A Formal Proof of PAC Learnability for Decision Stumps

Joseph Tassarotti
Boston College

Jean-Baptiste Tristan
Oracle Labs

Koundinya Vajjha
University of Pittsburgh

Rigour in Machine Learning

John had proved on paper that an ML algorithm they had developed at Oracle was fair/non-discriminatory. Given the importance and subtlety of this code, he wanted to have a machine checked proof, and started to wonder what that would take.

Rigour in Machine Learning

John had proved on paper that an ML algorithm they had developed at Oracle was fair/non-discriminatory. Given the importance and subtlety of this code, he wanted to have a machine checked proof, and started to wonder what that would take.

We are starting to see machine learning systems that (on paper) are proven to provide certain guarantees, about things like privacy, fairness, or robustness. Given the importance of these properties, we should strive to give machine-checked proofs that they hold.

Motivation: ML systems with provable guarantees

Deep Learning with Differential Privacy

October 25, 2016

Martín Abadi[*] Andy Chu[*] Ian Goodfellow[†]
H. Brendan McMahan[*] Ilya Mironov[*] Kunal Talwar[*]
Li Zhang[*]

Certified Robustness to Adversarial Examples with Differential Privacy

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana
Columbia University

Challenges:

- Unlike many classic verification problems, these proofs require quite a bit of mathematical prerequisites.

Challenges:

- Unlike many classic verification problems, these proofs require quite a bit of mathematical prerequisites.
- Claims are usually about the probabilistic behavior of an algorithm.

Challenges:

- Unlike many classic verification problems, these proofs require quite a bit of mathematical prerequisites.
- Claims are usually about the probabilistic behavior of an algorithm.
- Unlike cryptographic algorithms or randomized algorithms like quicksort, we need to support probability with continuous numbers and distributions.

Challenges:

- Unlike many classic verification problems, these proofs require quite a bit of mathematical prerequisites.
- Claims are usually about the probabilistic behavior of an algorithm.
- Unlike cryptographic algorithms or randomized algorithms like quicksort, we need to support probability with continuous numbers and distributions.

Literature is not always as rigorous/detailed as we'd like. Technical conditions on lemmas are omitted, and serious details are skipped.

Case-study: generalization bound for stumps

What we did:

- Took the simplest possible example we could think of, called the **decision stump learning problem**.
- Proved a **generalization bound** about it in Lean

This theorem is often the “motivating example” used in textbooks on computational learning theory.

Case-study: generalization bound for stumps

What we did:

- Took the simplest possible example we could think of, called the **decision stump learning problem**.
- Proved a **generalization bound** about it in Lean

This theorem is often the "motivating example" used in textbooks on computational learning theory.

Goals:

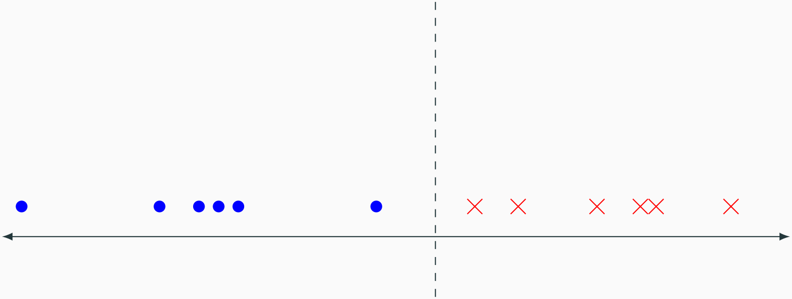
- Exercise libraries, see what else is needed
- Warm-up for more advanced results.

Stump Learning

Stump Learning

Goal is to learn to distinguish two classes of items. Blue o's and red x's.

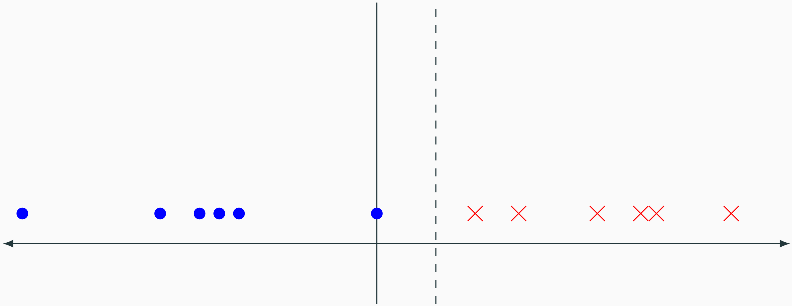
Stump Learning



Goal is to learn to distinguish two classes of items. Blue o's and red x's.

An unknown boundary value, represented by the dashed line, separates the classes.

Stump Learning



A reasonable thing to do is to take the largest training example that's a circle, and use its value as the boundary.

Stump Learning

This turns out to work well: we can prove that given enough training examples, the classifier we obtain this way can be made to have arbitrarily small error, with high probability.

Stump Learning

This turns out to work well: we can prove that given enough training examples, the classifier we obtain this way can be made to have arbitrarily small error, with high probability. Formally,

- Let $\mathcal{X}$ be $[0, \infty)$.
- The concept class of *decision stumps* $\mathcal{C}$ is the subset of $\{0, 1\}^{\mathcal{X}}$ defined as $\{\lambda x. \mathbb{1}(x \leq c) \mid c \in \mathcal{X}\}$. Each element in $\mathcal{C}$ is a red-blue labelling as above.

Stump Learning

Theorem (Informal)
There exists a learning function (algorithm) $\mathcal{A}$ and a sample complexity function m such that for any distribution μ over $\mathcal{X}$, $c \in \mathcal{C}$, $\epsilon \in (0, 1)$, and $\delta \in (0, 1)$, when running the learning function $\mathcal{A}$ on $n \geq m(\epsilon, \delta)$ i.i.d. samples from μ labeled by c, $\mathcal{A}$ returns a hypothesis $h \in \mathcal{C}$ such that, with probability at least $1 - \delta$,

$$\mu(\{x \in \mathcal{X} \mid h(x) \neq c(x)\}) \leq \epsilon$$

Sketch of Proof

Because a decision stump is entirely determined by the boundary value it uses for decisions, we will refer to a stump and its boundary value interchangeably.

Sketch of Proof

Because a decision stump is entirely determined by the boundary value it uses for decisions, we will refer to a stump and its boundary value interchangeably.

Note also that the error probability of a hypothesis is the measure of the interval between it and the target.

$$\mu\{x : h(x) \neq c(x)\} = \mu(h, c]$$

Sketch of Proof

- Given μ and ϵ, consider (random i.i.d) labeled samples $(X_1, l_1), \ldots, (X_n, l_n)$.
- The learning function $\mathcal{A}$ takes the above as input and returns the hypothesis

$$h = \lambda x. \mathbb{1}(x \leq \max\{X_i \mid l_i = 1\})$$

- If $\mu(0, c] < \epsilon$ then the bound is trivial. (Since $(h, c] \subseteq (0, c]$). So error is bounded with probability 1.

So assume $\mu(0, c] \geq \epsilon$.

Sketch of Proof

- Find a θ such that $\mu[\theta, c] = \epsilon$. Call $\mathcal{I} = [\theta, c]$.
- If the boundary point h, selected by $\mathcal{A}$, is in $\mathcal{I} = [\theta, c]$ then we have $\mu(h, c] \leq \mu[\theta, c] = \epsilon$. So the error is bounded by epsilon.

Sketch of Proof

- Find a θ such that $\mu[\theta, c] = \epsilon$. Call $\mathcal{I} = [\theta, c]$.
- If the boundary point h, selected by $\mathcal{A}$, is in $\mathcal{I} = [\theta, c]$ then we have $\mu(h, c] \leq \mu[\theta, c] = \epsilon$. So the error is bounded by epsilon.
- So for the error $\mu(h, c] \geq \epsilon$, this means that θ lies inside $(h, c]$.

Sketch of Proof

- Find a θ such that $\mu[\theta, c] = \epsilon$. Call $\mathcal{I} = [\theta, c]$.
- If the boundary point h, selected by $\mathcal{A}$, is in $\mathcal{I} = [\theta, c]$ then we have $\mu(h, c] \leq \mu[\theta, c] = \epsilon$. So the error is bounded by epsilon.
- So for the error $\mu(h, c] \geq \epsilon$, this means that θ lies inside $(h, c]$.
- By the choice of h, it means that none of our samples X_i came from $\mathcal{I}$.

Sketch of Proof

- Find a θ such that $\mu[\theta, c] = \epsilon$. Call $\mathcal{I} = [\theta, c]$.
- If the boundary point h, selected by $\mathcal{A}$, is in $\mathcal{I} = [\theta, c]$ then we have $\mu(h, c] \leq \mu[\theta, c] = \epsilon$. So the error is bounded by epsilon.
- So for the error $\mu(h, c] \geq \epsilon$, this means that θ lies inside $(h, c]$.
- By the choice of h, it means that none of our samples X_i came from $\mathcal{I}$.
- The probability of a point X_i not belonging to $\mathcal{I}$ has probability at most $1 - \epsilon$.

Sketch of Proof

- Find a θ such that $\mu[\theta, c] = \epsilon$. Call $\mathcal{I} = [\theta, c]$.
- If the boundary point h, selected by $\mathcal{A}$, is in $\mathcal{I} = [\theta, c]$ then we have $\mu(h, c] \leq \mu[\theta, c] = \epsilon$. So the error is bounded by epsilon.
- So for the error $\mu(h, c] \geq \epsilon$, this means that θ lies inside $(h, c]$.
- By the choice of h, it means that none of our samples X_i came from $\mathcal{I}$.
- The probability of a point X_i not belonging to $\mathcal{I}$ has probability at most $1 - \epsilon$.
- Since samples are i.i.d, the probability is atmost $(1 - \epsilon)^n$.

Choose an appropriate m such that for $n \geq m$ we get the result.

Problem!

Problem!

Such a θ may not exist!

Problem!

Such a θ may not exist!

Indeed, take μ to be the Bernoulli distribution with $p = .5$, $c = .5$, and $\epsilon = .25$. Then the desired θ does not exist.

Problem!

Such a θ may not exist!

Indeed, take μ to be the Bernoulli distribution with $p = .5$, $c = .5$, and $\epsilon = .25$. Then the desired θ does not exist.

Since PAC learning is distribution free, we need to account for all distributions μ, including those with a discrete component.

Fixing the proof

The point θ we are looking for should be defined as

$$\theta = \sup\{x \in \mathcal{X} \mid \mu[x, c] \geq \epsilon\}$$

and we not only need to prove that θ satisfies $\mu[\theta, c] \geq \epsilon$ but *also* that $\mu(\theta, c] \leq \epsilon$.

Fixing the proof

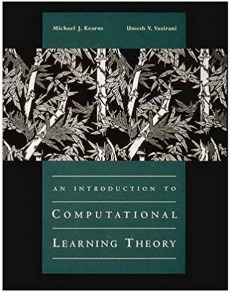
The point θ we are looking for should be defined as

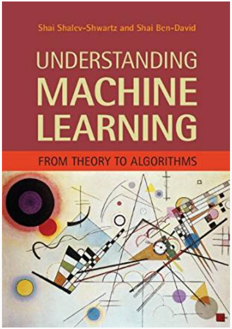
$$\theta = \sup\{x \in \mathcal{X} \mid \mu[x, c] \geq \epsilon\}$$

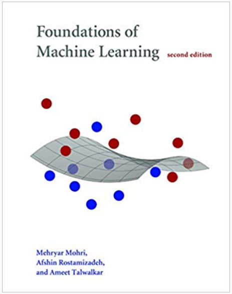
and we not only need to prove that θ satisfies $\mu[\theta, c] \geq \epsilon$ but *also* that $\mu(\theta, c] \leq \epsilon$.

Using this we fix the proof. And this proof has now been formalized.

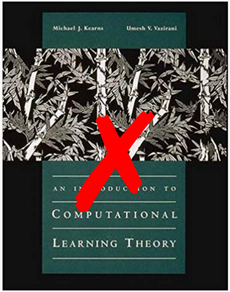
Proofs in Textbooks?

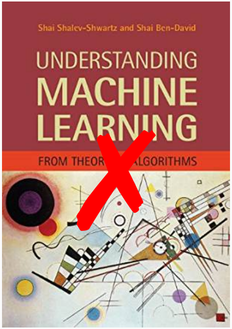


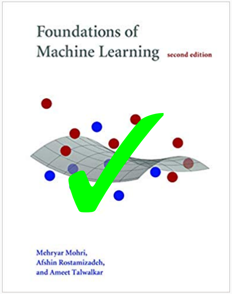




Proofs in Textbooks?







The one correct proof still omits the hardest part! ("Not hard to see...")

It Gets Worse

For stumps, the theorem is true even if many proofs are wrong.

Many other results in these books are **wrong as stated**.

Problem: they omit “technical” conditions about measurability.

It Gets Worse

For stumps, the theorem is true even if many proofs are wrong.

Many other results in these books are **wrong as stated**.

Problem: they omit "technical" conditions about measurability.

Challenge: can formalization automate these tedious details? Or provide a more "intuitive" interface that is checked?

Expressing the algorithm

A major challenge is formally describing the learning algorithm.

Three “stages”:

1. Draw a random sample of labeled training examples
2. Run the learning algorithm
3. Consider behavior of returned classifier on test examples

Enter Giry Monad

Enter Giry Monad

- The Giry monad lets us rigorously formalize certain common informal arguments in probability theory.

Enter Giry Monad

- The Giry monad lets us rigorously formalize certain common informal arguments in probability theory.
- Provides a natural denotational semantics for (a subset of) probabilistic programming languages.

Enter Giry Monad

- The Giry monad lets us rigorously formalize certain common informal arguments in probability theory.
- Provides a natural denotational semantics for (a subset of) probabilistic programming languages.
- Allows us to perform *ad hoc* notation overloading.

Enter Giry Monad

- The Giry monad lets us rigorously formalize certain common informal arguments in probability theory.
- Provides a natural denotational semantics for (a subset of) probabilistic programming languages.
- Allows us to perform *ad hoc* notation overloading.
- Simplifies certain constructions.

What is it?

- Let **Meas** be the category of all measurable spaces together with measurable maps.
- If $M \in$ **Meas**, then let $\mathcal{P}(M)$ stand for all of the *(probability) measures* on M.
- For measurable functions $f : M \to \mathbb{R}$, this space comes naturally equipped with the maps $\tau_f : \mathcal{P}(M) \to \mathbb{R}$ which are given by

$$\tau_f(\nu) = \int_M f d\nu$$

- Note that if $f = \chi_A$, the indicator function of a measurable set A, then $\tau_A(\nu) = \nu(A)$.

What is it?

- $\mathcal{P}(M)$ can be equipped with a topology the weak* topology which is the smallest topology on $\mathcal{P}(M)$ which makes the maps $\{\tau_f : \mathcal{P}(M) \to \mathbb{R}\}$ (for measurable f), continuous.
- Now that we have a topology on $\mathcal{P}(M)$, we can talk about the Borel σ-algebra of $\mathcal{P}(M)$ as the smallest σ-algebra generated by the functions $\{\tau_f\}$. So we get that $\mathcal{P}(M) \in$ **Meas**.
- Given a measurable function $f : M \to M$, we have the pushforward map $\mathcal{P}(f) : \mathcal{P}(M) \to \mathcal{P}(M)$ given by $\mathcal{P}(f)\nu = \nu \circ f^{-1}$.
- The above three points now show that $\mathcal{P} \in \mathcal{E}($**Meas**$)$, the category endofunctors of **Meas**.

Bind and Return

Fix an arbitrary $M \in \mathbf{Meas}$.

Let us now define the natural transformation $\eta : \mathbf{1} \to \mathcal{P}$ in componentwise as

$$\begin{aligned} \eta_M : M = \mathbf{1}_M &\longrightarrow \mathcal{P}(M) \\ x &\mapsto \left(A \mapsto \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases} \right) \end{aligned}$$

That is, $\eta_M(x)$ is the dirac measure at x.

Bind and Return

We also find the following definition for the bind operator.

$$\gg\!=: \mathcal{P}(M) \to (M \to \mathcal{P}(N)) \to \mathcal{P}(N)$$

$$(\rho \gg\!= g)(f) = \int_M \left\{\lambda m. \int_N f dg(m)\right\} d\rho.$$

- $(\mu \gg\!= f)$ is "simply computing the distribution that results from applying f while marginalizing over ρ".

Bind and Return

We also find the following definition for the bind operator.

$$\gg\!=: \mathcal{P}(M) \to (M \to \mathcal{P}(N)) \to \mathcal{P}(N)$$

$$(\rho \gg\!= g)(f) = \int_M \left\{ \lambda m. \int_N f dg(m) \right\} d\rho.$$

- $(\mu \gg\!= f)$ is "simply computing the distribution that results from applying f while marginalizing over ρ".
- Monad laws hold subject to **measurability** conditions. (Which is why we can't make it part of the monad typeclass in Lean)

Giry Monad for Learning Algorithms

We express learning algorithms in Lean using **Giry Monad**:

$$(\mu \gg= \lambda x.fx) := x \leftarrow \mu; \\ f(x)$$

" Sample from μ as x, continue by $f(x)$ "

Giry Monad for Learning Algorithms

$$\begin{aligned}
&\text{training} \leftarrow \text{sample}(n, \mu);\\
&\text{c} \leftarrow \text{learn}(\text{training});\\
&\text{test} \leftarrow \text{sample}(m, \mu);\\
&\text{return score}(\text{c}, \text{test})
\end{aligned}$$

Can reason about separate stages to derive bound on overall.

Giry Monad for Probabilistic Constructions

- We can also describe many probabilistic problems using probabilistic programs, e.g. draw a normal value ‘x‘, and depending on it a normal value ‘y‘ with variance ‘x‘:

$$\begin{aligned} &\mathsf{x} \leftarrow \mathsf{Normal}(0, 1); \\ &\mathsf{y} \leftarrow \mathsf{Normal}(0, x); \\ &\mathsf{return}\ (\mathsf{x}, \mathsf{y}) \end{aligned}$$

Giry Monad for Probabilistic Constructions

- We can also describe many probabilistic problems using probabilistic programs, e.g. draw a normal value ‘x‘, and depending on it a normal value ‘y‘ with variance ‘x‘:

$$
\begin{aligned}
&\mathrm{x} \leftarrow \mathrm{Normal}(0, 1); \\
&\mathrm{y} \leftarrow \mathrm{Normal}(0, x); \\
&\mathrm{return}\ (\mathrm{x}, \mathrm{y})
\end{aligned}
$$

- Construction of the product measure.

Giry Monad for Stump Learning

Theorem
Let $\mathcal{H} = \{\lambda x.\mathbb{1}(x \leq c) \mid c \in \mathbb{R}_+\}$ *be the class of decision stumps. There exists a measurable function* $\mathcal{A} : \Pi n \to (\mathbb{R}_+ \times \{0,1\})^n \to \mathcal{H}$*, called the learning function, and a sample complexity function* $m : (0,1)^2 \to \mathbb{N}$ *such that for any probability measure* μ *on the measurable space* $(\mathbb{R}_+, \mathcal{B}(\mathbb{R}_+))$*,* $(\epsilon, \delta) \in (0,1)^2$*, and for any* $n \geq m(\epsilon, \delta)$

$$\mathcal{A}_*(c_*(\mu_n))\{h \in \mathcal{H} \mid \mu\{x \in \mathbb{R}_+ \mid h(x) \neq c(x)\} \geq \epsilon\} \geq 1 - \delta$$

Giry Monad for Stump Learning

Here, for μ a measure:

$$f_*(\mu)(A) = \mu(f^{-1}(A))$$

and

$$\begin{aligned} \mu_1 &= \mu \\ \mu_n &= v \leftarrow \mu_{n-1}\,;\ \omega \leftarrow \mu\,;\ \text{ret}(\omega, v) \end{aligned}$$

is the n-fold product measure.

Giry Monad for Probabilistic Programs

- A probabilistic program is interpreted as a parameterized probability distribution, i.e., a measurable arrow $A \to \mathcal{P}B$. (These are nothing but the Kleisli arrows of the Giry monad.)

Giry Monad for Probabilistic Programs

- A probabilistic program is interpreted as a parameterized probability distribution, i.e., a measurable arrow $A \to \mathcal{P}B$. (These are nothing but the Kleisli arrows of the Giry monad.)
- The Giry monad allows us to combine such probabilistic programs.

Giry Monad for Probabilistic Programs

However, not everything is so nice.

- By a classical result of Aumann, there is no generic measurable space structure on the function space $\alpha \to \beta$ which makes the evaluation maps continuous. This means the Giry monad cannot eat it!

Giry Monad for Probabilistic Programs

However, not everything is so nice.

- By a classical result of Aumann, there is no generic measurable space structure on the function space $\alpha \to \beta$ which makes the evaluation maps continuous. This means the Giry monad cannot eat it!
- So we can't pass around higher order functions in the monad.

In Conclusion

In Conclusion

1. Lot of work to be done still to formalize Learning theory.
2. Monadic abstractions can help conveniently structure formal proofs.

In Conclusion

1. Lot of work to be done still to formalize Learning theory.
2. Monadic abstractions can help conveniently structure formal proofs.
3. Other monads?