Know What it Knows : A Framework for Self-Aware Learning

Lihong Li, Michael L. Littman, Thomas J. Walsh

Rutgers University

ICML-08. *25th International Conference on Machine Learning*, Helsinki, Finland, July 2008

Presented by – Praveen Venkateswaran, 2/26/18

Slides and images based on the paper

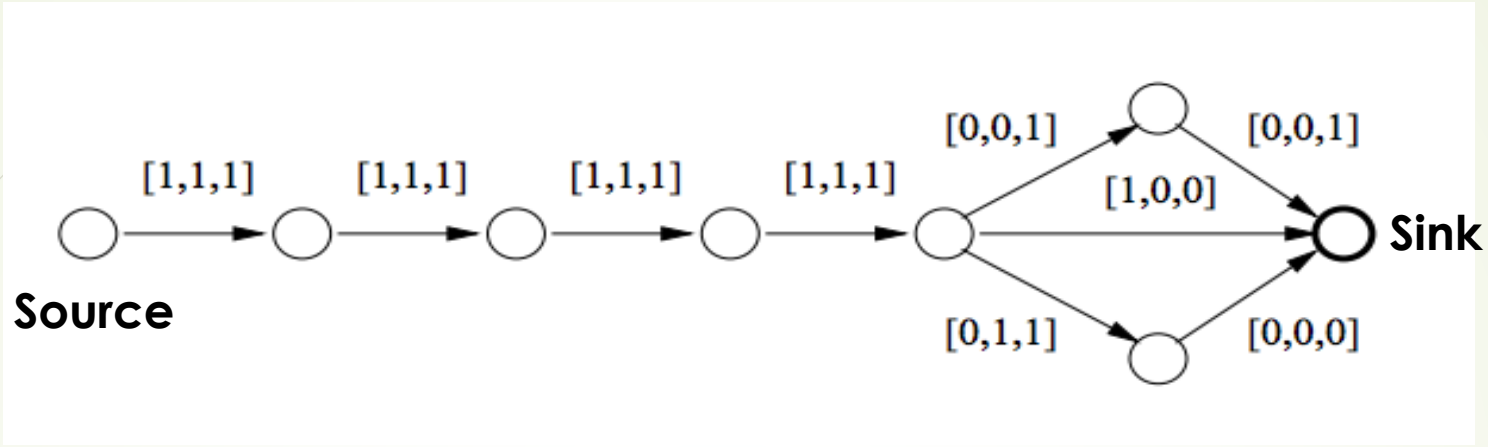
KWIK Framework

- The paper introduces a new framework called “Know What It Knows” (KWIK)
- Utility in settings where active exploration can impact the training examples that the learner is exposed to (e.g.) Reinforcement learning, Active learning
- Core of many reinforcement learning algorithms have the idea of distinguishing between instances that have been learned with sufficient accuracy vs. those whose outputs are unknown.

KWIK Framework

- The paper makes explicit some properties that are sufficient for a learning algorithm to be used in efficient exploration algorithms.
- Describes set of hypothesis classes for which KWIK algorithms can be created.
- 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 $\perp$. The authors call such a learning algorithm KWIK.

Motivation : Example

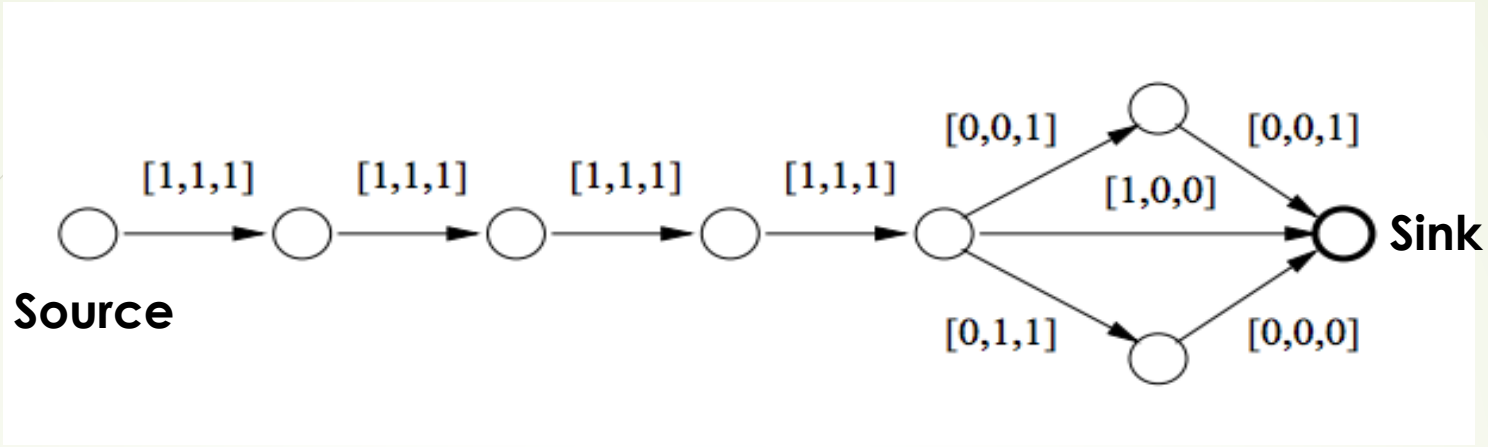


- Each edge associated with binary cost vector – dimension **d = 3**
- Cost of traversing edge = cost vector ● fixed weight vector
- Fixed weight vector given to be $\boldsymbol{w}$ **= [1,2,0]**
- Agent does not know $\boldsymbol{w}$, but knows topology and all cost vectors
- In each episode, agent starts from source and moves along a path to sink
- Every time an edge is crossed, agent observes its true cost

Learning Task – Take non-cheapest path in as few episodes as possible!

- Given w, we see ground truth cost of top path = 12, middle path = 13, and bottom path = 15

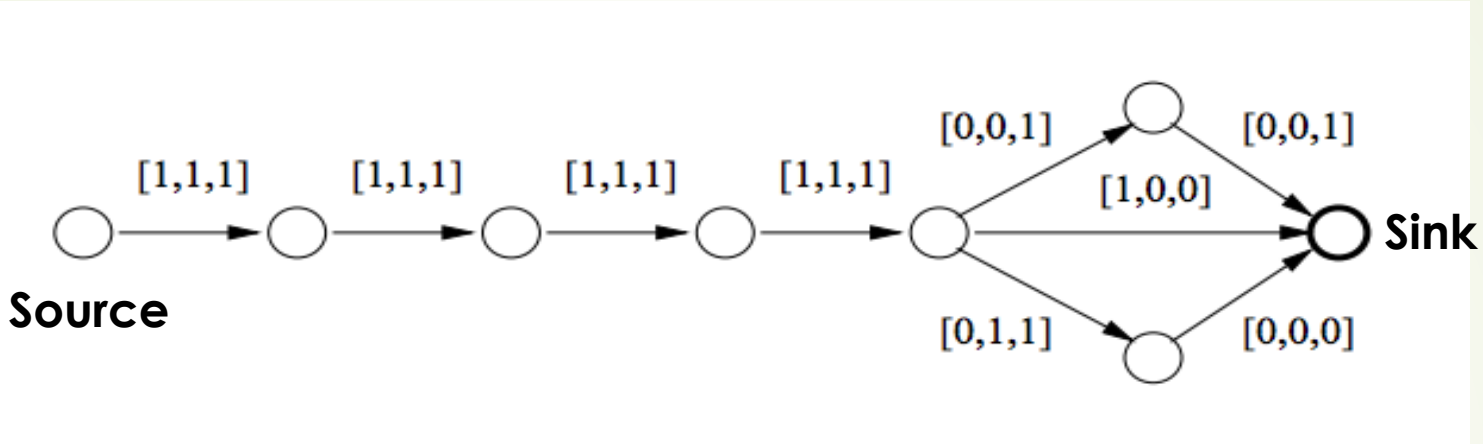
Motivation : Simple Approach



- Simple approach – Agent assumes edge costs are uniform and walks the shortest path (middle)
- Since agent observes true cost upon traversal, it gathers 4 instances of **[1,1,1] → 3** and 1 instance of **[1,0,0] → 1**
- Using standard regression, the learned weight vector $\hat{w}$ **= [1,1,1]**
- Using $\hat{w}$, cost of top = 14, middle = 13, bottom = 14

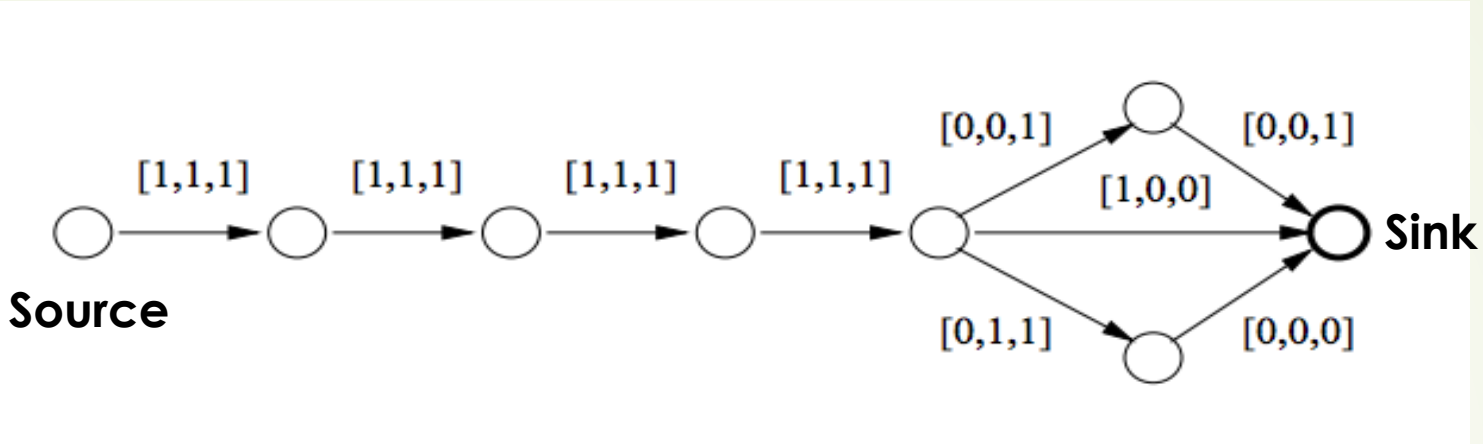
Using these estimates, an agent will take the middle path forever without realizing its not optimal

Motivation : KWIK Approach



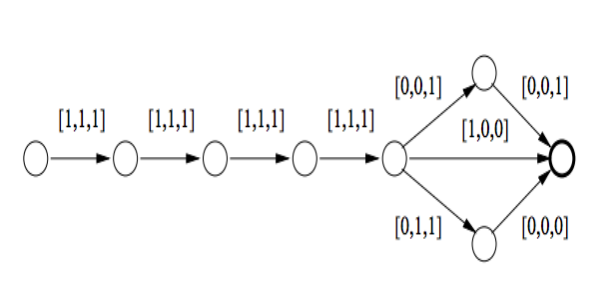
- Consider learning algorithm that "knows what it knows"
- Instead of creating $\hat{w}$, it only attempts to reason if edge costs can be obtained from available data
- After the first episode, it knows the middle path's cost is 13
- Last edge of bottom path has cost vector **[0,0,0]** which has to be 0, however penultimate edge has cost **[0,1,1]**
- Regardless of w, $w \bullet [0,1,1] = w \bullet ([1,1,1] - [1,0,0]) = w \bullet [1,1,1] - w \bullet [1,0,0] = 3-1 = 2$
- Hence, agent knows that cost of bottom is 14 which is worse than middle

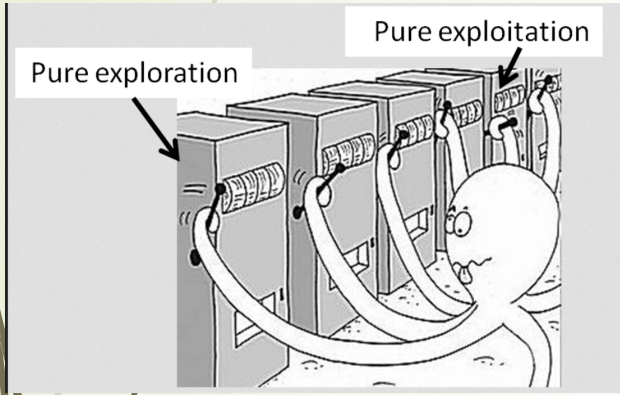
Motivation : KWIK Approach

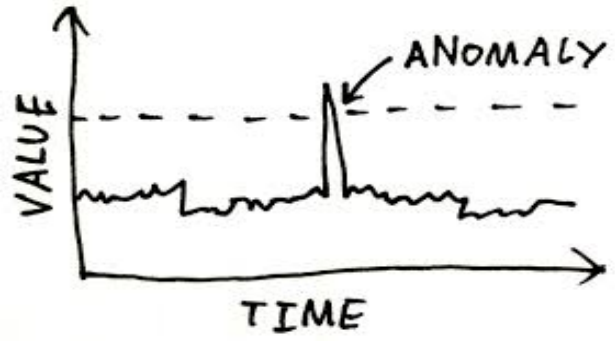


- However **[0,0,1]** on top path is linearly independent of previously seen cost vectors - (i.e.) its cost is unconstrained in the eyes of the agent
- The agent "knows that it doesn't know"
- Try out the top path in the second episode. Observes **[0,0,1]→0** allowing it to solve for $\boldsymbol{w}$ and accurately predict the cost for any vector
- Also finds that optimal path is top with high confidence

Motivation : KWIK Approach







- 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.**
- KWIK's ideation originated from its use in multi-state sequential decision making problems (like this one)
- Also action selection in bandit and associative bandit problems (bandit with inputs) can be addressed by choosing the better arm when payoffs are known and an unknown arm otherwise
- Can also be used in active learning and anomaly detection since both require some degree of reasoning if recently presented input is predictable from previous examples.

KWIK : Formal Definition

- Input set **X**, output set **Y**. Hypothesis class **H** consists of a set of functions from **X** to **Y**

$$\mathbf{H} \subseteq (\mathbf{X} \rightarrow \mathbf{Y})$$

- The target function $\boldsymbol{h}^* \in \mathbf{H}$ is source of training examples and is unknown to the learner (we assume target function is in the hypothesis class)

Protocol for a run :

- **H** and accuracy parameters $\boldsymbol{\epsilon}$ and $\boldsymbol{\delta}$ are known to both the learner and the environment
- Environment selects $\boldsymbol{h}^*$ adversarially
- Environment selects input $\boldsymbol{x} \in \mathbf{X}$ adversarially and informs the learner
- Learner predicts an output $\hat{\boldsymbol{y}} \in \mathbf{Y} \cup \{\perp\}$ where $\perp$ refers to "I don't know"
- If $\hat{\boldsymbol{y}} \neq \perp$, it should be accurate (i.e.) $|\hat{\boldsymbol{y}} - \boldsymbol{y}| \leq \boldsymbol{\epsilon}$, where $\boldsymbol{y} = \boldsymbol{h}^*(\boldsymbol{x})$. Otherwise the run is considered a failure.
- The probability of a failed run must be bounded by $\boldsymbol{\delta}$

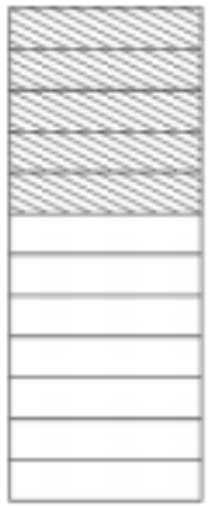
KWIK : Formal Definition

- Over a run, the total number of steps on which $\hat{y} = \perp$ must be bounded by $\mathbf{B}(\epsilon, \delta)$, ideally polynomial in $\mathbf{1}/\epsilon$, $\mathbf{1}/\delta$, and parameters defining $\mathbf{H}$.

- If $\hat{y} = \perp$, the learner makes an observation $z \in \mathbf{Z}$, of the output where
  - $z = y$ in the deterministic case
  - $z = 1$ with probability y and $z = 0$ with probability $(1 - y)$ in the Bernoulli case
  - $z = y + \eta$ for zero-mean random variable η in the additive noise case

KWIK : Connection to PAC and MB

- In all 3 frameworks – PAC (Probably Approximately Correct), MB (Mistake-Bound) and KWIK (Know What It Knows) – a series of inputs (instances) is presented to the learner.



PAC

iid inputs

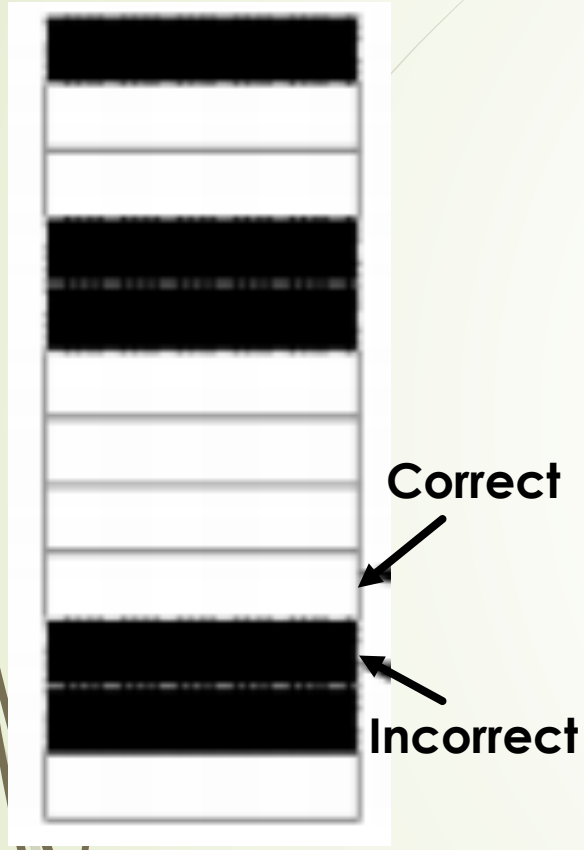
labels up front

no mistakes

Each rectangular box is an input

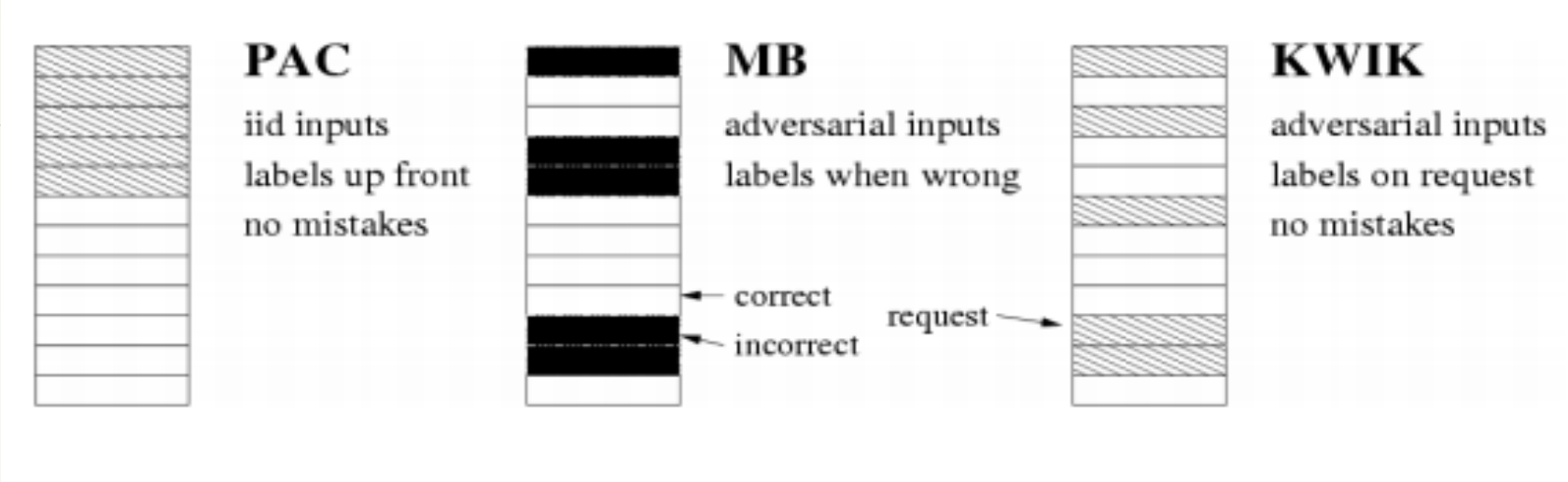
- In the PAC model, the learner is provided with labels (correct outputs) for an initial sequence of inputs, depicted by crossed boxes in figure.
- Learner is then responsible for producing accurate outputs (empty boxes) for all new inputs where inputs are drawn from a fixed distribution.

KWIK : Connection to PAC and MB



- 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 (black boxes)
- Inputs selected adversarially so there is no bound on when last mistake may be made
- MB algorithms guarantee that the *total* number of mistakes made is small (i.e.) ratio of incorrect to correct → 0 asymptotically

KWIK : Connection to PAC and MB



- KWIK model has elements of both PAC and MB. Like PAC, KWIK algorithms are 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.
- A KWIK algorithm can be turned into an MB algorithm with the same bound by having the algorithm guess an output each time its not certain.

Examples of KWIK algorithms

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

- The algorithm keeps a mapping $\hat{h}$ initialized to $\hat{h}(x) = \perp \forall\, x \in \mathbf{X}$
- The algorithm reports $\hat{h}(x)$ for every input x chosen by the environment
- If $\hat{h}(x) = \perp$, environment reports correct label y (noise-free) and algorithm reassigns $\hat{h}(x) = y$
- Since it only reports $\perp$ once for each input, the KWIK bound is $|\mathbf{X}|$

Examples of KWIK algorithms

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

- The algorithm initializes $\hat{H} = \mathbf{H}$.
- For every input x, it computes $\hat{L} = \{h(x) \mid h \in \hat{H}\}$ (i.e.) set of all outputs for all hypotheses that have not yet been ruled out.
- If $|\hat{L}| = 0$, version space has been exhausted and $h^* \notin \mathbf{H}$
- If $|\hat{L}| = 1$, all hypotheses in $\hat{H}$ agree on the output.
- If $|\hat{L}| > 1$, two hypotheses disagree, ⊥ is returned and the true label y is received. The version space is then updated such that $\hat{H}' = \{h \mid h \in \hat{H} \wedge h(x) = y\}$
- Hence the new version space reduces (i.e.) $|\hat{H}'| \leq |\hat{H}'| - 1$
- If $|\hat{H}| = 1$, then $|\hat{L}| = 1$ and the algorithm will no longer return ⊥. Hence $|\mathbf{H}|-1$ is the bound of ⊥'s that can be returned.

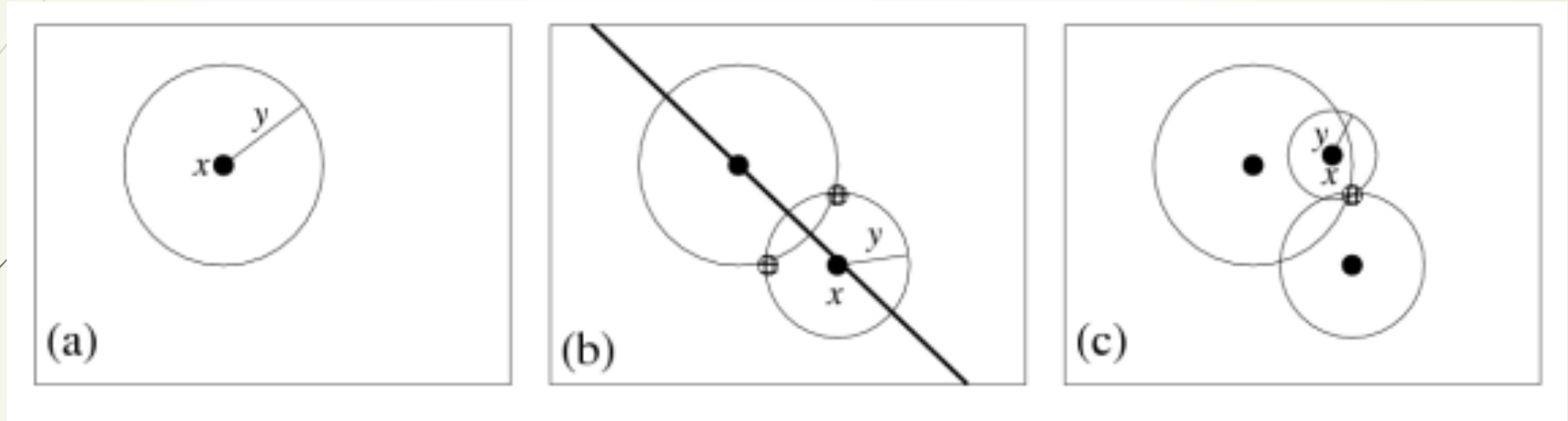
Illustrative Example

Given a group of people **P** where **a** ∈ **P** is a troublemaker and **b** ∈ **P** is a nice person who can control **a**. For any subset of people **G** ⊆ **P**, figure out if there will be trouble if you do not know the identities of **a** and **b**.

- Input space **X** = 2^P and output space **Y = {trouble, no trouble}**
- Memorization algorithm achieves bound of 2^n since it may have to see each possible subset of people (where n = |**P**|)
- Enumeration algorithm can achieve a bound of $n(n-1)$ since there is a hypothesis for each possible assignment of **a** and **b**.
- Each time the algorithm reports ⊥, it can rule out one possible combination

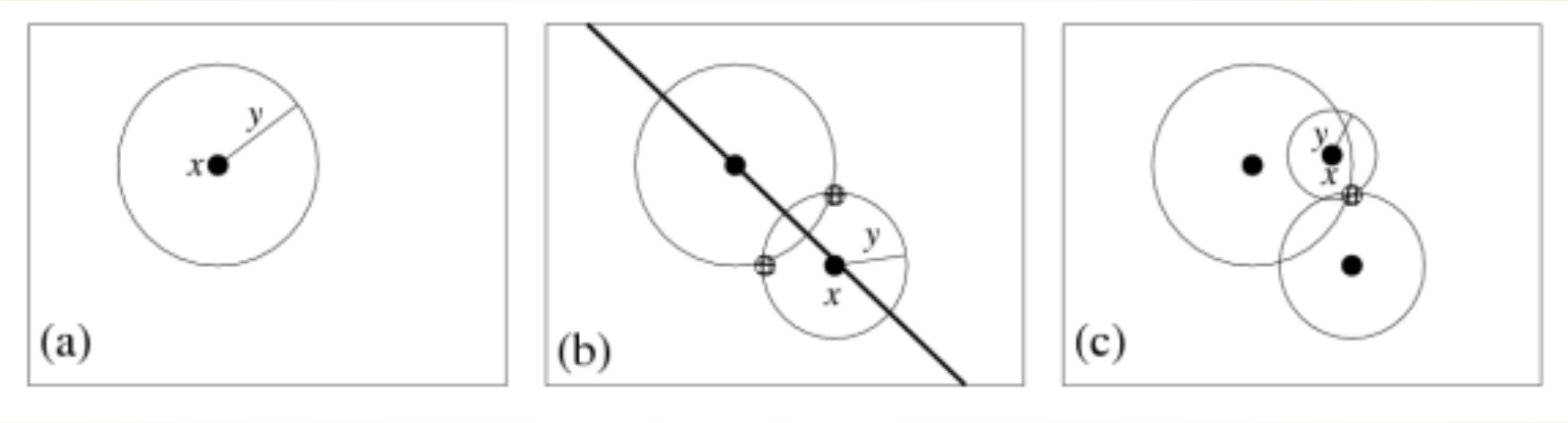
Examples of KWIK algorithms

KWIK bounds can also be achieved when the set of hypothesis class and input space are infinite



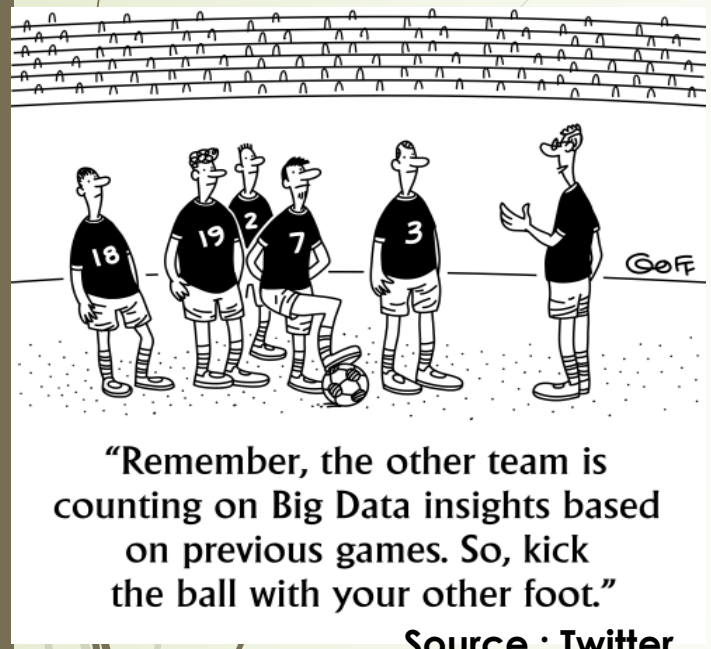
If there is an unknown point and a given target function that maps input points to their Euclidean distance ($|\boldsymbol{x} - \boldsymbol{c}|_2$) from that unknown point, the **planar distance algorithm** can learn in this hypothesis class with a KWIK bound of 3.

Examples of KWIK algorithms



- Given initial input $\boldsymbol{x}$, the algorithm returns ⊥ and receives output $\boldsymbol{y}$. Hence, it means that the unknown point must lie on the circle depicted in Figure (a)
- For a future new input, the algorithm again returns ⊥ and similarly finds a second circle (Fig (b)). If the circles intersect at one point, it becomes trivial. Else there will be 2 potential locations.
- Next, for any new input that is not collinear with the first 2 inputs, the algorithm returns ⊥ which will then allow it to identify the location of the unknown point.
- Generally, a **d-dimensional** version of this problem will have a KWIK bound of **d+1**

Examples of KWIK algorithms



"Remember, the other team is counting on Big Data insights based on previous games. So, kick the ball with your other foot."

Source : Twitter

What if the observations are not noise-free?

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 $\mathbf{B}(\epsilon, \delta) = \frac{1}{2\epsilon^2} \ln \frac{2}{\delta} = O(\frac{1}{\epsilon^2} \ln \frac{1}{\delta})$

- The unknown probability of heads is $\boldsymbol{p}$ and we want to estimate $\hat{\boldsymbol{p}}$ with high accuracy ($|\hat{\boldsymbol{p}} - \boldsymbol{p}| \leq \epsilon$) and high probability ($\mathbf{1} - \delta$)
- Since observations are noisy, we see either **1** with probability $\boldsymbol{p}$ or **0** with probability ($\mathbf{1} - \boldsymbol{p}$).
- Each time the algorithm returns $\perp$, it gets an independent trial that it can use to compute $\hat{\boldsymbol{p}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{z_t}$ where $\boldsymbol{z_t} \in \{0,1\}$ is the t'th observation in T trials.
- The number of trials needed to be ($\mathbf{1} - \delta$) certain the estimate is within ϵ can be computed using a Hoeffding bound.

Combining KWIK Learners

Can KWIK learners be combined to provide learning guarantees for more complex hypothesis classes?

Let $\mathbf{F : X \rightarrow Y}$ and $\boldsymbol{H_1, \ldots, H_k}$ be the learnable hypothesis classes with bounds of $\boldsymbol{B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)}$ where $\boldsymbol{H_i \subseteq F \; \forall \; 1 \leq i \leq k}$ (i.e.) hypothesis classes share the same input/output sets. The **union algorithm** can learn the joint hypothesis class $\mathbf{H = \cup_i H_i}$ with a KWIK bound of $\boldsymbol{B(\epsilon, \delta) = (1\text{-}k) + \sum_i B_i(\epsilon, \delta)}$

- The union algorithm maintains a set of active algorithms $\boldsymbol{\hat{A}}$, one for each hypothesis class.
- Given an input $\boldsymbol{x}$, the algorithm queries each $\boldsymbol{i \in \hat{A}}$, to obtain the prediction $\boldsymbol{\hat{y}_i}$ from each active sub-algorithm and adds it to the set $\boldsymbol{\hat{L}}$.
- If $\exists \perp \in \boldsymbol{\hat{L}}$, it returns $\perp$ and obtains correct output $\boldsymbol{y}$ and sends it to any sub-algorithm $\boldsymbol{i}$ for which $\boldsymbol{\hat{y}_i} = \perp$ to allow it to learn.
- If $\boldsymbol{|\hat{L}| > 1}$, then there is a disagreement among the sub-algorithms and again returns $\perp$ and removes any sub-algorithm that made a prediction other than $\boldsymbol{y}$ or $\perp$.
- So for each input that returns $\perp$, either a sub-algorithm reported $\perp$ (maximum $\boldsymbol{\sum_i B_i(\epsilon, \delta)}$ times) or two sub-algorithms disagreed and at least one was removed (at most $\boldsymbol{(k\text{-}1)}$ times) thus resulting in the bound.

Combining KWIK Learners : Example

Let **X = Y = $\mathfrak{R}$**. Define $\boldsymbol{H_1}$ consisting of functions mapping distance of input $\boldsymbol{x}$ from an unknown point ($|\boldsymbol{x} - \boldsymbol{c}|$). $\boldsymbol{H_1}$ can be learned with a bound of **2** using a 1-D version of the **planar algorithm**. Let $\boldsymbol{H_2}$ be the set of lines **{f |f($\boldsymbol{x}$) =$\boldsymbol{mx}$ + $\boldsymbol{b}$ }** which can also be learnt with a bound of **2** using the **regression algorithm**. We need to learn **H =** $\boldsymbol{H_1} \cup \boldsymbol{H_2}$

- Let the first input be $\boldsymbol{x_1}$**= 2**. The **union algorithm** asks $\boldsymbol{H_1}$ and $\boldsymbol{H_2}$ for the output and both return ⊥. And say it receives the output $\boldsymbol{y_1}$**= 2**
- For the next input $\boldsymbol{x_2}$**= 8**, $\boldsymbol{H_1}$ and $\boldsymbol{H_2}$ still return ⊥ and receive $\boldsymbol{y_2}$**= 4**
- Then, for the third input $\boldsymbol{x_3}$**= 1**, $\boldsymbol{H_1}$ returns **4** since (|**2-4**|**= 2** and |**8-4**|**= 4**) and $\boldsymbol{H_2}$ having computed $\boldsymbol{m}$ **= 1/3** and $\boldsymbol{b}$ **= 4/3** returns **5/3**.
- Since $\boldsymbol{H_1}$ and $\boldsymbol{H_2}$ disagree, the algorithm returns ⊥ and then finds out $\boldsymbol{y_3}$**= 3**, and then makes all future predictions accurately.

Combining KWIK Learners

What about disjoint input spaces (i.e.) $X_i \cap X_j = \emptyset$ if $i \neq j$

Let $H_1, \ldots, H_k$ be the learnable hypothesis classes with bounds of $B_1(\epsilon, \delta), \ldots, B_k(\epsilon, \delta)$ where $H_i \in (X_i \rightarrow Y)$. The **input-partition algorithm** can learn the hypothesis class $H = X_1 \cup \cdots \cup X_k \rightarrow Y$ 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 returns the response from H_i which is ϵ accurate.
- To achieve $(1-\delta)$ certainty, it insists on $(1-\delta/k)$ certainty from each of the sub-algorithms. By the union bound, the overall failure probability must be less than the sum of the failure probabilities for the sub-algorithms

Disjoint Spaces Example

- An MDP consists of $\boldsymbol{n}$ states and $\boldsymbol{m}$ 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 $\boldsymbol{n^2m}$ input space of state–action–state combinations to probabilities via Bernoulli observations.
- Thus, the problem can be solved via the **input-partition algorithm** over a set of individual probabilities learned via the **coin-learning algorithm**. The resulting KWIK bound is $\mathbf{B}(\epsilon, \delta) = \mathbf{O}(\frac{n^2m}{\epsilon^2} \ln \frac{nm}{\delta})$.

Conclusion

- The paper describes the KWIK ("Know What It Knows") model of supervised learning and identifies and generalizes key steps for efficient exploration.
- The authors provide algorithms for some basic hypothesis classes for both deterministic and noisy observations.
- They also describe methods for composing hypothesis classes to create more complex algorithms.

- Open Problem – How do you adapt the KWIK framework when the target hypothesis can be chosen from outside the hypothesis class **H**.