every $x_i \in R$ and $z'_j \in [0, \varepsilon]$ for every $x_j \notin R$, where R is the set of relevant variables. We prove that z' satisfies all of the added constraints. We add a constraint $\sum_{x_i \in Q'} z_i \geq 1$ for a set Q' satisfying either $Q' \cap R \neq \emptyset$ or $Q' \cap R = \emptyset$. In the first case, there exists some variable $x_{i'} \in R \cap Q'$, and we have that $z'_{i'}$ is in range $[1, 1+\varepsilon]$, so z' satisfies this constraint. In the second case, we have that $\sum_{x_i \in Q'} z'_i \leq \varepsilon |Q'|$ because none of the variables in Q' is in R. On the other hand, the size of Q'' is at most $\lg n$ because we are in a type-b query. We conclude that the sum $\sum_{x_i \in Q''} z'_i$ is at most $\varepsilon \lg n = \frac{1}{2}$, which means that z' satisfies these kinds of constraints as well. □

Now we can bound the total number of mistakes and $\perp$ answers as follows.

Lemma 4.2. *The total number of mistakes is at most $n/\lg n + n(2 + \lg\lg n)/\lg n = O\left(n \frac{\log\log n}{\log n}\right)$, and the number of $\perp$ answers is at most $O(n^2 \log\log n)$.*

Proof: At first we find an upper bound for the number of mistakes. We have at most $n/\lg n$ type-a mistakes as we insert at least $\lg n$ variables into set P^- by each of these mistakes. Each type-b mistake reduces the volume of LP by a factor of n. By Lemma 4.1, the number of these mistakes should not be more than $\log_n \frac{2^n}{\varepsilon^n} = n \lg(2/\varepsilon)/\lg n = n(2 + \lg\lg n)/\lg n$, which completes the claim of the lemma about the number of mistakes.

For each $\perp$ answer, we add an extra constraint that chops off at least a $1/n$ fraction of the volume of LP, no matter what the correct label of the query is. This means that, after every n $\perp$ answers, the volume of LP is reduced by at least a factor of $(1 - 1/n)^n \leq 1/e$. So again, by Lemma 4.1, the total number of $\perp$ answers cannot be more than $n \log_e \frac{2^n}{\varepsilon^n} = O(n^2 \log\log n)$. □

Finally we can prove our main result of the section:

Proof of Theorem 4.1: The sample complexity bounds for the number of mistakes and $\perp$ answers are proved in Lemma 4.2. All the checks in our algorithm can be done in $O(n)$ time. To compute the fraction of volume eliminated by an extra linear constraint, we just need to sample $\lg n$ random points from the current LP via $\lg n$ calls to the Hit-and-Run algorithm in [LV06], which takes $\tilde{O}(n^3)$ amortized time. This completes the proof of our main result. □

5 Hardness Result

In this section, we prove that any algorithm for learning disjunction functions that makes at most a polynomial number of $\perp$ answers must make at least $\Omega(n/\log n)$ mistakes.

We use the following framework to construct hard instances in order to obtain this hardness result. Parameters C and k have constant values chosen below. We construct $m = n^k$ queries $Q_1, Q_2, \ldots, Q_m$ as follows. We use only the positive forms of the n variables $X = \{x_1, x_2, \ldots, x_n\}$ in our queries and in the disjunction function. To construct a query Q_i, $1 \leq i \leq m$, we sample $C \lg n$ variables uniformly at random from X with replacement. Let Q_i be the set of these $C \lg n$ samples, so that $|Q_i| \leq C \lg n$. (The size is smaller than $C \lg n$ when a variable gets sampled more than once.)

If the answer to query Q_i is implied from the correct answers to the previous queries $Q_1, Q_2, \ldots, Q_{i-1}$, then we have no choice but to evaluate the algorithm's answer by comparing it to the implied answer. Formally, if Q_i is a superset of one of the previous queries with label True, then the label of Q_i must also be True. On the other hand, if Q_i is a subset of the union of the previous queries with label False, then the label of Q_i must also be False. In all other cases, the correct label of query Q_i is not implied, i.e., it could be either True or False; then we wait for the algorithm to answer. If the algorithm answers $\perp$, we choose the correct label to be True. Otherwise, if the algorithm answers True, we choose the correct label to be False; and if the algorithm answers False, we choose the correct label of True; thus forcing the algorithm to make a mistake.

Now we prove that any algorithm must make either $\Omega(n/\log n)$ mistakes or a superpolynomial number of $\perp$ answers. Because our hardness result is information theoretic, we do not need to assume anything about the running time of the learning algorithm. Indeed, our results hold for randomized algorithms as well.

THEOREM 5.1. *For some $C > 10$, and for any $k \leq C/2$, any learning algorithm for disjunctions (allowing randomization and unbounded computational power), after facing the $m = n^k$ queries $Q_1, Q_2, \ldots, Q_m$ above, either makes more than $n/(10\,C \lg n)$ mistakes, or answers $\perp$ for at least n^{k-2} times, with probability $1 - 1/n^{\Theta(\log n)}$. In other words, any learning algorithm that makes $o(n/\log n)$ mistakes must answer $\perp$ a superpolynomial (more than $n^{k'}$ for any constant k') number of times in the worst case.*

In fact, we prove a stronger claim: the theorem holds even if we let the algorithm know our strategy, and inform the algorithm about all m queries $Q_1, Q_2, \ldots, Q_m$ in advance. Note that the algorithm must still work for every sequence of queries, i.e., it should work with probability 1 over the query distribution.

Let P^- be the set of variables that are not in the disjunction function based on the information the algorithm has. In other words, P^- is the union of all query sets whose correct label is False up to now. The set P^- is empty at the beginning, and it grows gradually.

We claim that P^- always has at most $n/10$ variables, e.g., after all m queries have been answered. The algorithm might predict the label True for query Q_i to make us say that the correct label of query Q_i is False. In this case, the algorithm makes a mistake and at most $C \lg n$ variables are added to P^-. The only other case in which we say that the correct label of Q_i is False is by implication, when Q_i is already a subset of P^-, in which case no variable gets added to P^-. So assuming the algorithm makes at most $n/(10\,C \lg n)$ mistakes, we can conclude that P^- has at most $n/10$ variables throughout.

Now we prove that, for many of these queries, the answer is not implied by the correct answers to the previous queries. Consequently, the algorithm should answer $\perp$ or make a mistake for many queries, which gives us the desired result. We use the following two lemmas:

LEMMA 5.1. *With probability $1 - 1/n^{\Theta(\log n)}$, there are not more than $\lg n$ variables in the intersection of two queries Q_i and Q_j for every pair of queries i and j.*

Proof: Note that there are at most $C \lg n$ variables in Q_i, so every time we sample a variable to add to query Q_j, this variable is also in query Q_i with probability $|Q_i|/n \leq (C \lg n)/n$. So the expected number of variables in both sets Q_i and Q_j is at most $(C^2 \lg^2 n)/n$. By Lemma B.1 in Appendix B, we can prove that the probability of event $|Q_i \cap Q_j| > \lg n$ is at most $e^{(\lg n)(1-\ln(n/(C^2 \lg n)))} < n^{-\ln(n/(C^2 \lg n))}$. On the other hand, there are $\binom{m}{2} < n^{2l}$ pairs of queries Q_i and Q_j. By a union bound, the probability of existing two queries Q_i and Q_j with $|Q_i \cap Q_j| > \lg n$ is at most $n^{2l} n^{-\ln(n/(C^2 \lg n))} = 1/n^{\Theta(\log n)} \leq 1/n^{\alpha}$ for any constant α. Because k, C, and α are just some constants, we can assume that n is sufficiently large so the desired inequalities hold. □

The property in the above lemma is independent of the algorithm's responses, so it holds with high probability anyway. The next lemma depends on the algorithm's choices, but we prove that it holds with high probability no matter what decisions the algorithm makes.

LEMMA 5.2. *The number of queries Q_i for which $|Q_i \setminus P^-| \leq \lg n$ is at most n/C with probability $1 - 1/n^{\Theta(n)}$ (no matter how the algorithm answers).*

Proof: The query set Q_i contains $C \lg n$ random samples from the variable set X. Because P^- has at most $n/10$ variables, each of these $C \lg n$ samples is also in P^- with probability at most $1/10$. So the expected size of $Q_i \cap P^-$ is at most $(C \lg n)/10$. So the probability of event $|Q_i \cap P^-| \geq (C-1) \lg n$ is less than $e^{0.8 C \lg n - 0.8 C \lg n \ln(1+8)} < e^{-0.95 C \lg n} < n^{-C}$ by Lemma B.1. There are $m = n^k$ queries so in expectation, there are $n^{k-C} \leq n^{-C/2}$ queries Q_i with $|Q_i \setminus P^-| \leq \lg n$. The probability that there are more than n/C queries with this property is at most $e^{(n/C)(1-\ln(1+(n/C)/n^{-C/2}))} < e^{(n/C)(-C/2)\ln n} = n^{-n/2}$ again by Lemma B.1. So the probability that the number of violating queries exceeds n/C is extremely small. But the algorithm can choose different answers to reach different choices for set P^-. We note that P^- has at most $n/10$ variables so there are less than $n^{n/10}$ choices for P^-. By a union bound, we conclude that the number of violating queries is at most n/C with probability at least $1 - n^{-n/2+n/10} \geq 1 - n^{-0.4n}$ for any set of decisions the algorithm makes. □

Finally we have all the building blocks needed to prove Theorem 5.1:

Proof of Theorem 5.1: If the answer to query Q_i cannot be implied based on the previous queries and their correct answers, the algorithm either makes a mistake or answers $\perp$. So we just need to prove that for many of the input queries, the correct label of the query cannot be implied. We can imply the label of Q_i is False if Q_i is a subset of P^-. In this case, $|Q_i \setminus P^-|$ should be equal to 0. Using Lemma 5.2, we know that the number of these queries is at most $n/C \ll m = n^k$.

We are also able to imply the correct label of Q_i when there exists a previous query Q_j whose correct label is True, and $Q_j \setminus P^-$ is a subset of Q_i. First of all, we note that $Q_j \setminus P^-$ cannot be a subset of Q_i, if there are more than $\lg n$ variables in $Q_j \setminus P^-$. Because there would be more than $\lg n$ variables in $Q_i \cap Q_j$ in this case which contradicts Lemma 5.1.

So Q_j is one of at most n/C violating queries of Lemma 5.2. For each Q_j with $0 < |Q_j \setminus P^-| \leq \lg n$, we pick a variable in $Q_j \setminus P^-$ arbitrarily and put it in set T. So there are at most n/C variables in set T. We note that if query Q_i has empty intersection with set T, its label cannot be implied. According to the way we make Q_i, the probability that Q_i does not intersect T is at least $(1 - 1/C)^{C \lg n} \geq n^{-1.5}$. So in expectation, there are at least $m \cdot n^{-1.5} - n/C = n^{k-1.5} - n/C$ queries whose labels cannot be implied, and with high probability the number of such queries is not less than $n^{k-2} + n/(10\, C \lg n)$.

We formalize this argument as follows. For large enough n and k, we know that $n^{k-1.5} - n/C$ is at least $n^{k-1.5}/2$, and $n^{k-2} + n/(10\, C \lg n)$ is at most $(4/n) \cdot n^{k-1.5}/2$. By applying Lemma B.1, we can prove that the probability that the number of such queries is less than $n^{k-2} + n/(10\, C \lg n)$ is upper bounded by

$$e^{-(1-4/n)^2 \cdot n^{k-1.5}/2} \leq e^{-\Theta(n)}.$$

To conclude all the error probability arguments, we remind that the claims of Lemmas 5.1, 5.2, and the above claim hold with probabilities $1 - 1/n^{\Theta(\log n)}$, $1 - 1/n^{\Theta(n)}$, and $1 - 1/e^{\Theta(n)}$ respectively. Using a union bound, we conclude that the claim of this theorem holds with probability $1 - 1/n^{\Theta(\log n)}$. In other words, the number of $\perp$ answers should be at least n^{k-2} unless the algorithm wants to make more than $n/(10\, C \lg n)$ mistakes. $\square$

6 Future Directions

I want to highlight three possible streams of future work:

- What is the optimal solution? There are three main factors in algorithms for learning disjunction functions studied here:
  - Mistake Bound: the maximum total number of mistakes the algorithm makes.
  - $\perp$ Bound: the maximum number of times the algorithm answers with $\perp$.
  - Query Complexity: The running time per answer.

  It would be ideal to find an algorithm with $O(n/\log(n))$ mistakes, $O(n)$ $\perp$ answers, and $O(n)$ running time per query. But is it possible? Even finding a polynomial time (per query) algorithm with polynomial number of $\perp$ answers and mistake bound $O(n/\log(n))$ seems very challenging and interesting. One possible approach is to construct and update a Markov Chain on the set of target functions consistent up to now during the algorithm that has two main properties. Its steady-state distribution is close to uniform, i.e. the ratio of maximum and minimum probability is at most a polynomial in n, the number of variables. It has a polynomial mixing time. This way one can adapt the idea of halving algorithm, and achieves the desired optimal mistake bound.

- Studying other learning problems in this model: It would be interesting to see how much one can reduce the number of mistakes in learning other concept classes using a polynomial number of $\perp$ answers. One good candidate is the set of linear threshold functions which are a generalization of disjunctions.

- Alternatives and Generalizations of the model: The interesting part of this model is to allow answering with different levels of confidence. In this model, we have two extreme ends, the algorithm either answers and make a mistake in the worst case, or skips the query and answers with $\perp$. We could allow the algorithm to choose between several levels of confidence (instead of two) and answer. Making a mistake in each level can have its own cost.

7 Acknowledgement

The second author wants to thank Avrim Blum and Adam Tauman Kalai for the helpful and insightful discussions throughout this work.

References

[AS00] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley, 2000.

[CBGZ06] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Worst-case analysis of selective sampling for linear classification. *J. Mach. Learn. Res.*, 7:1205–1230, December 2006.

[CBO11] Nicolò Cesa-Bianchi and Francesco Orabona. Better algorithms for selective sampling. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 433–440, New York, NY, USA, June 2011. ACM.

[DJ03] Robert P. W. Duin and Piotr Juszczak. Selective sampling methods in one-class classification problems. In *Proceedings of the 2003 joint international conference on Artificial neural networks and neural information processing*, ICANN/ICONIP'03, pages 140–148, Berlin, Heidelberg, 2003. Springer-Verlag.

[FSST97] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Mach. Learn.*, 28:133–168, September 1997.

[HP97] D. Helmbold and S. Panizza. Some label efficient learning results. In *Proceedings of the tenth annual conference on Computational learning theory*, pages 218–230. ACM, 1997.

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning*, 2(4):285–318, 1988.

[LLW08] L. Li, M.L. Littman, and T.J. Walsh. Knows what it knows: a framework for self-aware learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 568–575. ACM, 2008.

[LV06] László Lovász and Santosh Vempala. Hit-and-run from a corner. *SIAM J. Comput.*, 35(4):985–1005, 2006.

[RS88] R.L. Rivest and R. Sloan. Learning complicated concepts reliably and usefully. In *Proceedings AAAI-88*, pages 635–639, 1988.

[SZB10] Amin Sayedi, Morteza Zadimoghaddam, and Avrim Blum. Trading off mistakes and don't-know predictions. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Proceedings of the 24th Advances in Neural Information Processing Systems (NIPS 2010)*, pages 2092–2100, Vancouver, Canada, 2010.

A More Related Work

The idea of having both mistakes and $\perp$ answers was investigated by Helmbold and Panizza [HP97] before the KWIK framework was introduced. They allow the algorithm to make some mistakes, and also request some labels (in our case the answer to some queries). They study the trade-off between these two types of costs. An $\perp$ answer can also be seen as requesting the answer to the query. Rivest and Sloan [RS88] also studied a framework in which the learner algorithm has to make accurate predictions or says $\perp$. In fact, reliable learning in that setting means that the algorithm cannot make any wrong predictions which is similar to the KWIK model.

It is also worth noting that there exists some interesting connection between our setting and Selective Sampling methods used in Active Learning. Selective sampling, as an active learning method, reduces the cost of labeling supplementary training data by asking only for the labels of the most informative, unlabeled examples [DJ03]. In particular, if the algorithm is very unsure about the label of an example, it means that the example and its label are very informative, and therefore the selective sampling algorithm asks for its label. In our framework, a learning algorithm should detect these kinds of informative examples, and for them it should answer $\perp$. Because no matter what label we receive for the example, we will gain lots of information (and there is no need to risk to make a mistake). On the other hand, if the algorithm is pretty sure about the label of an example, the example is not very informative, and therefore in selective sampling the algorithm does not ask for the label. In our setting, for these examples, the algorithm should answer with the label we are pretty sure about, and in case we make a mistake, we will receive a huge amount of information. For more works on selective sampling algorithms, we refer to [CBGZ06, CBO11, FSST97] .

B Concentration Bounds

We need the following version of Chernoff bound as appears in [AS00, page 267, Corollary A.1.10 and Theorem A.1.13]. We change the notation a bit to avoid confusion with our notation.

LEMMA B.1. *Suppose $Z_1, Z_2, \ldots, Z_n$ are independent binary random variables, such that $\Pr\{Z_i = 1\} = p_i$. Let $\mu = \sum_{i=1}^n p_i$, and $Z = \sum_{i=1}^n Z_i$. For any $a > 0$, we have that*

$$\begin{aligned} & \Pr\{Z - \mu \geq a\} \leq e^{a-(a+\mu)\ln(1+a/\mu)} \\ \textit{and} \quad & \Pr\{Z - \mu \leq -a\} \leq e^{-a^2/\mu}. \end{aligned}$$