WHAT IT MEANS FOR A MACHINE TO LEARN

JIHONG CAI AND PARTH DESHMUKH

1. Introduction

ChatGPT has recently gained significant attention. Many consider it an early model of general AI, given its capability to handle a wide range of tasks, from crafting recipes to writing and annotating code. While some speculate that it might one day replace search engines like Google, others are recalling the story of an apple and a young Englishman who passed away 70 years ago.

At the age of 24, Alan Turing astounded the world by publishing a paper that introduced the concept of the Turing machine, a foundational protocol for all modern computers. He posited that anything computable can be emulated by the Turing machine. This proposition is known as the Church-Turing thesis.

The origins of machine learning are deeply rooted in philosophical inquiries into the essence of intelligence. What does it mean to be intelligent, and can this quality be replicated or simulated? Alan Turing, a pioneering figure in the realm of computer science and artificial intelligence, approached this profound question with pragmatism. Instead of delving into the abstract nature of thought, Turing posed a more tangible query: Can machines be designed to imitate human intelligence? His perspective shifted the discourse from a purely theoretical debate to a practical challenge, paving the way for the development and evolution of machine learning as we understand it today.

In *Computing Machinery and Intelligence* (1950), Turing begins by posing the intriguing question, "Can Machines Think?" Rather than dwell on the philosophical nuances of "thinking," he introduces a practical measure known as the Turing Test. In this test, a machine is deemed intelligent if a human evaluator, through conversation, cannot distinguish it from another human. This innovative approach shifted the discourse from abstract philosophical debates to tangible challenges in machine behavior and capabilities. Turing delves into the potential of digital computers, suggesting that with the right programming, they could replicate any human intellectual activity. He even anticipates the advent of machine learning, envisioning machines that can adapt and improve over time. Throughout the paper, he also confronts and rebuts various objections to the concept of machine intelligence, making a compelling case for the possibilities that lie ahead in the realm of AI.

One of the most significant challenges in realizing general AI has been the constraints of resources and computing power. Even if one designs the perfect algorithm for a machine to execute a task or to self-adapt, the resource requirements might be astronomical. Consequently, the potential of machine learning couldn't be fully

This note is for the MATRIX event on 10/25/2023. The textbook this note is based off of is available for free at https://www.cs.huji.ac.il/ shais/UnderstandingMachineLearning/.

explored until there were substantial advancements in computing technologies. In recent years, breakthroughs in computer engineering have made it possible for humans to implement simple self-adaptive learning tasks for computers. As a result, machine learning has garnered immense traction and attention.

2. Classifiers and Regressors

The first thing we need to do is define what we are trying to learn. Suppose we are running a linear regression on 100 points, finding the best-fit straight line that describes the relationship. Each point p has an x- and y-coordinate p_x and p_y, and we want to learn the correct linear equation as a function of x that maps $p_x \longmapsto p_y$ as best as possible. For example, if we want to find the line that describe the relation of the set $\{(1,2),(2,4),(3,6),\ldots,(99,198),(100,200)\}$, the linear equation we are looking for is $y = 2x$.

We can formalize this idea more broadly as follows. The **domain set** $\mathcal{X}$ contains the objects x we are mapping or labeling, and the **label set** $\mathcal{Y}$ contains the labels y we map the objects in $\mathcal{X}$ to. Our **training data** is a set of tuples $p = (p_x, p_y)$ in $\mathcal{X} \times \mathcal{Y}$, which we assume are the output of some "correct" labeling function f. That is to say, there exists some function $f : \mathcal{X} \to \mathcal{Y}$ such that for all p, $f(p_x) = p_y$. Our goal is to guess what this function f is, or at least try to approximate it somehow. Importantly, we want to guess what it is over the *entire* domain $\mathcal{X}$, not just the subset $\{p_x\} \subseteq \mathcal{X}$.

So our machine learning algorithm is an algorithm that has as its input the domain set $\mathcal{X}$, label set $\mathcal{Y}$, and training data $S = \{(p_x, p_y)\}$ and outputs a function $h : \mathcal{X} \to \mathcal{Y}$. This output function is called the predictor, hypothesis, or classifier depending on the specific use case. ML/AI researchers themselves aren't incredibly consistent with their terminology, so we will define them as such:

- A **hypothesis** is the general case, e.g. a function $h : \mathcal{X} \to \mathcal{Y}$ output by some algorithm A. We denote $A(S)$ to be the hypothesis h output by A with training data S.
- A **classifier** is a hypothesis where the label set $\mathcal{Y}$ is finite, e.g. categorical. For example, a binary classifier is a hypothesis where $\mathcal{Y} = \{0,1\}$ or $\{-1,1\}$.[1]
- A **predictor** is a hypothesis where the label set $\mathcal{Y}$ is infinite; for example, $\mathcal{Y} = \mathbb{R}$. A linear regression is one example.

Finally, we need a way to measure how good or bad we are at approximating f. For this, we define a **true error** or **loss function** $L_f(h)$ that in some way measures how different the hypothesis h and correct labeling function f are. Usually, $L_f(h)$ outputs a number or probability, so the function signature is

$$L_f(h) : (\mathcal{X} \to \mathcal{Y}) \times (\mathcal{X} \to \mathcal{Y}) \to \mathbb{R} \text{ (or } [0,1])$$

Usually, the loss function is defined with respect to f, so the signature is simplified to $L(h) : (\mathcal{X} \to \mathcal{Y}) \to \mathbb{R}$.[2] Remember that we have no access to f, so we cannot calculate the true error over the whole domain $\mathcal{X}$. Thus we calculate a **empirical error** L_S which is L restricted to the training data.

[1] The values of the labels doesn't matter, as long as they're two distinct labels. Which formulation is chosen is just what makes proving that problem easier.

[2] Functions that take functions as arguments are called *higher order functions.* They pop up often in math and are even more ubiquitous in computer science.

3. PAC Learnability

The objective of machine learning theory is to design algorithms that output useful hypotheses. No algorithm can output every possible function $\mathcal{X} \to \mathcal{Y}$, so we define **hypothesis classes.** A hypothesis class $\mathcal{H}$ is a class of similar functions $\mathcal{X} \to \mathcal{Y}$, and we restrict our algorithm to only output a hypothesis $h \in \mathcal{H}$. For example, the hypothesis class of single-variable linear regressions is the class of functions $\mathbb{R} \to \mathbb{R}$ such that

$$h(x) = a + bx \text{ for some } a, b \in \mathbb{R}$$

We want to know whether a hypothesis class can do a machine learning task. One way to answer this is to define whether it is **PAC-learnable**, short for "Probably Approximately Correct-learnable." The actual definition of PAC-learnable is very tricky, so we will loosely define it.

Definition. *Let $\mathcal{H}$ be a hypothesis class of functions $\mathcal{X} \to \mathcal{Y}$, and let $\varepsilon > 0, \delta \in (0,1)$. $\mathcal{H}$ is **PAC-learnable** if there exists an algorithm such that for any potential correct labeling function $f : \mathcal{X} \to \mathcal{Y}$ and for any randomly selected training data S of size at least $N(\epsilon, \delta)$, the algorithm returns a hypothesis $h \in \mathcal{H}$ such that there is at least $1 - \delta$ probability that the training error is bounded by*

$$L_S(h) \leq \varepsilon.$$

PAC-learnability, by default, assumes realizability. Realizability means that there exists $h^* \in \mathcal{H}$ such that the true error is completely minimized, e.g. $L_f(h^*) = 0$. So we want to use *agnostic PAC-learnability,* which drops that assumption. In agnostic PAC-learnability, the bound is given by

$$L_S(h) \leq \min_{h \in \mathcal{H}} L_f(h) + \varepsilon.$$

To recover the definition for PAC-learnability, simply note that if h^* exists,

$$\min_{h \in \mathcal{H}} L_f(h) = 0.$$

δ is our confidence parameter, corresponding to the "probably", and ε is an error bound, corresponding to the "approximately correct." What this definition says is that we can write an algorithm for $\mathcal{H}$ that can get us within a reasonable error bound of *any* desired labeling function with reasonable confidence with the only requirement being a sufficiently large training dataset.

One nice thing we can show with this definition is that if our hypothesis class is finite, it is agnostic PAC-learnable. This is especially nice because every practical hypothesis class is finite; we only have finite precision arithmetic on computers, no matter how big the computer.

4. No Free Lunch

PAC-learnability gives us a way to identify if a hypothesis class can be used to learn any labeling function. One might ask why we have to make many different hypothesis classes and. algorithms instead of just using one PAC-learnable hypothesis class. The answer to that is the No Free Lunch theorem, which tells us that for any learning algorithm, there always exists a task that learning algorithm will fail on. The statement

and proof for the No Free Lunch theorem is once again complicated, so we will do sketches.

Theorem. *(No Free Lunch) Let A be any learning algorithm over the domain $\mathcal{X}$ and labels $\mathcal{Y}$ with respect to some loss function $L(h)$. Let m be any finite number less than $|\mathcal{X}|/2$, representing the size of the training data. Then there exists some $f : \mathcal{X} \to \mathcal{Y}$ such that PAC-learnability fails; that is, that we cannot bound the true error $L_f(h)$ with reasonable confidence.*

Suppose we are doing binary classification, so let $\mathcal{Y} = \{-1, 1\}$, and let our training data be $S = \{x_1, x_2, \dots x_m\}$. For any hypothesis $A(S) = h$ that our algorithm returns, h has to give some labeling to every element in $\mathcal{X}$.

We can make h fail by taking advantage of the fact that the algorithm only gets to see less than half of the elements in $\mathcal{X}$. Set the correct labeling function to be

$$f : \mathcal{X} \to \{-1, 1\} : \ f(x) = \begin{cases} h(x) & \text{if } x \in S \\ -h(x) & \text{if } x \in \mathcal{X}/S \end{cases}$$

A only has access to the training data S and empirical error $L_S(h)$, so it will return h, but we have set it so outside of the training data, f is exactly opposite of h. Therefore our algorithm has failed to bound the true error.

The No Free Lunch theorem exists because there are many functions $\mathcal{X} \to \{-1, 1\}$. This lets us always pick an adversarial task the algorithm will fail on. The No Free Lunch theorem is why prior knowledge is important in machine learning, because if we don't pick the right algorithm and hypothesis class for the task, there is a possibility we run into an adversarial example. One might ask whether we can make our hypothesis class the set of all functions $\mathcal{X} \to \{-1, 1\}$. However, if $\mathcal{X}$ is infinite, that hypothesis class is not PAC-learnable. To understand why, we turn to VC-dimension.

5. VC-Dimension and Shattering

PAC-learnability is a nice definition to have, but it is very unwieldy to work with. Hence we look at it from another angle. Recall that in the No Free Lunch theorem, the proof relies on constructing an adversarial example; the VC-dimension[3] will focus on the behavior of a hypothesis class faced with adversarial examples.

We will focus on binary classification, as VC-dimension only applies there. Let $C \subset \mathcal{X}$. When constructing an adversarial example, we had taken some function from the set of all functions C to $\{0, 1\}$. Define the restriction of $\mathcal{H}$ onto C as the set of functions $C \to \{0, 1\}$ that can be derived from $\mathcal{H}$. Formally,

$$\mathcal{H}_C = \{q : C \to \{0, 1\} \mid \exists\, h \in \mathcal{H} \text{ such that } q(c) = h(c)\ \forall\ c \in C\}$$

$\mathcal{H}$ **shatters** C if the restriction $\mathcal{H}_C$ equals the set of all functions $C \to \{0, 1\}$. This means for any classification of the elements in C, we can perfectly match that with some hypothesis h.

We can finally define the VC-dimension:

Definition. *The VC-dimension of a hypothesis class $\mathcal{H}$ of functions $\mathcal{X} \to \{0, 1\}$ is the cardinality of the largest subset $C \subset \mathcal{X}$ that $\mathcal{H}$ shatters.*

[3]Short for Vapnik–Chervonenkis dimension, named after its inventors Vladimir Vapnik and Alexey Chervonenkis.

Recall that finite hypothesis classes are agnostic PAC-learnable. It turns out that there is a deep link between VC-dimension and PAC-learnability:

Theorem. *(Fundamental Theorem of Statistical Learning) Let $\mathcal{H}$ be a hypothesis class of binary classifiers $\mathcal{X} \to \{0,1\}$ and let the loss function be the 0-1 loss. The following are equivalent:*

(1) *$\mathcal{H}$ has finite VC-dimension.*
(2) *$\mathcal{H}$ is agnostic PAC-learnable.*
(3) *$\mathcal{H}$ is PAC-learnable.*
(4) *Any ERM rule (minimizing empirical error) is an agnostic PAC-learner for $\mathcal{H}$.*
(5) *Any ERM rule (minimizing empirical error) is a PAC-learner for $\mathcal{H}$.*

If $\mathcal{H}$ has finite VC-dimension, not only can we use it to learn any function, but we can do so by minimizing empirical error which we can actually calculate. Note here that we get both agnostic PAC-learnability and PAC-learnability since we chose a specific loss function that returns a probability, but it is sufficient to have either.

The intuitive reason why we want the VC-dimension to be finite is that the set of all functions from C to $\{0,1\}$ is very large, and if C is infinite, a bigger infinity.[4] If we have an infinite number of hypotheses to use for some adversarial example, we have no way of picking between them. By having everything, we know nothing.

6. Threshold Functions and Halfspaces

We can immediately apply VC-dimension to analyze some binary classifiers and see if they are PAC-learnable. The first we will cover is the hypothesis class of **threshold functions.** A threshold function is defined as:

$$\mathbf{1}_c(x) = \begin{cases} 0 & \text{if } x < c \\ 1 & \text{otherwise} \end{cases}$$

To show a hypothesis class has VC-dimension d, we want to show that there exists a set of size d that can be shattered (the maximal subset) and that no subset of size $d+1$ can be shattered. For threshold functions, we can always shatter a single point by placing the threshold left or right of it. Suppose we have a subset $\{3,5\}$ and label them $\{1,0\}$ respectively. By definition, if the threshold function labels 5 as 0, then 3 is also labeled 0, and vice versa, if 3 is labeled 1, 5 must also be labeled 1. Thus we cannot shatter this subset and the VC-dimension of threshold functions is 1.

A more useful example is that of **halfspaces.** Halfspaces extend the idea of splitting the real line in two to arbitrary dimensions, and can be flipped around as well. A halfspace in d dimensions is given by:

$$HS_d(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

The vector $\mathbf{w}$ is the normal vector to a splitting plane, which is then offset by the "bias" b. The sign function tells us what side of the plane $\mathbf{x}$ is on. If it is on the side of the plane $\mathbf{w}$ points to, it is labeled 1, otherwise it is labeled -1. Training halfspaces can be done with linear programming.

[4] Specifically what that bigger infinity is is called the continuum hypothesis, which is undecidable under ZFC.

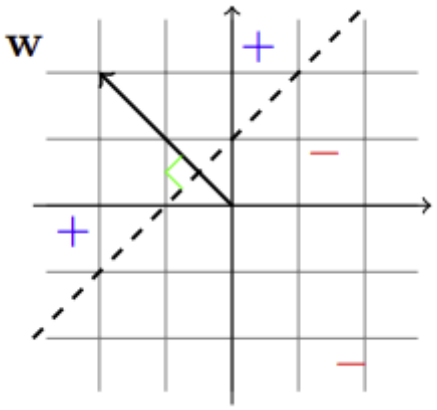

FIGURE 1. A visual example of a halfspace in 2 dimensions.

For a halfspace in d dimensions, it turns out the VC-dimension is $d+1$. First, we can rewrite the formula for halfspaces by noting that if we do the changes

$$\mathbf{w'} = \begin{pmatrix} b, & w_1, & w_2, & \dots & w_d \end{pmatrix}$$
$$\mathbf{x'} = \begin{pmatrix} 1, & x_1, & x_2, & \dots & x_d \end{pmatrix}$$

we absorb the bias into the dot product:

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = \langle \mathbf{w'}, \mathbf{x'} \rangle$$

This is called a *homogenous halfspace* over $\mathbb{R}^d$. It's easier to prove the VC-dimension in this form, but note that we've added one more dimension to our halfspace. Thus if we show the VC-dimension of a homogenous halfspace in d dimensions is d, that is equivalent to showing the VC-dimension of regular (nonhomogenous) halfspaces in d dimensions to be $d+1$.

Let C be a set of $d+1$ vectors x_i in $\mathbb{R}^d$. From linear algebra, we know that the maximal size of a spanning set in $\mathbb{R}^d$ is d. Therefore, there exist nonzero a_i such that

$$\sum_{i=1}^{d+1} a_i \mathbf{x}_i = \mathbf{0}$$

Let I be the set of positive a_i and J be the set of negative a_i. We can split the above sum into

$$\sum_{i \in I} a_i \mathbf{x}_i = \sum_{j \in J} |a_j| \mathbf{x}_i$$

Next, suppose we can shatter C. This means that the normal vector to our splitting plane is $\mathbf{w}$ such that $\langle \mathbf{w}, \mathbf{x}_i \rangle > 0$ for all $i \in I$ and $\langle \mathbf{w}, \mathbf{x}_j \rangle < 0$ for all $j \in J$. Therefore

$$0 < \sum_{i \in I} a_i \langle \mathbf{x}_i, w \rangle = \left\langle \sum_{i \in I} a_i \mathbf{x}_i, \mathbf{w} \right\rangle = \left\langle \sum_{j \in J} a_j \mathbf{x}_j, \mathbf{w} \right\rangle = \sum_{j \in J} |a_j| \langle \mathbf{x}_j, w \rangle < 0$$

which is a contradiction. We have shown that for homogenous halfspaces, the VC-dimension is d. Therefore the VC-dimension of nonhomogenous halfspaces is $d+1$.

7. Decision Trees

Decision trees are some of the most common and useful models in modern ML. A decision tree is a tree where each internal node is a decision node and each leaf node is a prediction node. When inputting a data point, the decision tree sends that point to one of the leaf nodes depending on the decisions taken in the internal nodes and returns the output in the leaf node.

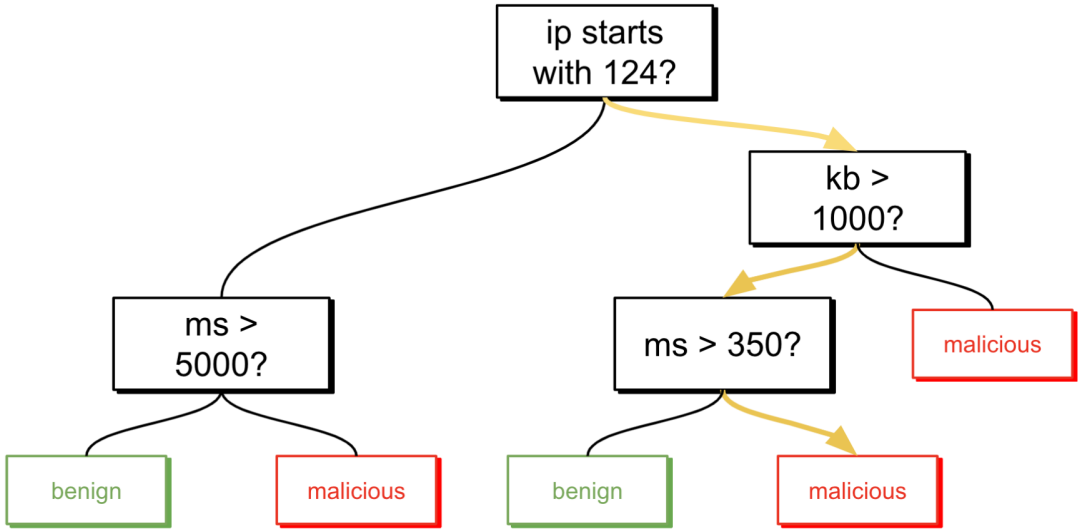

FIGURE 2. A hypothetical binary classification decision tree, used to classify website requests into benign or malicious. Sourced from https://www.elastic.co/blog/benchmarking-binary-classification-results-in-elastic-machine-learning

The input to a decision tree model is a dataset with d dimensions. In the above example, $d = 3$ and the three dimensions are IP, kilobytes, and milliseconds. Each decision node is an inequality on one of the dimensions. We already covered threshold functions, which are inequalities on a single dimension; therefore, we can think of a decision tree as a composition of threshold functions in $\mathbb{R}^d$.

At each decision node of the decision tree, we use the threshold to split $\mathbb{R}^d$ into two cells, one for each child of the decision tree. It follows that the decision tree splits $\mathbb{R}^d$ into k cells where k is the number of leaf nodes. Thus a decision tree can shatter any subset of size k, and so the VC-dimension of a decision tree is k. With the restraint that decision trees cannot be infinite, we have that the VC-dimension is finite and therefore decision trees are PAC-learnable.

Email address: jihongc2@illinois.edu

Email address: parthd2@illinois.edu

MATHEMATICAL ADVANCEMENT THROUGH RESEARCH IDEA EXCHANGE (MATRIX)

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN