Trading off Mistakes and Don't-Know Predictions: Combining KWIK and Mistake Bound Models

Amin Sayedi[*]
Tepper School of Business
CMU
Pittsburgh, PA 15213
ssayedir@cmu.edu

Avrim Blum
Department of Computer Science
CMU
Pittsburgh, PA 15213
avrim@cs.cmu.edu

Morteza Zadimoghaddam
CSAIL
MIT
Cambridge, MA 02139
morteza@mit.edu

Abstract

We discuss an online learning framework in which the agent is allowed to say "I don't know" as well as making wrong predictions on given examples. We analyze the trade off between saying "I don't know" and making mistakes. If the number of don't know predictions is forced to be zero, the model reduces to the famous mistake-bound model introduced by Littlestone [Lit88]. On the other hand, if no mistake is allowed, the model reduces to KWIK framework introduced by Li et. al. [LLW08]. We propose a general, though exponential-time, algorithm that minimizes the number of don't-know predictions if a certain number of mistakes is allowed. Polynomial-time versions of our algorithm are presented for concept classes monotone disjunctions and linear separators.

1 Introduction

Motivated by [KS02, KK99] among others, Li, Littman and Walsh [LLW08] introduced the KWIK, standing for *knows what it knows*, framework for online learning. Roughly stated, in KWIK model, the learning algorithm needs to make only accurate predictions, although it can opt out of predictions by saying "I don't know"($\perp$). The algorithm is not allowed to make any mistake, still, it learns from those examples on which it answers $\perp$. The goal of the algorithm is to minimize the number of examples on which it answers $\perp$. Several aspects of the model are discussed in [LLW08], and there are many other papers, including [WSDL, DLL09, SL08], using the framework. It is worth mentioning that the idea of forcing the algorithm to say "I don't know" instead of making mistake has also appeared in some earlier work like [RS88], referred to as *reliable learning*.

Generally, it is highly desirable to have an algorithm that learns a concept in KWIK framework using a few, or even polynomial, number of $\perp$s. But unfortunately, for many concepts, no such algorithm exists. In fact, it turns out that even for many basic concepts which are very easy to learn in Mistake-bound model [Lit88], e.g. class of $[0, a]-$intervals or singletons, the KWIK algorithm needs to say $\perp$ exponentially many times. The purpose of our paper is to relax the assumption of *not making any mistake*, by allowing a few mistakes, but instead get much better bounds on the number of $\perp$s.

In [LLW08], the authors show, through a non-polynomial time algorithms called *enumeration algorithm*, that a finite class H of functions can be learned in KWIK framework with at most $|H| - 1$ number of $\perp$s. We show that if only one mistake is allowed, that number can be reduced to $\sqrt{2|H|}$. Furthermore, we show that the problem is equivalent to the famous egg-dropping puzzle, defined formally in section 2, hence getting bound $(k + 1)H^{\frac{1}{k+1}}$ when k mistakes are allowed. Our algorithm does not run in polynomial time in general since its running time depends on $|H|$; however,

[*]Part of this work was done when the author was an intern in Microsoft Research, Cambridge, MA.

we propose polynomial versions of our algorithm for two important concepts monotone disjunctions and linear separators.

Allowing the algorithm to make mistakes in KWIK model, is equivalent to allowing the algorithm to say "I don't know" in Mistake-bound model introduced in [Lit88]. In fact, one way of looking at the algorithms presented in section 3 is that we want to decrease the number of mistakes in Mistake-bound model by allowing the algorithm to say $\perp$. The rest of the paper is structured as follows. First we define the model and describe the limits of KWIK model. Then in section 2, we describe how would the bounds on the number of $\perp$s change if we allow a few mistakes in KWIK model. Finally, we give two polynomial algorithms for important classes, Monotone Disjunctions and Linear Separators in section 3.

1.1 Model

We want to learn a hypothesis class H consisting of functions $f : X \to \{+, -\}$. In each stage, the algorithm is given $x \in X$ and is asked to predict $h^*(x)$. The algorithm might answer, $+$, $-$ or $\perp$ representing "I don't know". After the prediction, even if it is $\perp$, the value of $h^*(x)$ is revealed to the algorithm. For a given integer k, we want to design an algorithm which, for any sequence of examples, the number of times that it makes a mistake, denoted by M, is not more than k, and the number of times that it answers $\perp$, denoted by I, is minimized.

For example, the special case of $k = 0$ is equivalent to the KWIK framework, or for a finite class H, if $k \geq \log(|H|)$, majority vote algorithm can learn the class with no $\perp$, i.e. $I = 0$.

Since we want to derive worst-case bounds, we assume that the sequence of the examples, as well as the target function h^* are selected by an adversary. The adversary sends the examples one by one. For each example $x \in X$, our algorithm should decide what to answer; then, the adversary reveals $h^*(x)$.

1.2 About KWIK Model

Although the idea of KWIK framework is quite useful, there are very few problems that can be solved effectively in this framework. The following example demonstrates how an easy problem in Mistake-bound model can turn into a hard problem in KWIK model.

Example 1 *Suppose that H is the class of singletons. In other words, for $h_i \in H$, where $h_i : \{0,1\}^n \to \{-, +\}$, we have $h_i(x) = +$ if x is the binary representation of i, and $h_i(x) = -$ otherwise. Class H can be learned in Mistake-bound model with mistake bound of only* 1. *The algorithm simply predicts $-$ on all examples until it makes a mistake. As soon as the algorithm makes a mistake, it can easily figure out what the target function is.*

However, class H needs exponentially many $\perp$'s in the KWIK framework to be learned. Since the algorithm does not know the answer, it must keep answering $\perp$ on all examples that it has not seen yet. Therefore, in the worst case, it answers $\perp$ and finds out that the answer is $-$ on all the first $2^n - 1$ examples that it sees.

The situation in Example 1 happens in many other classes of functions, e.g. conjunctions or disjunctions as well.

Next, we review an algorithm (called *enumeration algorithm* in [LLW08]) for solving problems in KWIK framework. This algorithm is the main ingredient of most of the algorithms proposed in [LLW08].

Algorithm 1 ***Enumeration***

The algorithm looks at all the functions in class H; if they all agree on the label of the current example $x \in X$, the algorithm outputs that label, otherwise the algorithm outputs $\perp$. The algorithm then removes those functions $h \in H$ who answered incorrectly on x from H and continues receiving the next example. Note that at least one function gets removed from H each time that algorithm answers $\perp$; therefore, the algorithm finds the target function with at most $|H| - 1$ number of $\perp$'s.

2 KWIK Model with Mistake

Example 1 shows how hard it can be to learn in KWIK model. For that, we give the following relaxation of the framework that allows more concepts to be learned and at the same time preserves the original motivation of the KWIK—it's better saying I don't know rather than making a mistake.

We allow the algorithm to make at most k mistakes. This lets us to get better bounds on the number of times that the algorithm answers $\perp$. For example, by letting $k = 1$, i.e. allowing one mistake, the number of $\perp$'s decreases from $2^n - 1$ to 0 for the class of singletons. Next, we will give an algorithm for learning in KWIK framework where the algorithm is allowed to make k mistakes. Later, we look at the problem from another angle; this time we allow "I don't know"s in Mistake-bound model.

We saw, in Algorithm 1, how to learn a concept class H with no mistake and with $O(|H|)$ number of $\perp$'s. In many cases, the bound $O(|H|)$ is tight; in fact, if for every subset $S \subseteq H$ with $|S| > 1$ there exists some $x \in X$ for which $|\{h \in S | h(x) = +\}| \in \{1, |S| - 1\}$, then the bound is tight. This condition, for example, is easily satisfied for the class of $[0, a]$-intervals.

However, if we allow the algorithm to make one mistake, we show that the number of $\perp$'s can be reduced to $O(\sqrt{|H|})$. In general, if k mistakes are allowed, there is an algorithm which can learn the class with at most $(k+1)|H|^{1/k+1}$ number of $\perp$'s. The algorithm is similar to the one for the classic "egg game" puzzle (See [GF]). First suppose that $k = 1$. We make a pool of all functions in H, initially consisting of $|H|$ candidates. Whenever an example arrives, we see how many of the candidates label it $+$, and how many label it $-$. If the population of the minority is $< \sqrt{|H|}$, we predict the label that the majority says on the example; however, if the population of the minority is $\geq \sqrt{|H|}$, we say $\perp$. Those function who have been wrong on an example will be removed from the pool in each step. If we make a mistake in some step, the size of the population will reduce to $< \sqrt{|H|}$. Hence, using Algorithm 1, we can complete the learning with at most $\sqrt{|H|}$ number of $\perp$'s after our first mistake. Furthermore, note that before making any mistake, we remove at least $\sqrt{|H|}$ of the functions from the pool each time we answer $\perp$. Therefore, the total number of $\perp$'s cannot exceed $2\sqrt{|H|}$. This technique can be generalized for k mistakes, but before we present a proof for the case of k mistakes, we like to mention a nice connection between this problem and the classic "egg game" puzzle.

Example 2 ***Egg Game Puzzle***

You are given 2 identical eggs, and you have access to a n-storey building. The eggs can be very hard or very fragile, i.e. they may break if dropped from the first floor or may not break even if dropped from the n-th floor. You need to figure out the highest floor an egg can be dropped from without breaking. The question is how many drops you need to make. Note that you can break only two eggs in the process.

The answer to this puzzle is $\sqrt{2n}$ up-to some additive constant. In fact, a thorough analysis of the puzzle when there are e eggs available, instead of just two eggs, is given in [GF]. It is not hard to see that $\perp$ minimization problem when k mistakes are allowed is equivalent to the egg game puzzle when the building has $|H|$ floors and there are $k + 1$ eggs available. As a result, with a slightly smarter algorithm which adjusts the threshold $\sqrt{|H|}$ recursively each time an example arrives, we can decrease the number of $\perp$s from $2\sqrt{|H|}$ to $\sqrt{2|H|}$.

Algorithm 2 ***Learning in KWIK Model with at most k Mistakes***

Let $s = |H|^{\frac{k}{k+1}}$, and let P be a pool for all functions that might be the target; initially $P = H$, but during the learning process, we remove functions from P. For each example that arrives, we see how the functions in P label it (some of them label it $+$, and some label it $-$). If the minority population is $> s$, we answer $\perp$, otherwise, we answer what the majority says. At the end of each step, we remove the functions that made a mistake in the last step from P. Whenever we make a mistake, we update $s = |P|^{\frac{k-i}{k+1-i}}$, where i is the number of mistakes we have made so far.

Proposition 1 *Algorithm 2 learns a concept class H with at most k mistakes and $(k+1)|H|^{1/k+1}$ "I don't know"s.*

Proof: After our first mistake, the size of the pool reduces to $< |H|^{\frac{k}{k+1}}$. Hence, using induction, we can argue that after the first mistake, the learning can be done with $k-1$ mistakes and $k(|H|^{\frac{k}{k+1}})^{1/k}$ "I don't know"s. There can exist at most $\frac{|H|}{|H|^{\frac{k}{k+1}}} = |H|^{1/k+1}$ number of $\perp$'s before the first mistake, therefore, the total number of $\perp$'s will not exceed

$$|H|^{1/k+1} + k(|H|^{\frac{k}{k+1}})^{1/k} = (k+1)|H|^{1/k+1}.$$

$\square$

Before moving to the next section, we should mention that Algorithm 2 is not computationally efficient. Particularly, if H contains exponentially many functions, which is the case most of the times, the running time of Algorithm 2 becomes exponential. In the next section, we give polynomial-time algorithm for a couple of important concept classes.

3 Mistake Bound Model with "I don't know"

We can look at the problem from another perspective: instead of adding mistakes to the KWIK framework, we can add "I don't know" to the Mistake-bound model. We prefer our algorithm saying "I don't know" rather than making a mistake. Therefore, in this section, we try improve the mistake bounds in Mistake-bound model by allowing the algorithm to say "I don't know" in some cases. Of course, if the algorithm always answers $\perp$, it makes no mistake; so, we are particularly interested in the tradeoff between the number of mistakes and the number of $\perp$'s. Please note that an algorithm can always replace its $\perp$'s with random $+$'s and $-$'s, therefore, we must expect that decreasing the number of mistakes by one requires increasing the number of $\perp$'s by at least one.

3.1 Monotone Disjunctions

We start with the concept class *Monotone Disjunctions*. A monotone disjunction is a disjunction in which no literal appears negated, that is, a function of the form

$$f(x_1, \ldots, x_n) = x_{i_1} \vee x_{i_2} \vee \ldots \vee x_{i_k}.$$

We know that this class can be learned with at most n mistakes in Mistake-bound Model [Lit88] where n is the total number of variables. This class is particularly interesting because the results derived about Monotone disjunctions can be applied to Monotone Conjunctions as well. Furthermore, by defining n new variables, we can extend the results to the class of non-monotone Disjunctions and non-monotone Conjunctions as well. We are interested in decreasing the number of mistakes for the cost of having (hopefully few) number of $\perp$'s. Suppose that we know that the target function is a monotone disjunction of a subset of the n variables $\{x_1, \ldots, x_n\}$. Each example is a boolean vector of length n. An example is labeled $+$, if and only if at least one of the variables that belong to the target function are set to 1 in the example.

First, let's not worry about the running time and see how well Algorithm 2 performs here. We have $|H| = 2^n$; if we let $k = n/i$, the bound that we get on the number of $\perp$'s will be $\simeq \frac{n2^i}{i}$; this is not bad, especially, for the case of small i, e.g. $i = 2, 3$. In fact, for the case of $i = 2$, we are trading off each mistake for four "I don't know"s. But unfortunately, Algorithm 2 cannot do this in polynomial time. Our next goal is to design an algorithm which runs in polynomial time and guarantees the same good bounds on the number of $\perp$'s.

Algorithm 3 ***Learning Monotone Disjunctions with at most*** $n/2$ ***Mistakes***

Let P, P^+ and P^- be three pools of variables. Initially, $P = \{x_1, \ldots, x_n\}$ and $P^+ = P^- = \phi$. During the process of learning, the variables will be moved from P to P^- or P^+. The pool P^+ is the set of variables that we know must exist in the target function; the pool P^- is the set of the variables that we know cannot exist in the target function. The learning process finishes by the time that P gets empty.

In each step, an example $a \in \{0,1\}^n$ arrives. Let $S \subseteq \{x_1, \ldots, x_n\}$ be the set representation of a, i.e., $x_i \in S$ if and only if $a_i = 1$. If $S \cap P^+ \neq \phi$, we can say for sure that the example is $+$. If $S \subseteq P^-$, we can say for sure that the example is negative. Otherwise, it must be the case *that $S \cap P \neq \phi$, and we cannot be sure about our prediction. Here, if $|S \cap P| \geq 2$ we answer $+$, otherwise, i.e. if $|S \cap P| = 1$, we answer $\perp$.*

If we make a mistake, we move $S \cap P$ to P^-. Every time we answer $\perp$, we move $S \cap P$ to P^+ or P^- depending on the correct label of the example.

Proposition 2 *Algorithm 3 learns the class of Monotone Disjunction with at most $M \leq n/2$ mistake and $n - 2M$ number of $\perp$s.*

Proof: If we make a mistake, it must be the case that the answer had been $-$ while we answered $+$; for this to happen, we must have $|S \cap P| \geq 2$. So, after a mistake, we can move $S \cap P$ to P^-. The size of P, hence, decreases by at least 2.

Every time we say $\perp$, it must be the case that $|S \cap P| = 1$. The size of P decreases by at least one, each time. $\square$

Algorithm 3, although very simple, has an interesting property. If in an online learning setting, saying $\perp$ is cheaper than making a mistake, Algorithm 3 strictly dominates the best algorithm in Mistake-bound model. Note that the sum of its $\perp$s and its mistakes is never more than n. More precisely, if the cost of making a mistake is 1 and the cost of saying $\perp$ is < 1, the worst-case cost of this algorithm is strictly smaller than n.

Next we present an algorithm for decreasing the number of mistakes to $n/3$.

Algorithm 4 ***Learning Monotone Disjunctions with at most $n/3$ Mistakes***

Let P, P^+, P^- be defined as in Algorithm 3. We have another pool P' which consists of pairs of variables such that for each pair we know at least one of the variables belongs to the target function. Like before, the pools form a partition over the set of all variables. Please note that a variable can belong to at most one pair in P', furthermore, if a variable is in some pair in P', it cannot belong to any of the other sets P, P^+ or P^-.

Whenever an example $a \in \{0,1\}^n$ arrives we do the following. Let $S \subseteq \{x_1, \ldots, x_n\}$ be the set representation of a, i.e. $x_i \in S$ if and only if $a_i = 1$. If $S \cap P^+ \neq \phi$, we answer $+$. If $S \subseteq P^-$, we answer $-$. Also, if S contains both members of a pair in P', we can say that the label is $+$.

If none of the above cases happen, we cannot be sure about our prediction. In this case, if $|S \cap P| \geq 3$, we answer $+$. If $|S \cap (P \cup P')| \geq 2$ and $|S \cap P'| \geq 1$ we again answer $+$. Otherwise, we answer $\perp$.

Proposition 3 *Algorithm 4 learns the class of Monotone Disjunction with at most $M \leq n/3$ mistake and $3n/2 - 3M$ number of $\perp$'s.*

Proof: If $|S \cap P| \geq 3$ and we make a mistake on S, then the size of P will be reduced by at least 3, and the size of P^- will increase by at least 3. If $|S \cap (P \cup P')| \geq 2$ and $|S \cap P'| \geq 1$ and we make a mistake on S, then at least two variables will be moved from $(P' \cup P)$ to P^-, and at least one variable will be moved from P' to P^+ (since whenever a variable moves from P' to P^-, the other variable in its pair should move to P^+). Therefore, the size of $P^- \cup P^+$ will increase by at least 3. Since $P^- \cup P^+ \leq n$, we will not make more than $n/3$ mistakes.

There are three cases in which we may answer $\perp$. If $|S \cap P| = 0$ and $|S \cap P'| = 1$, we answer $\perp$; however, after knowing the correct label, $S \cap P'$ will be moved to P^+ or P^-. Therefore, the number of "I don't know"s of this type is bounded by $n - 3M$. If $|S \cap P| = 1$ and $|S \cap P'| = 0$, again, after knowing the correct label, $S \cap P$ will be moved to P^+ or P^-, so the same bound applies. If $|S \cap P| = 2$ and $|S \cap P'| = 0$, the correct label might be $+$ or $-$. If it is negative, then we can move $S \cap P$ to P^- and use the same bound as before. If it is $+$, the two variables in $S \cap P$ will be moved to P' as a pair. Note that there can be at most $n/2$ of such $\perp$'s; therefore, the total number of $\perp$'s cannot exceed $n/2 + n - 3M$. $\square$

3.2 Learning Linear Separator Functions

In this section, we analyze how we can use $\perp$ to decrease the number of mistakes for learning linear separators with margin γ. This is a good example to show that the idea applied to finite H can still be applied even when $|H|$ is infinite by looking at the measure of those functions who predict $+$ and those who predict $-$.

Definition 1 *A sequence S of n d-dimensional points arrive one by one. There are two types of points $+$ points and $-$ points. We want to find a separator hyperplane (vector) to distinguish them. We know that there exists a unit-length separator vector w^* such that $w^* \cdot x > 0$ if and only if x is a $+$ point. Define γ to be $min_{x \in S} \frac{w^* \cdot x}{|x|}$. It is also assumed that all points have unit length.*

The parameter γ in the above definition shows how well the vector w^* separates the points in sequence S. Following we show how to formulate the problem with a Linear Program to bound the number of mistakes using some "I don't know" answers.

Assume that n points $x_1, x_2, \cdots, x_n$ are in the sequence S. Each point is either a $+$ point or a $-$ point. We note that these points arrive one by one, and we do not have them in advance. We have to answer when a point arrives. The objective is to make a small number of mistakes and some "I don't know" answers to find a separation vector w such that $w \cdot x_i$ is positive if and only if x_i is a $+$ point.

We can write this problem as the following linear program.

$$w \cdot x_i > 0 \text{ If } x_i \text{ is a } + \text{ instance, and}$$

$$w \cdot x_i \leq 0 \text{ If } x_i \text{ is a } - \text{ instance}$$

Note that there are d variables which are the coordinates of vector w, and there are n linear constraints one per input point. Clearly we do not know which points are the $+$ points, so we can not write this linear program explicitly and solve it. But the points arrive one by one and the constraints of this program are revealed over the time. Note that if a vector w is a feasible solution of the above linear program, any positive multiple of w is also a feasible solution. In order to make the analysis easier and bound the core (the set of feasible solutions of the linear program), we can assume that the coordinates of the vector w are always in range $[-1 - \gamma/\sqrt{d}, 1 + \gamma/\sqrt{d}]$. We can add $2d$ linear constraints to make sure that the coordinates do not violate these properties. We will see later why we are choosing the bounds to be $-(1 + \gamma/\sqrt{d})$ and $1 + \gamma/\sqrt{d}$.

Now assume that we are at the beginning and no point has arrived. So we do not have any of the n constraints related to points. The core of the linear program is the set of vectors w in $[-1 - \gamma/\sqrt{d}, 1 + \gamma/\sqrt{d}]^d$ at the beginning. So we have a core (feasible set) of volume $(2 + 2\gamma/\sqrt{d})^d$ at first. For now assume that we can not use the "I don't know" answers. We show how to use them later. Then the first point arrives. There are two possibilities for this point. It is either a $+$ point or a $-$ point. If we add any of these two constraints to the linear program, we obtain a more restricted linear program with probably a less volume core. So we obtain one LP for each of these two possibilities, and interestingly the sum of the volumes of these two linear programs is equal to the volume of our current linear program. We will show how to compute these volumes, but for now assume that they are computed. If the volume of the linear program for the $+$ case is larger than the $-$ case, we answer $+$. If our answer is true, we are fine, and we have passed the query with no mistake. Otherwise we have made a mistake, but the volume of the core of our linear program is halved. We do the same for the $-$ case as well, i.e. we answer $-$ when the larger volume is for $-$ case.

Now there are two main issues we have to deal with. First of all, we have to find a way to compute the volume of the core of a linear program. Secondly, we have to find a way to bound the number of mistakes.

In fact computing the volume of a linear program is $\#P$-hard [DF88]. Interestingly, there is a randomized polynomial time algorithm that approximates the volume of the core of a linear program with $(1 + \epsilon)$ approximation [DFK91], i.e. the relative error is ϵ. The running time of their algorithm is polynomial in n, d, and $1/\epsilon$.

So assume that l points have arrived, and we answered them. Now the $l + 1$th point has arrived, and we have to answer. We have the linear program with the first l constraints. We compute the volume of its core, and name it V. We also write the two linear programs for the two possibilities of the $l+1$th constraint. We compute their volumes as well, name them V_1 and V_2. So we have these three volumes with relative errors at most ϵ for arbitrary constant ϵ. If V_1 is bigger than V_2, we answer $+$ (V_1 is the volume of the $+$ linear program). Otherwise, we answer $-$. The next lemma shows that for any mistake we make the volume decreases by a constant factor.

Lemma 4 *For any mistake, we make in our algorithm, the volume of the core of the linear program decreases by a factor of* $(1+\epsilon)/2$.

Proof: Without loss of generality, assume that we answered $+$, but the correct answer was $-$. Let V', V_1', and V_2' be the exact volumes of the three linear programs. So V' should be equal to $V_1' + V_2'$. So V_1 should be in range $[(1-\epsilon)V_1', (1+\epsilon)V_1']$, and V_2 should be in range $[(1-\epsilon)V_2', (1+\epsilon)V_2']$ because the algorithm for computing volumes has relative error at most ϵ. We answered $+$ because V_1 is at least V_2. We also have that: V_1' is at least $V_1/(1+\epsilon)$, and V_2 is at least $(1-\epsilon)V_2'$. Therefore V_1' is at least $\frac{(1-\epsilon)V_2'}{(1+\epsilon)}$. Since V' is equal to $V_1' + V_2'$, we have that $V_1' + \frac{(1-\epsilon)V_1'}{(1+\epsilon)}$ is at least $\frac{(1-\epsilon)V_2'}{(1+\epsilon)} + \frac{(1-\epsilon)V_1'}{(1+\epsilon)}$ which is equal to $\frac{(1-\epsilon)V'}{(1+\epsilon)}$. We conclude that V_1' is at least $\frac{(1-\epsilon)V'}{2}$. Since our $+$ answer was wrong, at least $(1-\epsilon)/2$ portion of the core is deleted, and we conclude that the volume of the core is multiplied by $(1+\epsilon)/2$ or a smaller number. $\square$

Now we show that the core of the linear program after adding all n constraints (the constraints of the variables) has a descent volume in terms of γ.

Lemma 5 *If there is a unit-length separator vector w^* with $min_{x \in S} \frac{w^* \cdot x}{|x|} = \gamma$, the core of the complete linear program after adding all n constraints of the points has volume at least* $(\gamma/\sqrt{d})^d$.

Proof: Clearly w^* is in the core of our linear program. Consider a vector w' whose all coordinates are in range $(-\gamma/\sqrt{d}, \gamma/\sqrt{d})$. We claim that $(w^* + w')$ is a correct separator. Consider a point x_i. Without loss of generality assume that it is a $+$ point. So $w^* \cdot x_i$ is at least γ. We also know that $|w' \cdot x_i|$ is at most $|w'|$ because the point x_i has unit length. We know that $|w'|$ is less than γ because all its d coordinates are in range $(-\gamma/\sqrt{d}, \gamma/\sqrt{d})$. So $(w^* + w') \cdot x_i = w^* \cdot x_i + w' \cdot x_i$ is positive. We also know that the coordinates of $w^* + w'$ are in range $(-1-\gamma/\sqrt{d}, 1+\gamma/\sqrt{d})$ because w^* has unit length (so all its coordinates are between -1 and 1), and the coordinates of w' are in range $(-\gamma/\sqrt{d}, \gamma/\sqrt{d})$. So all vectors of form $w^* + w'$ are in the core. Therefore the volume of the core is at least $(2\gamma/\sqrt{d})^d$. $\square$

Now we can bound the number of mistakes.

Theorem 6 *The total number of mistakes in the above algorithm is not more than* $\log_{2/(1+\epsilon)} \frac{(2+2\gamma/\sqrt{d})^d}{(2\gamma/\sqrt{d})^d} = \log_{2/(1+\epsilon)} \frac{(1+\gamma/\sqrt{d})^d}{(\gamma/\sqrt{d})^d} = O(d(\log d + \log 1/\gamma))$.

Proof: It can be easily proved using the Lemmas 4 and 5. $\square$

Now we make use of the "I don't know" answers to reduce the number of mistakes to any number we want. Assume that we do not want to make more than k mistakes. Define Y_1 to be $(2+2\gamma/\sqrt{d})^d$ which is the volume of the core at the beginning before adding any of the constraints of the points. Define Y_2 to be $(2\gamma/\sqrt{d})^d$ which is a lower bound for the volume of the core after adding all the constraints of the points. Let R to be the ratio $\frac{Y_2}{Y_1}$. In the above algorithm, we do not make more than $\log_{2/(1+\epsilon)} R$ mistakes.

We want to answer "I don't know" sometimes to reduce this number of mistakes. Define C to be $R^{1/k}$. Now we do the following. When the next point arrives. We compute the three volumes V, V_1, and V_2 (the estimates). We know that there is a relative error of ϵ between them and their exact values V', V_1', and V_2'. Based on the three estimates V, V_1, and V_2, if we can imply that $V_1' \leq V'/C$, we answer $-$. If our answer is wrong, the volume of the core is decreased by a factor of C. So we do not make more than k mistakes. Because C is defined to be $R^{1/k}$, and we can not reduce the volume of the core by a factor of more than R in total (in all mistakes together). If we can imply that V_2' is at most V'/C, we answer $+$. We have the same analysis for this case as well. Otherwise we answer "I don't know". We bound the number of "I don't know" answers as follows. We note that V_1' is at most $V_1/(1-\epsilon)$, and V' is at least $V/(1+\epsilon)$. So $\frac{V_1'}{V'}$ is at most $\frac{V_1}{V} \times \frac{1+\epsilon}{1-\epsilon}$. If $\frac{V_1}{V}$ is at most $C \times \frac{1-\epsilon}{1+\epsilon}$, we can imply that $\frac{V_1'}{V'}$ is at most $\frac{1}{C}$, and we answer $-$. So $\frac{V_1}{V}$ is at least $\frac{1}{C} \times \frac{1-\epsilon}{1+\epsilon}$.

We also know that V_1' is at least $\frac{V_1}{1+\epsilon}$, and V is at most $V'/(1-\epsilon)$. So $\frac{V_1'}{V'}$ is at least $\frac{V_1}{V} \times \frac{1-\epsilon}{1+\epsilon}$. So the ratio $\frac{V_1'}{V'}$ is at least $\frac{1}{C} \times \frac{1-\epsilon}{1+\epsilon}^2$ which is equal to $\frac{1}{C} \times (1-\delta)$ for an arbitrary small constant δ (we can pick an arbitrary small ϵ to make δ small enough).

We can also imply that V_2'/V' is at least $\frac{1}{C} \times (1-\delta)$. So when the correct answer of the point is revealed, the volume of the core is multiplied by $1 - \frac{1}{C} \times (1-\delta)$ or a smaller number. So after l points with "I don't know" answers, the volume of the core is decreased at least by a factor of $(1 - \frac{1}{C} \times (1-\delta))^l$. So after $C/(1-\delta)$ of these points, the volume is decreased by a factor of $1/e$. We conclude that there can not be more than $O(C \times \log R)$ "I don't know" answers. This completes the proof of the following theorem.

Theorem 7 *For any $k > 0$, we can learn the linear separator using the above algorithm with k mistakes and $O(R^{1/k} \times \log R)$ "I don't know" answers, where R is equal to $\frac{(1+\gamma/\sqrt{d})^d}{(\gamma/\sqrt{d})^d}$.*

4 Conclusion

We discussed a learning framework that combines the elements of KWIK model and mistake-bound model. From one perspective, we are allowing the algorithm to make mistakes in KWIK model. We showed, using a version-space algorithm and through a reduction to egg-game puzzle, that allowing a few mistakes in KWIK model can significantly decrease the number of don't-know predictions.

From another point of view, we are letting the algorithm in mistake-bound model to say "I don't know". This can be particularly useful if don't-know predictions are cheaper than mistakes. We gave polynomial-time algorithms that effectively reduce the number of mistakes in mistake-bound model using efficient number of don't-know predictions for two concept classes monotone disjunctions and linear separators.

Acknowledgement

The authors are very grateful to Adam Kalai, Sham Kakade and Nina Balcan as well as anonymous reviewers for helpful discussions and comments.

References

[DF88] Martin E. Dyer and Alan M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.*, 17(5):967–974, 1988.

[DFK91] Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

[DLL09] C. Diuk, L. Li, and B.R. Leffler. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 249–256. ACM, 2009.

[GF] Gasarch and Fletcher. The Egg Game. www.cs.umd.edu/~gasarch/BLOGPAPERS/egg.pdf.

[KK99] M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 740–747. Citeseer, 1999.

[KS02] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2):209–232, 2002.

[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.

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

[SL08] A.L. Strehl and M.L. Littman. Online linear regression and its application to model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 20, 2008.

[WSDL] T.J. Walsh, I. Szita, C. Diuk, and M.L. Littman. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09), 2009b*.