Knows What It Knows: A Framework For Self-Aware Learning

Lihong Li LIHONG@CS.RUTGERS.EDU
Michael L. Littman MLITTMAN@CS.RUTGERS.EDU
Thomas J. Walsh THOMASWA@CS.RUTGERS.EDU
Department of Computer Science, Rutgers University, Piscataway, NJ 08854 USA

Abstract

We introduce a learning framework that combines elements of the well-known PAC and mistake-bound models. The KWIK (knows what it knows) framework was designed particularly for its utility in learning settings where active exploration can impact the training examples the learner is exposed to, as is true in reinforcement-learning and active-learning problems. We catalog several KWIK-learnable classes and open problems.

1. Motivation

At the core of recent reinforcement-learning algorithms that have polynomial sample complexity guarantees (Kearns & Singh, 2002; Kearns & Koller, 1999; Kakade et al., 2003; Strehl et al., 2007) lies the idea of distinguishing between instances that have been learned with sufficient accuracy and those whose outputs are still unknown.

The Rmax algorithm (Brafman & Tennenholtz, 2002), for example, estimates transition probabilities for each state–action–next-state triple of a Markov decision process (MDP). The estimates are made separately, as licensed by the Markov property, and the accuracy of the estimate is bounded using Hoeffding bounds. The algorithm explicitly distinguishes between probabilities that have been estimated accurately (known) and those for which more experience will be needed (unknown). By encouraging the agent to gather more experience in the unknown states, Rmax can guarantee a polynomial bound on the number of timesteps in which it has a non-near-optimal policy (Kakade, 2003).

In this paper, we make explicit the properties that are sufficient for a learning algorithm to be used in efficient exploration algorithms like Rmax. Roughly, the learning algorithm needs to make only accurate predictions, although it can opt out of predictions by saying "I don't know" ($\perp$). However, there must be a (polynomial) bound on the number of times the algorithm can respond $\perp$. We call such a learning algorithm KWIK ("know what it knows").

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

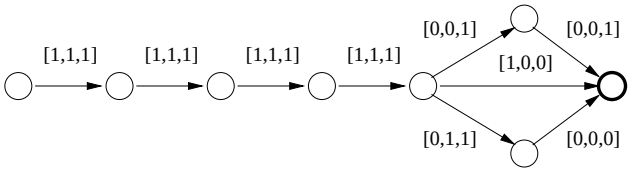

Figure 1. A cost-vector navigation graph.

Section 2 provides a motivating example and sketches possible uses for KWIK algorithms. Section 3 defines the KWIK conditions more precisely and relates them to established models from learning theory. Sections 4 and 5 survey a set of hypothesis classes for which KWIK algorithms can be created.

2. A KWIK Example

Consider the simple navigation task in Figure 1. There is a set of nodes connected by edges, with the node on the left as the source and the dark one on the right as the sink. Each edge in the graph is associated with a binary *cost vector* of dimension $d = 3$, indicated in the figure. The cost of traversing an edge is the dot product of its cost vector with a fixed weight vector $w = [1, 2, 0]$. Assume that w is not known to the agent, but the graph topology and all cost vectors are. In each episode, the agent starts from the source and moves along some path to the sink. Each time it crosses an edge, the agent observes its true cost. Once the sink is reached, the next episode begins. The learning task is to take a non-cheapest path in as few episodes as possible. There are 3 distinct paths in this example. Given the w above, the top has a cost of 12, the middle

13, and the bottom 15.

A simple approach for this task is for the agent to assume edge costs are uniform and walk the shortest (middle) path to collect data. It would gather 4 examples of $[1,1,1] \rightarrow 3$ and one of $[1,0,0] \rightarrow 1$. Standard regression algorithms could use this dataset to find a $\hat{w}$ that fits this data. Here, $\hat{w} = [1,1,1]$ is a natural choice. The learned weight vector could then be used to estimate costs for the three paths: 14 for the top, 13 for the middle, 14 for the bottom. Using these estimates, an agent would continue to take the middle path forever, never realizing it is not optimal.

In contrast, consider a learning algorithm that "knows what it knows". Instead of creating an approximate weight vector $\hat{w}$, it reasons about whether the costs for each edge can be obtained from the available data. The middle path, since we've seen all its edge costs, is definitely 13. The last edge of the bottom path has cost vector $[0,0,0]$, so its cost must be zero, but the penultimate edge of this path has cost vector $[0,1,1]$. This vector is a linear combination of the two observed cost vectors, so, regardless of w,

$$w{\cdot}[0,1,1] = w{\cdot}([1,1,1]-[1,0,0]) = w{\cdot}[1,1,1]-w{\cdot}[1,0,0],$$

which is just $3 - 1 = 2$. Thus, we know the bottom path's cost is 14—worse than the middle path.

The vector $[0,0,1]$ on the top path is linearly independent of the cost vectors we've seen, so its cost is unconstrained. We know we don't know. A safe thing to assume provisionally is that it's zero, encouraging the agent to try the top path in the second episode. Now, it observes $[0,0,1] \rightarrow 0$, allowing it to solve for w and accurately predict the cost for any vector (the training data spans $\Re^d$). It now knows that it knows all the costs, and can confidently take the optimal (top) path.

In general, any algorithm that guesses a weight vector may never find the optimal path. An algorithm that uses linear algebra to distinguish known from unknown costs will either take an optimal route *or* discover the cost of a linearly independent cost vector on each episode. Thus, it can never choose suboptimal paths more than d times.

The motivation for studying KWIK learning grew out of its use in multi-state sequential decision making problems like this one. However, other machine-learning problems could benefit from this perspective and from the development of efficient algorithms. For instance, action selection in bandit problems (Fong, 1995) and associative bandit problems (Strehl et al., 2006) (bandit problems with inputs) can both be addressed in the KWIK setting by choosing the better arm when both payoffs are known and an unknown arm otherwise.

KWIK could also be a useful framework for studying active learning (Cohn et al., 1994) and anomaly detection (Lane & Brodley, 2003), both of which are machine-learning problems that require some degree of reasoning about whether a recently presented input is predictable from previous examples. When mistakes are costly, as in utility-based data mining (Weiss & Tian, 2006) or learning robust control (Bagnell et al., 2001), having explicit predictions of certainty can be very useful for decision making.

3. Formal Definition

This section provides a formal definition of KWIK learning and its relationship to existing frameworks.

3.1. KWIK Definition

KWIK is an objective for supervised learning algorithms. In particular, we begin with an *input set* X and *output set* Y. The *hypothesis class* H consists of a set of functions from X to Y: $H \subseteq (X \rightarrow Y)$. The *target function* $h^* \in H$ is the source of training examples and is unknown to the learner. Note that the setting is "realizable", meaning we assume the target function is in the hypothesis class.

The protocol for a "run" is:

- The hypothesis class H and accuracy parameters ϵ and δ are known to both the learner and environment.
- The environment selects a target function $h^* \in H$ adversarially.
- Repeat:
  - The environment selects an input $x \in X$ adversarially and informs the learner.
  - The learner predicts an output $\hat{y} \in Y \cup \{\perp\}$.
  - If $\hat{y} \neq \perp$, it should be accurate: $|\hat{y} - y| \leq \epsilon$, where $y = h^*(x)$. Otherwise, the entire run is considered a failure. The probability of a failed run must be bounded by δ.
  - Over a run, the total number of steps on which $\hat{y} = \perp$ must be bounded by $B(\epsilon, \delta)$, ideally polynomial in $1/\epsilon$, $1/\delta$, and parameters defining H. Note that this bound should hold even if $h^* \notin H$, although, obviously, outputs need not be accurate in this case.
  - If $\hat{y} = \perp$, the learner makes an observation $z \in Z$ of the output, where $z = y$ in the deterministic case, $z = 1$ with probability y and

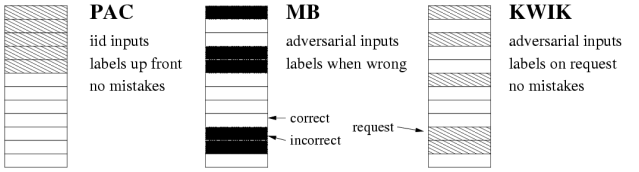

Figure 2. Relationship of KWIK to existing PAC and MB (mistake bound) frameworks in terms of how labels are provided for inputs.

0 with probability $1-y$ in the Bernoulli case, or $z = y + \eta$ for zero-mean random variable η in the additive noise case.

3.2. Connection to PAC and MB

Figure 2 illustrates the relationship of KWIK to the similar PAC (probably approximately correct) (Valiant, 1984) and MB (mistake bound) (Littlestone, 1987) frameworks. In all three cases, a series of inputs (instances) is presented to the learner. Each input is depicted in the figure by a rectangular box.

In the PAC model, the learner is provided with labels (correct outputs) for an initial sequence of inputs, depicted by cross-hatched boxes. After that point, the learner is responsible for producing accurate outputs (empty boxes) for all new inputs. Inputs are drawn from a fixed distribution.

In the MB model, the learner is expected to produce an output for every input. Labels are provided to the learner whenever it makes a mistake (filled boxes). Inputs are selected adversarially, so there is no bound on when the last mistake might be made. However, MB algorithms guarantee that the *total* number of mistakes is small, so the ratio of incorrect to correct outputs must go to zero asymptotically. Any MB algorithm for a hypothesis class can be used to provide a PAC algorithm for the same class, but not necessarily vice versa (Blum, 1994).

The KWIK model has elements of both PAC and MB. Like PAC, a KWIK algorithm is not allowed to make mistakes. Like MB, inputs to a KWIK algorithm are selected adversarially. Instead of bounding mistakes, a KWIK algorithm must have a bound on the number of label requests ($\perp$) it can make. By requiring performance to be independent of the distribution, a KWIK algorithm can be used in cases in which the input distribution is dependent in complex ways on the KWIK algorithm's behavior, as can happen in on-line or active learning settings. And, like PAC and MB, the definition of KWIK algorithms can be naturally extended to enforce low computational complexity.

Note that any KWIK algorithm can be turned into a MB algorithm with the same bound by simply having the algorithm guess an output each time it is not certain. However, some hypothesis classes are exponentially harder to learn in the KWIK setting than in the MB setting. An example is conjunctions of n Boolean variables, in which MB algorithms can guess "false" when uncertain and learn with $n+1$ mistakes, but a KWIK algorithm may need $\Omega(2^{n/2})$ $\perp$s to acquire the negative examples required to capture the target hypothesis.

3.3. Other Online Learning Models

The notion of allowing the learner to opt out of some inputs by returning $\perp$ is not unique to KWIK. Several other authors have considered related models. For instance, sleeping experts (Freund et al., 1997) can respond $\perp$ for some inputs, although they need not learn from these experiences. Learners in the settings of Selective Sampling (SS) (Cesa-Bianchi et al., 2006) and Label Efficient Prediction (Cesa-Bianchi et al., 2005) request labels randomly with a changing probability and achieve bounds on the expected number of mistakes and the expected number of label requests for a finite number of interactions. These algorithms cannot be used unmodified in the KWIK setting because, with high probability, KWIK algorithms must not make mistakes at any time. In the MB-like Apple-Tasting setting (Helmbold et al., 2000), the learner receives feedback asymmetrically only when it predicts a particular label (a positive example, say), which conflates the request for a sample with the prediction of a particular outcome.

Open Problem 1 *Is there a way of modifying SS algorithms to satisfy the KWIK criteria?*

4. Some KWIK Learnable Classes

This section describes some hypothesis classes for which KWIK algorithms are available. It is not meant to be an exhaustive survey, but simply to provide a flavor for the properties of hypothesis classes KWIK algorithms can exploit. The complexity of many learning problems has been characterized by defining the *dimensionality* of hypothesis classes (Angluin, 2004). No such definition has been found for the KWIK model, so we resort to enumerating examples of learnable classes.

Open Problem 2 *Is there a way of characterizing the "dimension" of a hypothesis class in a way that can be used to derive KWIK bounds?*

4.1. Memorization and Enumeration

We begin by describing the simplest and most general KWIK algorithms.

Algorithm 1 *The* memorization algorithm *can learn any hypothesis class with input space X with a KWIK bound of $|X|$. This algorithm can be used when the input space X is finite and observations are noise free.*

To achieve this bound, the algorithm simply keeps a mapping $\hat{h}$ initialized to $\hat{h}(x) = \perp$ for all $x \in X$. When the environment chooses an input x, the algorithm reports $\hat{h}(x)$. If $\hat{h}(x) = \perp$, the environment will provide a label y and the algorithm will assign $\hat{h}(x) := y$. It will only report $\perp$ once for each input, so the KWIK bound is $|X|$.

Algorithm 2 *The* enumeration algorithm *can learn any hypothesis class H with a KWIK bound of $|H|-1$. This algorithm can be used when the hypothesis class H is finite and observations are noise free.*

The algorithm keeps track of $\hat{H}$, the version space, and initially $\hat{H} = H$. Each time the environment provides input $x \in X$, the algorithm computes $\hat{L} = \{h(x) | h \in \hat{H}\}$. That is, it builds the set of all outputs for x for all hypotheses that have not yet been ruled out. If $|\hat{L}| = 0$, the version space has been exhausted and the target hypothesis is not in the hypothesis class ($h^* \notin H$).

If $|\hat{L}| = 1$, it means that all hypotheses left in $\hat{H}$ agree on the output for this input, and therefore the algorithm knows what the proper output must be. It returns $\hat{y} \in \hat{L}$. On the other hand, if $|\hat{L}| > 1$, two hypotheses in the version space disagree. In this case, the algorithm returns $\perp$ and receives the true label y. It then computes an updated version space

$$\hat{H}' = \{h | h \in \hat{H} \wedge h(x) = y\}.$$

Because $|\hat{L}| > 1$, there must be some $h \in \hat{H}$ such that $h(x) \neq y$. Therefore, the new version space must be smaller $|\hat{H}'| \leq |\hat{H}| - 1$. Before the next input is received, the version space is updated $\hat{H} := \hat{H}'$.

If $|\hat{H}| = 1$ at any point, $|\hat{L}| = 1$, and the algorithm will no longer return $\perp$. Therefore, $|H|-1$ is the maximum number of $\perp$s the algorithm can return.

Example 1 *You own a bar that is frequented by a group of n patrons P. There is one patron $\mathbf{f} \in P$ who is an instigator—whenever a group of patrons is in the bar $G \subseteq P$, if $\mathbf{f} \in G$, a fight will break out. However, there is another patron $\mathbf{p} \in P$, who is a peacemaker. If $\mathbf{p}$ is in the group, it will prevent a fight, even if $\mathbf{f}$ is present.*

You want to predict whether a fight will break out among a subset of patrons, initially without knowing the identities of $\mathbf{f}$ and $\mathbf{p}$. The input space is $X = 2^P$ and the output space is $Y = \{$fight, no fight$\}$.

The memorization algorithm achieves a KWIK bound of 2^n for this problem, since it may have to see each possible subset of patrons. However, the enumeration algorithm can KWIK learn this hypothesis class with a bound of $n(n-1)$ since there is one hypothesis for each possible assignment of a patron to $\mathbf{f}$ and $\mathbf{p}$. Each time it reports $\perp$, it is able to rule out at least one possible instigator–peacemaker combination.

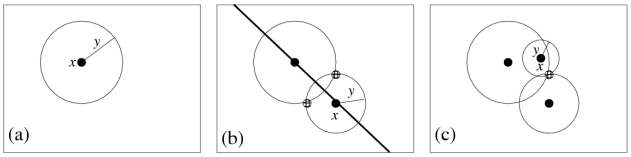

Figure 3. Schematic of behavior of the planar-distance algorithm after the first (a), second (b), and third (c) time it returns $\perp$.

4.2. Real-valued Functions

The previous two examples exploited the finiteness of the hypothesis class and input space. KWIK bounds can also be achieved when these sets are infinite.

Algorithm 3 *Define $X = \Re^2$, $Y = \Re$, and*

$$H = \{f | f : X \to Y, c \in \Re^2, f(x) = \|x - c\|_2\}.$$

This is, there is an unknown point and the target function maps input points to the distance from the unknown point. The planar-distance algorithm *can learn in this hypothesis class with a KWIK bound of 3.*

The algorithm proceeds as follows, illustrated in Figure 3. First, given initial input x, the algorithm says $\perp$ and receives output y. Since y is the distance between x and some unknown point c, we know c must lie on the circle illustrated in Figure 3(a). (If $y = 0$, then $c = x$.) Let's call this input–output pair x_1, y_1. The algorithm will return y_1 for any future input that matches x_1. Otherwise, it will need to return $\perp$ and will obtain a new input–output pair x, y, as shown in Figure 3(b). They become x_2 and y_2.

Now, the algorithm can narrow down the location of c to the two hatch-marked points. In spite of this ambiguity, for any input on the dark diagonal line the algorithm will be able to return the correct distance—all

these points are equidistant from the two possibilities. The two circles must intersect, assuming the target hypothesis is in H[1].

Once an input x is received that is not co-linear with x_1 and x_2, the algorithm returns $\perp$ and obtains another x, y pair, as illustrated in Figure 3(c). Finally, since three circles will intersect at at most one point, the algorithm can identify the location of c and use it to correctly answer any future query. Thus, three $\perp$s suffice for KWIK learning in this setting. The algorithm generalizes to d-dimensional versions of the setting with a KWIK bound of $d + 1$.

Algorithm 3 illustrates a number of important points. First, since learners have no control over their inputs in the KWIK setting, they must be robust to degenerate inputs such as inputs that lie precisely on a line. Second, they can often return valid answers for some inputs even before they have learned the target function over the entire input space.

4.3. Noisy Observations

Up to this point, observations have been noise free. Next, we consider the simplest noisy KWIK learning problem in the Bernoulli case.

Algorithm 4 *The* coin-learning algorithm *can accurately predict the probability that a biased coin will come up heads given Bernoulli observations with a KWIK bound of* $B(\epsilon, \delta) = \frac{1}{2\epsilon^2} \ln \frac{2}{\delta} = O\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right)$.

We have a biased coin whose unknown probability of heads is p. In the notation of this paper, $|X| = 1$, $Y = [0, 1]$, and $Z = \{0, 1\}$. We want to learn an estimate $\hat{p}$ that is accurate ($|\hat{p} - p| \leq \epsilon$) with high probability $(1 - \delta)$.

If we could observe p, then this problem would be trivial: Say $\perp$ once, observe p, and let $\hat{p} = p$. The KWIK bound is thus 1. Now, however, observations are noisy. Instead of observing p, we see either 1 (with probability p) or 0 (with probability $1 - p$).

Each time the algorithm says $\perp$, it gets an independent trial that it can use to compute $\hat{p} = \frac{1}{T} \sum_{t=1}^{T} z_t$, where $z_t \in Z$ is the tth observation in T trials. The number of trials needed before we are $1-\delta$ certain our estimate is within ϵ can be computed using a Hoeffding bound:

$$T \geq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta} = O\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right).$$

[1]They can also intersect at one point, if the circles are tangent, in which case the algorithm can identify c unambiguously.

Algorithm 5 *Define* $X = \Re^d$, $Y = \Re$, *and*

$$H = \{f | f : X \to Y, w \in \Re^d, f(x) = w \cdot x\}.$$

That is, H is the linear functions on d variables. Given additive noise, the noisy linear-regression algorithm *can learn in H with a KWIK bound of* $B(\epsilon, \delta) = \tilde{O}(d^3/\epsilon^4)$, *where* $\tilde{O}(\cdot)$ *suppresses log factors.*

The deterministic case was described in Section 2 with a bound of d. Here, the algorithm must be cautious to average over the noisy samples to make predictions accurately. This problem was solved by Strehl and Littman (2008). The algorithm uses the least squares estimate of the weight vector for inputs with high certainty. Certainty is measured by two terms representing (1) the number and proximity of previous samples to the current point and (2) the appropriateness of the previous samples for making a least squares estimate. When certainty is low for either measure, the algorithm reports $\perp$ and observes a noisy sample of the linear function.

Here, solving a noisy version of a problem resulted in an increased KWIK bound (from d to essentially d^3). Note that the deterministic Algorithm 3 also has a bound of d, but no bound has been found for the stochastic case.

Open Problem 3 *Is there a general scheme for taking a KWIK algorithm for a deterministic class and updating it to work in the presence of noise?*

5. Combining KWIK Learners

This section provides examples of how KWIK learners can be combined to provide learning guarantees for more complex hypothesis classes.

Algorithm 6 *Let* $F : X \to Y$ *be the set of functions mapping input set X to output set Y. Let* $H_1, \ldots, H_k$ *be a set of KWIK learnable hypothesis classes with bounds of* $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ *where* $H_i \subseteq F$ *for all* $1 \leq i \leq k$. *That is, all the hypothesis classes share the same input/output sets. The* union algorithm *can learn the joint hypothesis class* $H = \bigcup_i H_i$ *with a KWIK bound of* $B(\epsilon, \delta) = (1 - k) + \sum_i B_i(\epsilon, \delta)$.

The union algorithm is like a higher-level version of the enumeration algorithm (Algorithm 2) and applies in the deterministic setting. It maintains a set of active algorithms $\hat{A}$, one for each hypothesis class: $\hat{A} = \{1, \ldots, k\}$. Given an input x, the union algorithm queries each algorithm $i \in \hat{A}$ to obtain a prediction $\hat{y}_i$ from each active algorithm. Let $\hat{L} = \{\hat{y}_i | i \in \hat{A}\}$.

If $\perp \in \hat{L}$, the union algorithm reports $\perp$ and obtains the correct output y. Any algorithm i for which $\hat{y} = \perp$ is then sent the correct output y to allow it to learn. If $|\hat{L}| > 1$, then there is disagreement among the subalgorithms. The union algorithm reports $\perp$ in this case because at least one of the algorithms has learned the wrong hypothesis and it needs to know which.

Any algorithm that made a prediction other than y or $\perp$ is "killed"—removed from consideration. That is,

$$\hat{A}' = \{i | i \in \hat{A} \wedge (\hat{y}_i = \perp \vee \hat{y}_i = y)\}.$$

On each input for which the union algorithm reports $\perp$, either one of the subalgorithms reported $\perp$ (at most $\sum_i B_i(\epsilon, \delta)$ times) or two algorithms disagreed and at least one was removed from $\hat{A}$ (at most $k - 1$ times). The KWIK bound follows from these facts.

Example 2 *Let $X = Y = \Re$. Now, define $H_1 = \{f | f(x) = |x - c|, c \in \Re\}$. That is, each function in H_1 maps x to its distance from some unknown point c. We can learn H_1 with a KWIK bound of 2 using a 1-dimensional version of Algorithm 3. Next, define $H_2 = \{f | f(x) = yx + b, m \in \Re, b \in \Re\}$. That is, H_2 is the set of lines. We can learn H_2 with a KWIK bound of 2 using the regression algorithm in Section 2. Finally, define $H = H_1 \cup H_2$, the union of these two classes. We can use Algorithm 6 to KWIK learn H.*

Assume the first input is $x_1 = 2$. The union algorithm asks the learners for H_1 and H_2 the output for x_1 and neither has any idea, so it returns $\perp$ and receives the feedback $y_1 = 2$, which it passes to the subalgorithms. The next input is $x_2 = 8$. The learners for H_1 and H_2 still don't have enough information, so it returns $\perp$ and sees $y_2 = 4$, which it passes to the subalgorithms. Next, $x_3 = 1$. Now, the learner for H_1 unambiguously computes $c = 4$, because that's the only interpretation consistent with the first two examples ($|2 - 4| = 2$, $|8 - 4| = 4$), so it returns $|1 - 4| = 3$. On the other hand, the learner for H_2 unambiguously computes $m = 1/3$ and $b = 4/3$, because that's the only interpretation consistent with the first two examples ($2 \times 1/3 + 4/3 = 2$, $8 \times 1/3 + 4/3 = 4$), so it returns $1 \times 1/3 + 4/3 = 5/3$. Since the two subalgorithms disagree, the union algorithm returns $\perp$ one last time and finds out that $y_3 = 3$. It makes all future predictions (accurately) using the algorithm for H_1.

Next, we consider a variant of Algorithm 1 that combines learners across disjoint input spaces.

Algorithm 7 *Let $X_1, \ldots, X_k$ be a set of disjoint input spaces ($X_i \cap X_j = \emptyset$ if $i \neq j$) and Y be an output space. Let $H_1, \ldots, H_k$ be a set of KWIK learnable hypothesis classes with bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ where $H_i \in (X_i \to Y)$. The* input-partition algorithm *can learn the hypothesis class $H \in (X_1 \cup \cdots \cup X_k \to Y)$ with a KWIK bound of $B(\epsilon, \delta) = \sum_i B_i(\epsilon, \delta/k)$.*

The input-partition algorithm runs the learning algorithm for each subclass H_i. When it receives an input $x \in X_i$, it queries the learning algorithm for class H_i and returns its response, which is ϵ accurate, by request. To achieve $1 - \delta$ certainty, it insists on $1 - \delta/k$ certainty from each of the subalgorithms. By the union bound, the overall failure probability must be less than the sum of the failure probabilities for the subalgorithms.

Example 3 *An MDP consists of n states and m actions. For each combination of state and action and next state, the transition function returns a probability. As the reinforcement-learning agent moves around in the state space, it observes state–action–state transitions and must predict the probabilities for transitions it has not yet observed. In the model-based setting, an algorithm learns a mapping from the size n^2m input space of state–action–state combinations to probabilities via Bernoulli observations. Thus, the problem can be solved via the input-partition algorithm (Algorithm 7) over a set of individual probabilities learned via Algorithm 4. The resulting KWIK bound is $B(\epsilon, \delta) = O\left(\frac{n^2 m}{\epsilon^2} \ln \frac{nm}{\delta}\right)$.*

Note that this approach is precisely what is found in most efficient RL algorithms in the literature (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002).

Algorithm 7 combines hypotheses by partitioning the input space. In contrast, the next example concerns combinations in input and output space.

Algorithm 8 *Let $X_1, \ldots, X_k$ and $Y_1, \ldots, Y_k$ be a set of input and output spaces and $H_1, \ldots, H_k$ be a set of KWIK learnable hypothesis classes with bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ on these spaces. That is, $H_i \in (X_i \to Y_i)$. The* cross-product algorithm *can learn the hypothesis class $H \in ((X_1 \times \cdots \times X_k) \to (Y_1 \times \cdots \times Y_k))$ with a KWIK bound of $B(\epsilon, \delta) = \sum_i B_i(\epsilon, \delta/k)$.*

Here, each input consists of a vector of inputs from each of the spaces $X_1, \ldots, X_k$ and outputs are vectors of outputs from $Y_1, \ldots, Y_k$. Like Algorithm 7, each component of this vector can be learned independently via the corresponding algorithm. Each is learned to within an accuracy of ϵ and confidence $1 - \delta/k$. Any time any component returns $\perp$, the cross-product algorithm returns $\perp$. Since each $\perp$ returned can be traced to one of the subalgorithms, the total is bounded as

described above. By the union bound, total failure probability is no more than $k \times \delta/k = \delta$.

Example 4 *Transitions in factored-state MDP can be thought of as mappings from vectors to vectors. Given known dependencies, the cross-product algorithm can be used to learn each component of the transition function. Each component is, itself, an instance of Algorithm 7 applied to the coin-learning algorithm. This three-level KWIK algorithm provides an approach to learn the transition function of a factored-state MDP with a polynomial KWIK bound. This insight can be used to derive the factored-state-MDP learning algorithm used by Kearns and Koller (1999).*

The previous two algorithms apply to both deterministic and noisy observations. We next provide a powerful algorithm that generalizes the union algorithm (Algorithm 6) to work with noisy observations as well.

Algorithm 9 *Let $F : X \to Y$ be the set of functions mapping input set X to output set $Y = [0, 1]$. Let $Z = \{0, 1\}$ be a binary observation set. Let $H_1, \ldots, H_k$ be a set of KWIK learnable hypothesis classes with bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ where $H_i \subseteq F$ for all $1 \leq i \leq k$. That is, all the hypothesis classes share the same input/output sets. The* noisy union algorithm *can learn the joint hypothesis class $H = \bigcup_i H_i$ with a KWIK bound of $B(\epsilon, \delta) = O\left(\frac{k}{\epsilon^2} \ln \frac{k}{\delta}\right) + \sum_{i=1}^{k} B_i(\frac{\epsilon}{4}, \frac{\delta}{k+1})$.*

For simplicity, we sketch the special case of $k = 2$. The general case will be briefly discussed at the end. The noisy union algorithm is similar to the union algorithm (Algorithm 6), except that it has to deal with noisy observations. The algorithm proceeds by running the KWIK algorithms, using parameters (ϵ_0, δ_0), as subalgorithms for each of the H_i hypothesis classes, where $\epsilon_0 = \frac{\epsilon}{4}$ and $\delta_0 = \frac{\delta}{3}$. Given an input x_t in trial t, it queries each algorithm i to obtain a prediction $\hat{y}_{ti}$. Let $\hat{L}_t$ be the set of responses.

If $\perp \in \hat{L}_t$, the noisy union algorithm reports $\perp$, obtains an observation $z_t \in Z$, and sends it to all subalgorithms i with $\hat{y}_{ti} = \perp$ to allow them to learn. In the following, we focus on the other case where $\perp \notin \hat{L}_t$.

If $|\hat{y}_{t1} - \hat{y}_{t2}| \leq 4\epsilon_0$, then these two predictions are sufficiently consistent, and we claim that, with high probability, the prediction $\hat{p}_t = (\hat{y}_{t1} + \hat{y}_{t2})/2$ is ϵ-close to $y_t = \Pr(z_t = 1)$. This claim follows because, by assumption, one of the predictions, say $\hat{y}_{t1}$, deviates from y_t by at most ϵ_0 with probability at least $1 - \delta/3$, and hence $|\hat{p}_t - y_t| = |\hat{p}_t - \hat{y}_{t1} + \hat{y}_{t1} - y_t| \leq |\hat{p}_t - \hat{y}_{t1}| + |\hat{y}_{t1} - \hat{y}_t| = |\hat{y}_{t1} - \hat{y}_{t2}|/2 + |\hat{y}_{t1} - \hat{y}_t| \leq 2\epsilon_0 + \epsilon_0 < \epsilon$.

If $|\hat{y}_{t1} - \hat{y}_{t2}| > 4\epsilon_0$, then the individual predictions are not consistent enough for the noisy union algorithm to make an ϵ-accurate prediction. Thus, the noisy union algorithm reports $\perp$ and needs to know which subalgorithm provided an inaccurate response. But, since the observations are noisy in this problem, it cannot eliminate h_i on the basis of a single observation. Instead, it maintains the total squared prediction error for every subalgorithm i: $\ell_i = \sum_{t \in I} (\hat{y}_{ti} - z_t)^2$, where $I = \{t \,|\, |\hat{y}_{t1} - \hat{y}_{t2}| > 4\epsilon_0\}$ is the set of trials in which the subalgorithms gave inconsistent predictions. We observe that $|I|$ is the number of $\perp$s returned by the noisy union algorithm alone (not counting those returned by the subalgorithms). Our last step is to show ℓ_i provides a robust measure for eliminating invalid predictors when $|I|$ is sufficiently large.

Applying the Hoeffding bound and some algebra, we find $\Pr(\ell_1 > \ell_2) \leq$

$$\exp\left(-\frac{\sum_{t \in I} |\hat{y}_{t1} - \hat{y}_{t2}|^2}{8}\right) \leq \exp\left(-2\epsilon_0^2 |I|\right).$$

Setting the righthand side to be $\delta/3$ and solving for $|I|$, we have $|I| = \frac{1}{2\epsilon_0^2} \ln \frac{3}{\delta} = O\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right)$.

Since each h_i succeeds with probability $1 - \frac{\delta}{3}$, and the comparison of ℓ_1 and ℓ_2 also succeeds with probability $1 - \frac{\delta}{3}$, a union bound implies that the noisy union algorithm succeeds with probability at least $1 - \delta$. All $\perp$s are either from a subalgorithm (at most $\sum_i B_i(\epsilon_0, \delta_0)$) or from the noisy union algorithm ($O\left(\frac{1}{\epsilon^2} \ln \frac{1}{\delta}\right)$).

The general case where $k > 2$ can be reduced to the $k = 2$ case by pairing the k learners and running the noisy union algorithm described above on each pair. Here, each subalgorithm is run with parameter $\frac{\epsilon}{4}$ and $\frac{\delta}{k+1}$. Although there are $\binom{k}{2} = O(k^2)$ pairs, a slightly improved reduction and analysis can reduce the dependence of $|I|$ on k from quadratic to linearithmic, leading to the bound given in the statement.

Example 5 *Without known dependencies, learning a factored-state MDP is more challenging. Strehl et al. (2007) showed that each possible dependence structure can be viewed as a separate hypothesis and provided an algorithm for learning the dependencies in a factored-state MDP while learning the transition probabilities. The algorithm can be viewed as a four-level KWIK algorithm with a noisy union algorithm at the top (to discover the dependence structure), a cross-product algorithm beneath it (to decompose the transitions for the separate components of the factored-state representation), an input-partition algorithm below that (to handle the different combinations of state component and action), and a coin-learning algorithm*

at the very bottom (to learn the transition probabilities themselves). Note that Algorithm 9 is conceptually simpler, significantly more efficient ($k \log k$ vs. k^2 dependence on k), and more generally applicable than the one due to Strehl et al. (2007).

6. Conclusion and Future Work

We described the KWIK ("knows what it knows") model of supervised learning, which identifies and generalizes a key step common to a class of algorithms for efficient exploration. We provided algorithms for a set of basic hypothesis classes given deterministic and noisy observations as well as methods for composing hypothesis classes to create more complex algorithms. One example algorithm consisted of a four-level decomposition of an existing learning algorithm from the reinforcement-learning literature.

By providing a set of example algorithms and composition rules, we hope to encourage the use of KWIK algorithms as a component in machine-learning applications as well as spur the development of novel algorithms. One concern of particular interest in applying the KWIK framework to real-life data we leave as an open problem.

Open Problem 4 *How can KWIK be adapted to apply in the unrealizable setting in which the target hypothesis can be chosen from outside the hypothesis class H?*

References

Angluin, D. (2004). Queries revisited. *Theoretical Computer Science*, *313*, :175–194.

Bagnell, J., Ng, A. Y., & Schneider, J. (2001). *Solving uncertain Markov decision problems* (Technical Report CMU-RI-TR-01-25). Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Blum, A. (1994). Separating distribution-free and mistake-bound learning models over the Boolean domain. *SIAM Journal on Computing*, *23*, 990–1000.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*, 213–231.

Cesa-Bianchi, N., Gentile, C., & Zaniboni, L. (2006). Worst-case analysis of selective sampling for linear classification. *Journal of Machine Learning Research*, *7*, 1205–1230.

Cesa-Bianchi, N., Lugosi, G., & Stoltz, G. (2005). Minimizing regret with label efficient prediction. *IEEE Transactions on Information Theory*, *51*, 2152–2162.

Cohn, D. A., Atlas, L., & Ladner, R. E. (1994). Improving generalization with active learning. *Machine Learning*, *15*, 201–221.

Fong, P. W. L. (1995). A quantitative study of hypothesis selection. *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)* (pp. 226–234).

Freund, Y., Schapire, R. E., Singer, Y., & Warmuth, M. K. (1997). Using and combining predictors that specialize. *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (pp. 334–343).

Helmbold, D. P., Littlestone, N., & Long, P. M. (2000). Apple tasting. *Information and Computation*, *161*, 85–139.

Kakade, S., Kearns, M., & Langford, J. (2003). Exploration in metric state spaces. *Proceedings of the 20th International Conference on Machine Learning*.

Kakade, S. M. (2003). *On the sample complexity of reinforcement learning*. Doctoral dissertation, Gatsby Computational Neuroscience Unit, University College London.

Kearns, M. J., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 740–747).

Kearns, M. J., & Singh, S. P. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, *49*, 209–232.

Lane, T., & Brodley, C. E. (2003). An empirical study of two approaches to sequence learning for anomaly detection. *Machine Learning*, *51*, 73–107.

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*, 285–318.

Strehl, A. L., Diuk, C., & Littman, M. L. (2007). Efficient structure learning in factored-state MDPs. *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*.

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

Strehl, A. L., Mesterharm, C., Littman, M. L., & Hirsh, H. (2006). Experience-efficient learning in associative bandit problems. *Proceedings of the Twenty-third International Conference on Machine Learning (ICML-06)*.

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

Weiss, G. M., & Tian, Y. (2006). Maximizing classifier utility when training data is costly. *SIGKDD Explorations*, *8*, 31–38.

Acknowledgments

Support provided by NSF IIS-0325281 and DARPA TL.