Department of Computer Science

CSCI 599

Machine Learning Theory
Final Project Report

The “Knows What It Knows” Learning Model and Extensions

Nikolaos Flemotomos
USC I.D.: 7149389176

May 11, 2018

Abstract

In this project I present the "Knows What It Knows" learning model, in which the learning algorithm is not allowed to give wrong answers but it can abstain from giving an answer by replying "I don't know". The goal is to minimize the number of times the algorithm has to answer "I don't know". As a next step, I present a model where the learner is allowed a bounded number of mistakes, as well, and we are interested in a good trade-off between the number of mistakes and the number of "I don't know" answers. In this second model the main focus is on efficient algorithms for learning monotone disjunctions. Throughout this work, I am trying to shed light on specific aspects, analyzing in detail certain proofs that have been omitted for simplicity or brevity from the original papers, or highlighting points that could be presented in a different way.

1 Introduction

Motivated by the reinforcement learning and active learning settings, where exploration algorithms are used, the authors in [1] have proposed a learning model where the learner is only allowed to make accurate predictions, although it can abstain by replying "I don't know". The learning model is called "Knows What It Knows" (KWIK), while the "I don't know" answers are denoted by the symbol $\perp$. The model lives in the Online Learning framework and the general idea is based on the fact that there are scenarios where we would expect a wrong prediction (or a mistake in the Mistake Bound (MB) model language) to incur a greater cost than simply admitting that we are not certain about the answer. The authors present various algorithms and examples where the specific model could find use, while they also present ways of combining simple KWIK algorithms to solve more complex problems.

The KWIK model can be a powerful learning tool; however there are scenarios where the KWIK bound is exponentially bigger than the MB of algorithms trying to solve the same problem. Fot that reason, the restriction of producing only correct or $\perp$ answers is relaxed in [2], where some mistakes are also allowed. By allowing both mistakes and $\perp$'s, the problem is now transformed into a trade-off between those two types of answers, which depends on how costly a mistake may be or, equivalently, how important it is to get an actual response from the learner no matter what.

Although an optimal trade-off between mistakes and $\perp$'s is presented in [2] for learning finite hypotheses classes, the corresponding algortithm is not efficient, in the sense that it doesn't run in polynomial time, and thus, the authors present some specific alternatives for monotone disjunctions and Linear Threshold Functions (LTFs). Further improving the KWIK and mistake bounds for the case of monotone disjunctions by novel, efficient algorithms is the goal of [3].

The current work is structured as follows: In Section 2 I formally present the KWIK model in comparison to the well-known MB model. In Section 3 I focus on algorithms used to combine KWIK learners, giving details skipped in some proofs in [1] or presenting the analysis of the algorithms in a different way. In Section 4 I present the model in [2] which allows both mistakes and $\perp$'s. In Section 5 I give the algorithms used in [2] and [3] for learning monotone disjunctions, thoroughly analysing specific aspects of those. Finally, Section 6 gives a summary of the work.

2 The KWIK Model

The goal of a learning algorithm is to learn a target function $c : X \to Y$. In the Online Mistake Bound Learning model, the learning session proceeds in a sequence of trials. Throughout the session, the learner maintains an updatable hypothesis $h : X \to Y$ and, assuming for now that $Y = \{0, 1\}$, in each trial the learner

- is given an unlabeled example $x \in X$
- outputs $h(x)$
- is told the true label $c(x)$ and if $c(x) \neq h(x)$ there is a mistake
- updates its hypothesis h if there was a mistake (without loss of generality we may assume a conservative algorithm)

The total number of mistakes should be bounded.

In a similar line of work, and introducing a more general setting where Y can be equal to $\mathbb{R}$, given parameters ϵ and δ, in each trial a KWIK learner

- is given an unlabeled example $x \in X$
- outputs $h(x) \in Y \cup \{\perp\}$. If $h(x) \neq \perp$ then $|h(x) - c(x)| < \epsilon$ with probability $> 1 - \delta$
- is told the true output $c(x)$
- updates its hypothesis h if there was a $\perp$ answer

The total number of $\perp$'s should be bounded by $B(\epsilon, \delta)$. We note that in [1], the algorithm is told the true output only if it outputs $\perp$. However, the two approaches are equivalent, since otherwise, the algorithm just knows it produced an accurate answer, by the definition of the KWIK model. Additionally, the authors deal with the "noisy" Bernoulli case as well. In this scenario, $Y = [0, 1]$ and the learner is not told $c(x)$, but instead is told 1 with probability $c(x)$ and 0 with probability $1 - c(x)$.

The two basic algorithms presented for the case of finite instance space or finite hypothesis class are the memorization (Algorithm 1) and the enumeration (Algorithm 2) algorithm, respectively. The memorization algorithm has an obvious KWIK bound equal to $|X|$. By the way the enumeration algorithm is presented (which is the same as in [1]) it is assumed that Y is also finite. This is because we claim that the algorithm predicts the correct label only if $h(x) = c(x)$. I am going to elaborate more on that aspect in Section 3. The KWIK bound of this algorithm is $|H| - 1$, since initially $|\hat{L}| = |H|$, finally $|\hat{L}| = 1$ (the correct hypothesis which is identical to the target concept) and every time the algorithm outputs $\perp$, then $\hat{H}$ looses at least one element since at least one h gave a wrong prediction on x.

Algorithm 1 Memorization KWIK algorithm ($|X| < \infty$).

```
for $x$ in $X$ do
    $h(x) \leftarrow \perp$
end for
while True do
    Get some input $x \in X$; Report $h(x)$
    Get the true output $c(x)$
    if $h(x) = \perp$ then
        $h(x) \leftarrow c(x)$
    end if
end while
```

Algorithm 2 Enumeration KWIK algorithm ($|H| < \infty$).

```
$\hat{H} \leftarrow H$
while True do
    Get some input $x \in X$;  $\hat{L} = \{h(x) : h \in \hat{H}\}$
    if $|\hat{L}| = 1$ (which means all $h$'s agree on $x$) then
        Output the element of $\hat{L}$
    else if $|\hat{L}| > 1$ (which means there is at least one $h$ which predicts wrong on $x$) then
        Output $\perp$; Get the true output $c(x)$
        $\hat{H}' \leftarrow \{h : h \in \hat{H} \wedge h(x) = c(x)\}$;  $\hat{H} \leftarrow \hat{H}'$
    else if $|\hat{L}| = 0$ then
        FAIL (the problem is not realizable)
    end if
end while
```

3 Combining KWIK Learners

Here I am going to analyze some general algorithms which combine KWIK learners aiming at addressing a more complex problem. In particular, I am going to focus on the union algorithm (Algorithm 6 in [1]) and the noisy union algorithm (Algorithm 9 in [1]).

Theorem 3.1 (Union Algorithm)**.** *Let $F : X \to Y$ a set of functions. Let $\{H_i\}_{i=1}^k$ KWIK learnable classes with $H_i \subseteq F$ and KWIK bounds $\{B_i(\epsilon, \delta)\}_{i=1}^k$. Then $H = \bigcup_i H_i$ is KWIK learnable with KWIK bound $(k-1) + \sum_i B_i\left(\frac{\epsilon}{2}, \frac{\delta}{k}\right)$.*

Proof. This result is different compared to that in [1]. The reasons will become obvious from the following analysis. The union algorithm performs similarly as the enumeration algorithm. Here, we keep track of $\hat{A}$, a set of active algorithms. At each trial, the learner gets some input $x \in X$ and each algorithm in $\hat{A}$ gives an output, stored in $\hat{L}$. If $\perp \in \hat{L}$, then output $\perp$, get the true $c(x)$ and let the algorithms which predicted $\perp$ update their hypotheses. If $\perp \notin \hat{L}$ and $|\hat{L}| > 1$, [1] claims that at least one of the algorithms has predicted the wrong hypothesis. This is, however, not true in the general case, when for example $Y = \mathbb{R}$[1]. In particular, it may be the case that $|\hat{L}| > 1$ but $h_i(x)$ is accurate $\forall h_i(x) \in \hat{L}$, in the sense that $|h_i(x) - c(x)| \leq \epsilon \forall i$. However, the learner can't know $c(x)$ before it outputs something, so it cannot check this inequality. Instead, what we can do is force each individual algorithm predict with accuracy $\frac{\epsilon}{2}$ and check whether

$$\exists y \in Y : |h_i(x) - y| \leq \frac{\epsilon}{2} \;\; \forall h_i(x) \in \hat{L} \tag{1}$$

If this is true, output y, since it is guaranteed that there is at least one $h_k(x) \in \hat{L}$ (otherwise the problem is not realizable) such that

$$|y - c(x)| \leq |y - h_k(x)| + |h_k(x) - c(x)| \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$$

If this is not true, then output $\perp$, get the true $c(x)$ and update $\hat{A} \leftarrow \hat{A}'$ where

$$\hat{A}' = \{a_i : a_i \in \hat{A} \wedge |h_i(x) - c(x)| > \frac{\epsilon}{2}\} \tag{2}$$

Initially $|\hat{A}| = k$, finally there has to be at least one element in $\hat{A}$ and every time $\hat{A}$ is updated it looses at least one element since it means that condition (1) does not hold for any $y \in Y$, and thus, it does not hold for $c(x) \in Y$, which is appeared in the condition in (2). Thus, at most $k-1$ $\perp$'s may appear by this procedure. Since each individual algorithm can also output $\perp$, as already mentioned, the overal KWIK bound is $(k-1) + \sum_i B_i\left(\frac{\epsilon}{2}, \frac{\delta}{k}\right)$. We run each subalgorithm with the parameter $\frac{\delta}{k}$ instead of δ (which is yet another difference compared to [1]), because that way the overall probability of a "bad" hypothesis is bounded (by a union bound) by $k\frac{\delta}{k} = \delta$. □

The previous results are now generalized to the noisy Bernoulli case, as defined in Section 2.

Theorem 3.2 (Noisy Union Algorithm)**.** *Let $F : X \to [0,1]$ a set of functions. Let $\{H_i\}_{i=1}^k$ KWIK learnable classes with $H_i \subseteq F$ and KWIK bounds $\{B_i(\epsilon, \delta)\}_{i=1}^k$ in the noisy Bernoulli scenario. Then $H = \bigcup_i H_i$ is KWIK learnable with KWIK bound $O\left(\frac{k}{\epsilon^2} \ln \frac{k}{\delta}\right) + \sum_i B_i\left(\frac{\epsilon}{4}, \frac{\delta}{k+1}\right)$.*

[1]We note that an example (Example 2) with $Y = \mathbb{R}$ is indeed given in [1], where the less general result of Algorithm 6 is used.

Proof. Here I am going to give the proof for $k = 2$, filling in all the missing parts from the corresponding proof in [1]. Similarly as in the union algorithm, here we maintain again the sets $\hat{A}$ and $\hat{L}$ and we run the individual subalgorithms with parameters $\frac{\epsilon}{4}$ and $\frac{\delta}{3}$. If $\perp \in \hat{L}$ we behave similarly as in the union algorithm. If $\perp \notin \hat{L}$ then check whether for the input x_t

$$|h_1(x_t) - h_2(x_t)| \leq \epsilon \tag{3}$$

If yes, then output $h(x_t) = \frac{h_1(x_t)+h_2(x_t)}{2}$, since it is guaranteed that either $h_1(x_t)$ or $h_2(x_t)$ is accurate (if we assume realizability) and thus, assuming without loss of generality that this is the case for $h_1(x_t)$:

$$\begin{aligned}|h(x_t) - c(x_t)| &\leq |h(x_t) - h_1(x_t)| + |h_1(x_t) - c(x_t)| = \left|\frac{h_1(x_t)}{2} + \frac{h_2(x_t)}{2} - h_1(x_t)\right| + |h_1(x_t) - c(x_t)| \\ &= \frac{1}{2}|h_2(x_t) - h_1(x_t)| + |h_1(x_t) - c(x_t)| \leq \frac{\epsilon}{2} + \frac{\epsilon}{4} = \frac{3\epsilon}{4} < \epsilon\end{aligned}$$

If the condition (3) does not hold, then output $\perp$, get the observation z_t (where $y = P(z = 1)$) and update the current total squared prediction error l_i for each subargorithm. After sufficiently many steps, we can accurately eliminate the algorithm with the largest l_i. In particular, we define $l_i = \sum_{t \in I}(h_i(x_t) - z_t)^2$, where $I = \{t | |h_1(x_t) - h_2(x_t)| > \epsilon\}$, or equivalently $I = \{t | \perp \notin \hat{L}_t \wedge h(x_t) = \perp\}$. Let's say we finally eliminate the first algorithm because $l_1 > l_2$. We need to bound the probability of an error, that is the probability

$$P(l_1 > l_2) = P(l_1 - l_2 > 0) = P\left(\sum_{t \in I}(h_1(x_t) - z_t)^2 - (h_2(x_t) - z_t)^2 > 0\right) \triangleq P\left(\sum_{t \in I}\Psi_t > 0\right) \tag{4}$$

under the assumption that the first algorithm is accurate. We define the random variable
$\Psi_t = (h_1(x_t) - z_t)^2 - (h_2(x_t) - z_t)^2$ with mean

$$\begin{aligned}\mathbb{E}[\Psi_t] &= P(z_t = 1)[(h_1(x_t) - 1)^2 - (h_2(x_t) - 1)^2] + P(z_t = 0)[(h_1(x_t))^2 - (h_2(x_t))^2] \\ &= c(x_t)[h_1^2(x_t) + 1 - 2h_1(x_t) - h_2^2(x_t) - 1 + 2h_2(x_t)] + (1 - c(x_t))[h_1^2(x_t) - h_2^2(x_t)] \\ &= c(x_t)h_1^2(x_t) - 2c(x_t)h_1(x_t) - c(x_t)h_2^2(x_t) + 2c(x_t)h_2(x_t) + h_1^2(x_t) - h_2^2(x_t) - c(x_t)h_1^2(x_t) + c(x_t)h_2^2(x_t) \\ &= -2c(x_t)h_1(x_t) + 2c(x_t)h_2(x_t) + h_1^2(x_t) - h_2^2(x_t) \\ &= -h_1^2(x_t) - h_2^2(x_t) + 2h_1^2(x_t) - 2h_1(x_t)h_2(x_t) + 2h_1(x_t)h_2(x_t) - 2c(x_t)h_1(x_t) + 2c(x_t)h_2(x_t) \\ &= -(h_1(x_t) - h_2(x_t))^2 + 2[h_1^2(x_t) - h_1(x_t)h_2(x_t) + c(x_t)h_2(x_t) - c(x_t)h_1(x_t)] \\ &= -(h_1(x_t) - h_2(x_t))^2 + 2(h_1(x_t) - c(x_t))(h_1(x_t) - h_2(x_t)) \\ &\leq -(h_1(x_t) - h_2(x_t))^2 + 2|h_1(x_t) - c(x_t)||h_1(x_t) - h_2(x_t)| \\ &\leq -(h_1(x_t) - h_2(x_t))^2 + 2\frac{\epsilon}{4}|h_1(x_t) - h_2(x_t)| \qquad \text{(since we assumed } h_1 \text{ is } \frac{\epsilon}{4}\text{-accurate)} \\ &\overset{t \in I}{\leq} -(h_1(x_t) - h_2(x_t))^2 + \frac{|h_1(x_t) - h_2(x_t)|}{2}|h_1(x_t) - h_2(x_t)| = -\frac{(h_1(x_t) - h_2(x_t))^2}{2}\end{aligned} \tag{5}$$

Additionally,

$$|\Psi_t| = |(h_1(x_t) - z_t)^2 - (h_2(x_t) - z_t)^2| = |h_1^2(x_t) - h_2^2(x_t) - 2h_1(x_t)z_t - 2h_2(x_t)z_t|$$

$$= |(h_1(x_t) - h_2(x_t))(h_1(x_t) + h_2(x_t)) - 2z_t(h_1(x_t) - h_2(x_t))|$$
$$= |h_1(x_t) - h_2(x_t)||h_1(x_t) + h_2(x_t) - 2z_t| \leq 2|h_1(x_t) - h_2(x_t)| \tag{6}$$

At this point, we are going to use the Hoeffding's inequality, which we are stating without proof in the following Lemma:

Lemma 3.1 (Hoeffding's inequality)**.** *If $a_k \leq X_k \leq b_k$ $\forall k$ for some constants $a_k < b_k$ and if $S_n = \sum_{k=1}^n X_k$, then*

$$P(S_n - \mathbb{E}[S_n] > t) \leq \exp\left[-\frac{2t^2}{\sum_{k=1}^n (b_k - a_k)^2}\right]$$

So we have

$$P(l_1 > l_2) = P\left(\sum_{t\in I} \Psi_t > 0\right) = P\left(\sum_{t\in I} \Psi_t - \mathbb{E}\left[\sum_{t\in I} \Psi_t\right] > -\mathbb{E}\left[\sum_{t\in I} \Psi_t\right]\right)$$
$$= P\left(\sum_{t\in I} \Psi_t - \mathbb{E}\left[\sum_{t\in I} \Psi_t\right] > -\sum_{t\in I} \mathbb{E}\left[\Psi_t\right]\right) \overset{(5)}{\leq} P\left(\sum_{t\in I} \Psi_t - \mathbb{E}\left[\sum_{t\in I} \Psi_t\right] > \sum_{t\in I} \frac{(h_1(x_t) - h_2(x_t))^2}{2}\right)$$
$$\underset{\text{Hoeffding's}}{\overset{(6)}{\leq}} \exp\left[-\frac{2\left(\sum_{t\in I} \frac{(h_1(x_t)-h_2(x_t))^2}{2}\right)^2}{\sum_{t\in I} (4(h_1(x_t) - h_2(x_t)))^2}\right] = \exp\left[-\frac{\sum_{t\in I} (h_1(x_t) - h_2(x_t))^2}{32}\right]$$
$$\leq \exp\left[-\frac{\sum_{t\in I} \epsilon^2}{32}\right] = \exp\left[-\frac{|I|\epsilon^2}{32}\right]$$

We want this error to be bounded by $\frac{\delta}{3}$, so we choose $|I| = \frac{32}{\epsilon^2} \ln \frac{3}{\delta} = O(\frac{1}{\epsilon^2} \ln \frac{1}{\delta})$. So the total number of $\perp$'s is bounded by the sum of $\perp$'s due to this procedure and the $\perp$'s which are the output of the subalgorithms, giving a bound of $O(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}) + \sum_{i=1}^2 B_i\left(\frac{\epsilon}{4}, \frac{\delta}{3}\right)$. The total probability of error is bounded (by a union bound) by the error of the comparison between l_1 and l_2 and the error of the individual algorithms, so it is $\leq 3\frac{\delta}{3} = \delta$. □

4 Allowing Mistakes in the KWIK Model

Although at a high level the KWIK model may seem like a variant of the MB model, just replacing mistakes with $\perp$'s, in reality there are problems which are easy in the MB scenario, but need exponential time in the KWIK framework. A nice simple example is presented in [2]: Let the class $H : \{0,1\}^n \to \{0,1\}$ with any $h_i \in H, i = 1, 2, \cdots$ defined as

$$h_i(x) = \begin{cases} 1, & \text{if } x \text{ is the binary representation of } i \\ 0, & \text{otherwise} \end{cases}$$

Any target concept in H is MB-learnable with a bound of 1. The algorithm simply predicts 0 until it does the first mistake for a specific x, for which it now knows that the correct answer is 1 (and it is the only one by the definition of H). However, since mistakes are not allowed in KWIK, the corresponding bound is $2^n - 1$, because the algorithm has to reply $\perp$ for all the examples it has not seen until it learns that the correct label for a specific x is 1. But in the worst case this will be the last example to see and there are totally 2^n such examples with n variables. (If it reaches step $2^n - 1$, then it knows that the last one is actually labeled 1, since this is the only option left).

For that reason, the authors in [2] propose a hybrid learning model which can be viewed as the KWIK model allowing up to k mistakes, or as the MB model allowing up to $\tilde{k}$ $\perp$'s. For a specific k we are interested in minimizing the number of $\perp$'s and similarly, for a specific $\tilde{k}$ we are interested in minimizing the number of mistakes.

For a finite hypothesis class, we have seen the enumeration algorithm with a KWIK bound equal to $|H| - 1$. If we additionally allow k mistakes, we get the following result:

Theorem 4.1. *Let $F : X \to \{0,1\}$ and let $H \subseteq F$ with $|H| < \infty$. Then H is KWIK learnable with k mistakes by Algorithm 3 with a KWIK bound $(k+1)|H|^{\frac{1}{k+1}}$.*

Algorithm 3 KWIK learner with k mistakes for $|H| < \infty$.

```
$\hat{H} \leftarrow H$;  $m \leftarrow 0$;  $s \leftarrow |\hat{H}|^{\frac{k}{k+1}}$
while True do
    Get some input $x \in X$
    Calculate $S_0 = |\{h : h \in \hat{H} \wedge h(x) = 0\}|$ and $S_1 = |\{h : h \in \hat{H} \wedge h(x) = 1\}|$
    if $\min_i S_i > s$ then
        Output $\perp$
    else
        Output $\arg\max_i S_i$
    end if
    Get the true label $c(x)$
    $\hat{H}' \leftarrow \{h | h \in \hat{H} \wedge h(x) = c(x)\}$;  $\hat{H} \leftarrow \hat{H}'$
    if made a mistake then
        $m \leftarrow m + 1$;  $s \leftarrow |\hat{H}|^{\frac{k-m}{k+1-m}}$
    end if
end while
```

Proof. The proof proceeds by induction in k. As the base case, we have the number of $\perp$'s until the first mistake. Until then, at every step there is a $\perp$ and at least the minority group (from S_1 vs S_2) is removed from H. But the minority group has at least s elements and the size of H is initially $|H|$. So, the total steps with such removals cannot be more than $\frac{|H|}{s} = \frac{|H|}{|H|^{\frac{k}{k+1}}} = |H|^{\frac{1}{k+1}}$. As the inductive hypothesis we have that after the first mistake there will be at most $k-1$ mistakes and $k|\hat{H}|^{\frac{1}{k}}$ $\perp$'s. But at the first mistake, since the algorithm outputs $\arg\max_i S_i$, the majority group is removed from H, so $\hat{H}$ has now only the elements of the minority group, which are at most s: $\hat{H} \le s = |H|^{\frac{k}{k+1}}$. So, the total number of mistakes will be at most $1 + (k-1) = k$ and the total number of $\perp$'s will be at most $|H|^{\frac{1}{k+1}} + k\left(|H|^{\frac{k}{k+1}}\right)^{\frac{1}{k}} = |H|^{\frac{1}{k+1}} + k|H|^{\frac{1}{k+1}} = (k+1)|H|^{\frac{1}{k+1}}$. □

Unfortunately, Algorithm 3 has a worst-case exponential running time and large storage requirements, since it needs to evaluate and store all the members in H, which is finite, but probably exponentially large. For that reason, the authors analyze efficient algorithms for learning monotone disjunctions and LTFs. Here, I briefly mention the results and sketch the proofs for the latter. Monotone disjunctions will be the topic of the next Section. As we know, in the MB model the perceptron algorithm can learn an LTF for

δ-separable data with at most $O\left(\frac{1}{\delta^2}\right)$ mistakes. We note that for this problem H is not even finite. The main result in the new learning model is the following:

Theorem 4.2. *Let data be linearly δ-separable in $\mathbb{R}^d$. Then, a suitable LTF can be learnt with atmost k mistakes and $O\left(R^{\frac{1}{k}} \log R\right)$ $\perp$'s, where $R = \frac{\left(1+\frac{\sqrt{d}}{\delta}\right)^d}{\left(\frac{\sqrt{d}}{\delta}\right)^d} = \left(1+\frac{\sqrt{d}}{\delta}\right)^d$.*

Sketch of the Proof. The main intuition here is that we are treating the problem as a Linear Program (LP) with d variables and n linear constraints, where n is the number of examples already seen at a certain point. Of course, those constraints are not known beforehand, but are revealed gradually, every time the algorithm is told the true labels of the examples just seen. When a new unlabeled example x arrives, we can create two new LPs from the current LP, just by adding the two possible constraints; namely that $c(x) = 0$ and $c(x) = 1$. The set of feasible solutions, or core, of each one of those LPs has a smaller size, or volume, than the original one. But by examining Algorithm 3, we can see that what we need at each step for making decisions is not the actual volumes (related to the numbers S_0 and S_1 in Algorithm 3), but rather their relative volumes, with respect to some function of the volume of the core of the original LP (related to s in Algorithm 3). The trick is to uniformly sample at every step a sufficiently large number of points from the core of the current LP, without considering the newly arrived example, and make decisions based on the majority of those points. This can be done in polynomial time and it can be shown that we can make decisions about the new LPs with high probability. One final detail needed is to limit the elements of the weight vectors representing the LTFs in some specific ranges; something that does not affect the generality of the algorithm and gives us the possibility of estimating the volume of the initial LP core. A formal analysis of those intuitive steps can lead to the result stated in the Theorem. □

5 Algorithms for Learning Monotone Disjuctions with Mistakes and 'I don't know' Answers

In both [2] and [3] the authors propose algorithms to learn monotone disjunctions in the hybrid online learning model where both mistakes and $\perp$'s are allowed. Here I am going to analyze in detail one relatively simple and intuitive algorithm proposed in [2] and a more involved one proposed in [3], trying to present it in a more easily digestible format. For this analysis, it is easier to view the problem in terms of sets. The goal is to learn the set of relevant variables $R \subseteq X$ such that for any $Q \subseteq X$ the target concept is $c(Q) = 1$ iff $Q \cap R \neq \phi$. For any example x^q, the corresponding set Q is defined as $Q = \{x_i | x_i^q = 1\}$. We will maintain the sets $P^+ : P^+ \subseteq R$ (such that every $x_i \in P^+$ is relevant), $P^- : P^- \cap R \neq \phi$ (such that every $x_i \in P^-$ is irrelevant) and $P = X \subseteq (P^- \cup P^+)$ (such that every $x_i \in P^+$ is unknown).

Theorem 5.1. *Monotone disjunctions can be learnt by Algorithm 4 with at most $M \leq \frac{n}{2}$ mistakes and at most $n - 2M$ $\perp$'s.*

Proof. Every time the learner makes a mistake, P looses at least two elements and since initially $|P| = n$, there can be at most $\frac{n}{2}$ mistakes. Every time the learner outputs $\perp$, P looses exactly one element and finally $|P| = |\phi| = 0$, so for the total number of $\perp$'s, say B, it must be $B + 2M \leq n \Rightarrow B \leq n - 2M$. □

Using similar ideas, the authors propose an algorithm which makes at most $M \leq \frac{n}{3}$ mistakes and outputs at most $\frac{3n}{2} - 3M$ $\perp$'s. Vieiwing again the problem in terms of sets, an improved algorithm is presented in [3], the analysis of which is reviewed below. Before that we will need the following definition:

Algorithm 4 Learning Monotone Disjunctions with $\frac{n}{2}$ mistakes.

```
$P \leftarrow \{x_i\}_{i=1}^n$;  $P^+ \leftarrow \phi$;  $P^- \leftarrow \phi$
while $P \neq \phi$ do
    Get some input $Q$
    if $Q \cap P^+ \neq \phi$ (then we are certain the label is 1) then
        Output 1
    else if $Q \subseteq P^-$ (then we are certain the label is 0) then
        Output 0
    else(then we are not certain about the label)
        if $|Q \cap P| \geq 2$ then
            Output 1
        else(which means $|Q \cap P| = 1$)
            Output $\perp$
        end if
        Get the true label $c(Q)$
        if made a mistake then
            $P^- \leftarrow P^- \cup (Q \cap P)$;  $P \leftarrow P \setminus (Q \cap P)$
        else if output was $\perp \wedge c(Q) = 1$ then
            $P^+ \leftarrow P^+ \cup (Q \cap P)$;  $P \leftarrow P \setminus (Q \cap P)$
        else if output was $\perp \wedge c(Q) = 0$ then
            $P^- \leftarrow P^- \cup (Q \cap P)$;  $P \leftarrow P \setminus (Q \cap P)$
        end if
    end if
end while
return $P^+$
```

Definition 5.1. *Let some constant k and let the sets of sets $A_2, A_3, \cdots, A_k$ such that $M \in A_i \Rightarrow |M| = i \wedge M \cap R \neq \phi$ for the desired set of relevant variables R (which would mean that the true label of M is 1). A set S is called critical with respect to A_i, with $i > |S|$ if $|I_i| \triangleq |\{T : T \in A_i \wedge S \subseteq T\}| = l_{i-|S|+1} \triangleq l_j$ where $l_i = k^{4^i}$.*

Theorem 5.2. *Monotone disjunctions can be learnt with at most $\frac{n}{k}$ mistakes and at most $O\left(k^{4^{1+k}}n\right)$ $\perp$'s for any constant k.*

Proof. The algorithm initially proceeds in a similar way as Algorithm 4. Namely, for an input Q we first check whether $Q \cap P^+ \neq \phi$ or $Q \subseteq P^-$ and output 1 or 0 with centainty. But now we additionally maintain the sets $\{A_i\}_{i=2}^k$, as defined in Definition 5.1. So, we can additionally check whether $\exists i \in \{2, 3, \cdots, k\} \exists M \in A_i : Q \supseteq M$. If this condition holds, then output 1 with certainty. In a similar manner as in Algorithm 4, if those checks fail, we then check whether $|Q \cap P| > k$, which means Q has $> k$ unknown variables. If this condition holds, then we output 1 and if we made a mistake, then we know we have put at least $k+1$ variables in P^-.

Here comes the main difference from the previous algorithm. If Q has a critical subset S with respect to some A_i then output 1. If we made a mistake, then, using the notation in Definition 5.1, $\forall T_p \in I_i$ let $T'_p = T_p \setminus S$. We know that

$$T_p \in A_i \Rightarrow |T_p| = i \wedge T_p \cap R \neq \phi \tag{7}$$

$$T'_p \subseteq T_p \overset{(7)}{\Rightarrow} T'_p \cap R \neq \phi \tag{8}$$

$$|T'_p| = |T_p| - |S| \overset{(7)}{=} i - |S| = (i - |S| + 1) - 1 = j - 1 \tag{9}$$

So from (8) and (9), T'_p satisfies all the conditions to be added in the family A_{j-1} if $j > 2$. We know that there are $|I_i| = l_j$ such sets T'_p. We select a disjoint family of those following this procedure (PG): ◦ select one random T'_p; ◦ discard any $T'_m : T'_m \cap T'_p \neq \phi$; ◦ select one random $\hat{T}'_p$ from the remaining ones; ◦ discard any $T'_m : T'_m \cap \hat{T}'_p \neq \phi$; $\cdots$. If $j > 2$ add all those selected sets to A_{j-1}. If $j = 2$, we know that for every selected T'_p (8) $\Rightarrow c(T'_p) = 1$ and (9) $\Rightarrow |T'_p| = 1$, so this one variable in every T'_p is relevant and can thus be added to P^+.

Finally, if none of the above checks hold true, then output $\perp$ and get the true label $c(Q)$. If $c(Q) = 0$, then add all the variables of Q to P^-. If $c(Q) = 0$ then add Q to $A_{|Q|}$, which is legal, since the two conditions of elements in $A_{|Q|}$ are satisfied; namely $|Q| = |Q|$ and $c(Q) = 1$.

As it is the case for Algorithm 4, every time we add elements to onse set, we need to carefully update the others. So, in this case

- When we add a set S to A_i, we have to remove all its supersets from $A_{i+1}, A_{i+2}, \cdots, A_k$.
- When we add a variable x to P^+ or P^-, we have to remove it from P and also remove all the supersets of $\{x\}$ from $A_2, A_3, \cdots, A_k$.
- When we add a set S to A_i, we check whether some $\tilde{S} \subset S$ has become critical with respect to A_i. If yes, we have to remove all its supersets from $A_{i+1}, A_{i+2}, \cdots, A_k$.

Now, in order to prove the theorem, we first have to prove those Lemmas:

Lemma 5.1. $\forall i \in \{2, 3, \cdots, k\} \forall S |I_i| \leq l_{i-|S|+1} = l_j$ *(in case of equality S is critical wrt A_i)*

Proof. We may add an element to some A_i if (a) the learner outputs $\perp$ and $c(Q) = 1$, or (b) Q has a critical subset (we output 1) but $c(Q) = 0$. In case (a) we know that Q has no critical subset wrt A_i (otherwise the output would not be $\perp$), so $|I_i| < l_j$ and we just add one element (Q), so after the update $|I_i| \leq l_j$. In case (b) we add all the disjoint sets T'_p to A_{j-1}. What if for some T'_p $\exists S \subseteq T'_p : S$ is critical wrt A_{j-1}? But this is not possible since then we should have removed all its supersets from A_v, $v > j-1$, thus we should have removed $T_p \supset T'_p \supseteq S$ from A_i, which is a contradiction since $T_p \in A_i$. So, again after the update no property of the sets and familys is violated. □

Lemma 5.2. *The number of selected sets during the procedure (PG) is at least* $\frac{\sqrt{l_j}}{j}$.

Proof. First, we prove that from the l_j sets T'_p, there are at most $\sqrt{l_j}$ which share a common variable. This is easily done by contradiction, because $\sqrt{l_j} \geq l_{j-1}$ by defition. So, if there are $> \sqrt{l_j}$ such sets with a common variable x, that implies there are l_{j-1} sets T_p which share x and also all the variables in S, and thus share $|S| + 1$ variables. Now, letting $S' = S \cup \{x\}$ we have that $|\{T_p : T_p \in A_i \wedge S' \subseteq T_p\}| > l_{j-1}$, where $j - 1 = i - |S| + 1 - 1 = i - |S|$. But this number has to be $\leq l_{i-|S'|+1} = l_{i-|S|}$ by Lemma 5.1.

Having this result, after we select the first T'_p, we know we will discard at most $\sqrt{l_j}$ sets for each variable of T'_p, which has $j-1$ variables, so we will discard at most $(j-1)\sqrt{l_j} + 1$ sets (plus 1 since we also discard the exact T'_k which we had selected). Now, there are at least $l_j - ((j-1)\sqrt{l_j} + 1)$ sets to select from and after our selection we discard again at most $(j-1)\sqrt{l_j} + 1$ sets. We continue for r times until we have nothing left to select from:

$$l_j - r((j-1)\sqrt{l_j} + 1) = 0 \Rightarrow r = \frac{l_j}{(j-1)\sqrt{l_j} + 1} \geq \frac{l_j}{1 + j\sqrt{l_j}} \geq \frac{l_j}{h\sqrt{l_j}} = \frac{\sqrt{l_j}}{j}$$

□

Lemma 5.3. *The total number of sets inserted in* $\{A_j\}_{j=2}^{i}$ *is at most* $U_i = \frac{3i!n}{4}\prod_{j=2}^{i} l_j$.

Sketch of the Proof. The proof proceeds by induction in i. For $i = 2$, the number of sets we insert in A_2 is the sum of (a) the number of sets we remove from A_2 at some point and (b) the number of sets in A_2 at the end of the algorithm. We remove from A_2 only when we add some variable x to P^- or P^+, when we remove all the supersets of $\{x\}$. But $|\{T : T \in A_2 \wedge \{x\} \subseteq T\}| \leq l_{2-|\{x\}|+1} = l_2$. And since there are n variables, we can totally remove at most nl_2 sets. At the end of the algorithm there are at most l_2 sets in A_2 containing a variable $\tilde{x}$ and every set has 2 variables. So, there are at most $\frac{nl_2}{2}$ sets, giving a total of $nl_2 + \frac{nl_2}{2} = \frac{3nl_2}{2} = U_2$.

For $i > 2$, scenario (b) is the same as above, so it gives $\frac{nl_i}{i}$ sets. For (a), we may similarly remove up to nl_i sets because of variables added in P^+ and P^-. But now we may also remove sets because of all the other reasons listed before Lemma 5.1. It is easily shown that we can remove at most $(i-1)l_i$ sets from A_i because of any kind of insertion in A_p's, $p < i$. So, using the inductive hypothesis, we finally have at most $\frac{nl_i}{i} + nl_i + (i-1)l_iU_{i-1} + U_{i-1}$ which gives the wanted result. □

Now we can prove the bounds on the number of mistakes and $\perp$'s. We have those sources of mistakes: (a) If $|Q \cap P| > k$ and we make a mistake, we move $\geq k+1$ variables to P^-. But P^- can eventually have at most n variables. So, we can have at most $\frac{n}{k+1}$ mistakes. (b) If Q has a critical subset wrt some A_i, then at least $\frac{\sqrt{l_j}}{j}$ sets are added to A_{j-1}. But we may totally add to $A_{j-1} \leq$ sets than what we may add to $\{A_v\}_{v=2}^{j-1}$ which is $\leq U_{j-1}$. So, we can have at most $\frac{j}{\sqrt{l_j}}U_{j-1}$ mistakes (related to the specific A_i with $j > 2$) which is equal to

$$\frac{j}{\sqrt{l_j}}\frac{3n}{4}(j-1)!\prod_{p=2}^{j-1} l_p = \frac{3n}{4}j!\frac{1}{\sqrt{l_j}}\prod_{p=2}^{j-1} l_p \leq nj!\frac{k^{\sum_{p=2}^{j-1}4^p}}{k^{\frac{4^j}{2}}} = nj!\frac{k^{\frac{4^j-2}{3}}}{k^{\frac{4^j}{2}}} \leq nj!\frac{k^{\frac{4^j}{3}}}{k^{\frac{4^j}{2}}} = nj!k^{-\frac{4^j}{6}} \leq \frac{n}{k^{\frac{4^j}{6}}} < \frac{n}{k^3}$$

with the last inequality being valid because $j \geq 3 \Rightarrow 4^j \geq 4^3 \Rightarrow \frac{4^j}{6} \geq \frac{4^3}{6} > 3$. (c) If we are again in the same setting as in (b) but $j = 2$, then at least $\frac{\sqrt{l_2}}{2}$ variables are added to P^+. But P^+ can eventually have at most n variables. So, we can have at most

$$\frac{2}{\sqrt{l2}}n = \frac{2n}{k^{4^2}} = \frac{2n}{k^8} = \frac{2}{k}\frac{n}{k^7} \leq \frac{n}{k^7} < \frac{n}{k^3}$$

mistakes. Adding all the (a), (b), (c) scenarios, the number of mistakes is at most

$$\frac{n}{k+1} + \frac{n}{k^3}(\text{because of } A_2) + \frac{n}{k^3}(\text{because of } A_3) + \cdots + \frac{n}{k^3}(\text{because of } A_k) = \frac{n}{k+1} + (k-1)\frac{n}{k^3}$$

But $(k-1)k(k+1) = k(k^2-1) = k^3 - k < k^3$. So this is bounded by

$$\frac{n}{k+1} + (k-1)\frac{n}{(k-1)k(k+1)} = \frac{kn+n}{k(k+1)} = \frac{n}{k}$$

Every time the learner outputs $\perp$ we insert Q to $A_{|Q|}$ or the variables of Q to P^-. But eventually there are at most n variables in P^- and at most U_k sets added to all A_i's. So the total number of $\perp$'s is at most

$$n+U_k = n+\frac{3n}{4}k!\prod_{p=2}^{k} l_p = n+\frac{3n}{4}k!\prod_{p=2}^{k} k^{4^j} \leq n+nk!k^{\sum_{p=2}^{k}4^p} \leq n+nk^{k+\sum_{p=2}^{k}4^p} < n+n\left(k^{4^{k+1}}-1\right) = nk^{4^{k+1}}$$

since $k \geq 2$. □

It is briefly noted that the authors propose an additional algorithm with even better bounds using similar ideas with LPs and uniform sampling as explained during the analysis of Theorem 4.2. The algorithm makes at most $O\left(n\frac{\log\log n}{\log n}\right)$ mistakes and $O(n^2 \log\log n)$ $\perp$'s. This is a nearly optimal result in the sense that it is proved that any learner for disjunctions giving polynomial number of $\perp$'s has to make $\Omega\left(\frac{n}{\log n}\right)$ mistakes. The downside, however, is the running time per step, which is $\tilde{O}(n^3)$ (ignoring logarithmic terms), compared to the running time of the algorithm analyzed in Theorem 5.2 which is $O\left(k^{4^{1+k}}n\right)$.

6 Conclusion

In this work I presented the KWIK model and an extension of it allowing both mistakes and $\perp$ answers, trying to combine the positive aspects of the KWIK and MB models. Those tools can be helpful in situations where abstaining is preferable than predicting wrong. Apart from providing general definitions and algorithms, much attention was given to the specific problem of learning monotone disjunctions, presenting efficient algorithms for this important problem in Computational Learning Theory.

References

[1] Lihong Li, Michael L. Littman, and Thomas J. Walsh. "Knows what it knows: a framework for self-aware learning." Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[2] Amin Sayedi, Morteza Zadimoghaddam, and Avrim Blum. "Trading off mistakes and don't-know predictions." Advances in Neural Information Processing Systems. 2010.

[3] Erik D. Demaine, and Morteza Zadimoghaddam. "Learning Disjunctions: Near-Optimal Trade-off between Mistakes and I Don't Knows." Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2013.