CSCI699: Machine Learning Theory Fall 2022

Lecture 1: Introduction to Supervised Learning, PAC learning

Instructor: Vatsal Sharan

Machine learning appears to be one of the most important technologies of our time. There is palpable excitement around ML systems even in the general public, and these systems seem poised to play a central role in numerous areas of society, such as healthcare, transportation, education, commerce, culture and so on.

I would like to argue that all the excitement around ML makes it a pretty exciting time to be investigating the foundations of machine learning:

- There is a lot we don't understand, so there are many very interesting scientific questions! Let us try to argue why the questions that arise is ML are quite interesting.

  Learning appears to be quite a fundamental process (in society, and in nature). It involves the *acquisition of knowledge or intelligence* through the *interaction of data and computation*, all of which seem well worth understanding. As a field, machine learning theory has abundant connections with other rich fields such as computational complexity, algorithms, probability, statistics, information theory, statistical physics, game theory, optimization and so on. Perhaps due to machine learning's growing academic community and practical applications, ML often draws interesting briges between these fields, and offers new perspectives on them.

  Previous technolgies which had wide-scale impact also ushered in studies of new fundamental laws of nature: such as steam engines leading to the laws of thermodynamics, electricity and magnetism leading to Maxwell's equations, or telephone and communications sytems leading to information theory. What new aspects of the world might we uncover by studying the underlying principles behind machine learning?

- Understanding machine learning better could help inform practical algorithms, discover new ones, and can be especially helpful for charting the landscape for new problem settings.

  There are some examples of this from ML's history. A classical one is boosting. Kearns and Valiant [1] asked if you can boost a weak learning algorithm (which gets say 51% accuracy on binary classification) to a strong learner (which gets say 95% accuracy)? The study of weak learning led to boosting and algorithms such as Adaboost, which has been quite a useful framework in practice. There are also examples from optimization algorithms for ML, such as Adagrad and SGD.

  From a more recent vantage point, we see that despite massive progress in applications, there are still numerous challenges we are facing in practice where principles analysis and theorerical frameworks could be valuable. Some such challenges are making our algorithms more computationally efficient, making them more robust, making sure they are fair, that they do not violate privacy of individuals and so on. Theory here can be valuable in terms of providing algorithms, but could perhaps be even more crucial in terms of providing suitable definitions to formalize these requirements. An example of this is differential privacy as a framework to formally think about, analyze and achieve privacy.

The goals for this class are informed by these motivations. The hope is that you can understand the theoretical and conceptual foundations of ML so that:

1. you can appreciate the scientific questions that arise,
2. you can become a better practioner and can think deeply about the challenges you encounter,
3. you are equipped to undertake research into the foundations of machine learning.

Towards these goals, here are some of the questions we will study (with the main focus on the first two questions):

- *How much data is needed for learning?*
- *How much computation time needed for learning?*
- How much memory is need for learning?
- What are tradeoffs between these resources?
- How do neural networks work, and what are the challenges in understanding them?
- How to learn models which are private, fair, robust, ..?

Some of these questions will be investigated in the lectures, the rest will be through your course presentations and project.

1 Supervised learning: An introduction

For most of the course, our focus will be on supervised learning where the objective is to map some inputs to some desired outputs. The 'supervision' comes from a training set, which is a set of labelled input/output pairs. The goal of the learner is to learn some pattern to accurately predict outputs of unseen inputs. Here are some examples:

- Image classification: picture $\rightarrow$ dog/cat.
- Machine translation: text in English $\rightarrow$ text in French.
- SAT solving: SAT instance $\rightarrow$ satisfying assignment if it exists.
- Protein folding: Protein chain $\rightarrow$ 3D structure.

As we can see, supervised learning is quite general as a paradigm and captures a number of interesing problems. Additionally, it is often at the heart of learning systems which are not explicitly doing supervised learning (probably because supervised learning is what we've gotten really good at thanks to recent advances in neural networks):

- In language modelling, we train a supervised learner to predict the next word given previous words.
- In diffusion models, we train a supervised learner to predict the denoised image given a noisy image.
- In game playing systems such as AlphaGo, we train a model to predict the probability of winning given the current board state.

As we can see there are a plethora of interesting ways in which supervised learning can be applied. Though the application domains can vary, to understand the fundamental properies it is helpful to abstract away the details of the problem, and define a general setup. We will focus on binary classification (where there are two possible labels for any datapoint).

- To represent the input, we will embed it as points in d dimensional space. Thus there is an input space $\mathcal{X} \subset \mathbb{R}^d$. Similarly there is an output space $\mathcal{Y}$. Without loss of generality, we can take $\mathcal{Y} = \{\pm 1\}$ for binary classification.
- The goal is to come up with predictor $h(x) : \mathbb{R}^d \to \{\pm 1\}$ which predicts the output of x.

To reason about how well our predictor $h(x)$ is doing, we need to define a **loss function**, which we denote by $\ell(h(x), y)$. The choice of the loss function depends on the domain and the goal of the learning task.

- For a classification problem where the goal is to predict binary outputs, a reasonable choice could be the 0/1 loss: $\ell(h(x), y) = \mathbf{1}(h(x) \neq y)$.
- For a regression problem where the goal is to predict continuous valued attributes, a common choice is the squared loss: $\ell(h(x), y) = (h(x) - y)^2$.

We want to minimize this loss not just for one sample, or just some set of samples given to us, but for samples we might not have even seen yet. We formalize this using a **probablistic perspective**. We say that there is **distribution** P over (x, y) input/output pairs, and we care about minimizing the average loss over samples drawn from this distribution. This brings us to the definition of the **risk** $R(h)$ **of a predictor** on a supervised learning problem:

$$R(h) = \mathbb{E}_{(x,y)\sim P} \left[\ell(h(x), y)\right].$$

$R(h)$ is also sometimes referred to as the **population risk** (since it is the average risk with respect to the entire population versus a subset of samples). Given this definition of risk, it makes sense to define an optimal predictor h^*, which is just the predictor which minimizes the risk over all possible functions from the input to the output:

$$h^* = \arg\min_{h:\mathcal{X}\to\mathcal{Y}} \mathbb{E}_{(x,y)\sim P} \left[\ell(h(x), y)\right].$$

This optimal predictor h^* is also known as the **Bayes optimal predictor.** The risk $R(h^*)$ incurred by h^* is known as the **Bayes optimal risk** for the supervised learning problem.

It would be great if we could learn h^*, but there are two issues. First, it seems to ambitious to try to minimize the risk over the space of all possible functions (we will see this more formally later, but it would be be too statistically or computationally expensive). Second, we don't actually know the distribution D and hence cannot directly minimize the risk.

To deal with the first issue, we usually commit to to a set of functions beforehand. This set of functions $\mathcal{H}$ is known as the **hypothesis class** or the **function class**. For example, this hypothesis class could comprise of all decision trees of a certain depth, all linear classifiers, or all neural networks of a certain size. Here is one example of a hypothesis class, the class of linear functions:

$$\mathcal{H} = \{h : y = \langle w, x \rangle, w \in \mathbb{R}^d\}.$$

We still need to deal with the fact that we do not know D. What we do have is some set of **training samples** from D. When can we hope that minimizing the training risk over the training set S will actually minimize the risk over the distribution D? At a minimum, we would need S to be somewhat represetative of samples drawn from the distribution D. Usually, we work under the **i.i.d. assumption:** we assume that the training set S consists of n samples drawn independently and identically distributed according to D.

Let us call this training set $S = \{(x_i, y_i) : i \in [n]\}$. We can define the **empirical risk** (also known as training error) $\hat{R}_S(f)$ of any predictor f as its average risk over the training set S.

$$\hat{R}_S(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y).$$

Now that we have a function class $\mathcal{F}$ that we have committed to and can measure the empirical risk on the training set S, a natural goal is to find the predictor in $\mathcal{F}$ which minimizes the empirical risk as much as possible. This brings us to the notion of an **empirical risk minimizer (ERM)** $h_{S,\text{ERM}}$ for some function class $\mathcal{H}$, based on a training set $S = \{(x_i, y_i) : i \in [n]\}$,

$$h_{S,\text{ERM}} = \arg\min_{f \in \mathcal{H}} \hat{R}_S(h).$$

Finally, we note that our goal is to find a predictor with small risk $R(h)$, and not just small $\hat{R}(h)$. Since we do not have access to D, we estimate $R(h)$ by measuring the average error on a **test set**. The test samples should be drawn from the same sampling process as the training samples, therefore i.i.d. from the distribution D. This leads to the usual **training/test paradigm** in supervised learning: given a set of samples we randomly shuffle them and partition them into a training set and a test set based on some ratio (such as 80/20). The average error on the test set is a good approximation of the risk on the distribution D as long as the (1) we do not train on the test set, (2) test set is large enough, (3) the test samples are drawn i.i.d. from D (we will see this more formally later). Note that it is still not clear why minimizing the training error should do good things for the test error, but we will investigate this soon.

Comment 1. *A few remarks about the setup so far:*

1. *We are assuming that the datapoints are drawn i.i.d. from the distribution P. This may not always hold in practice. For e.g. there maybe some non-stationarity which causes the data distribution to drift over time. More generally, when ML models are deployed in the wild we*

may not have control over the data distribution they encounter. There has been significant recent interest both from the theory and applied side on training models robust to changes in the data distribution ([2] is a reasonably recent survey).

2. *We are assuming that the datapoints are handed down to us, but in most ML setups significant work can go into collecting the data, filtering and cleaning it, and suitably featurizing it before it is fed to the ML algorithm. This can be crucial for getting good performance.*

2 Error decomposition

Suppose we are working with the function class $\mathcal{F}$, and learn some function h_S based on the training set S. The following is a useful breakdown of the risk of h_S,

$$R(h_S) = \underbrace{\hat{R}_S(h_{S,ERM})}_{\text{representation error}} + \underbrace{\hat{R}_S(h_S) - \hat{R}_S(h_{S,ERM})}_{\text{optimization error}} + \underbrace{R(h_S) - \hat{R}_S(h_{S,ERM})}_{\text{generalization error}}$$

- Representation error (also known as approximation error): We are choosing to work over the function class $\mathcal{H}$, but the optimal predictor h^* might not lie in $\mathcal{H}$. The approximation error measures how much worse is it to work over the function class $\mathcal{H}$.
- Optimization error (also known as computation error): If we are just given the training set S, it seems the ERM is the right function to choose based on S, since it minimizes the risk over the training set. However, in some cases, finding the ERM might be computationally expensive or hard (we will see this later), or we may want to run a simpler/more efficient algorithm. We define the computation error as the difference between the emprical risk of the ERM, and the risk of the predictor h_S.
- Generalization error (also known as estimation error): Note that the ERM was the minimizer of the training error, but our goal is to actually minimize the risk over the true (unknown) distribution D. The minimizer over samples from D need not necessarily be good over the distribution D, for example, with some probability we could get a bad training set for which the ERM could be far from the true minimizer. We define the generalization error as the difference between the risk of the ERM, and the best possible risk achievable within the function class $\mathcal{H}$.

Comment 2. *For models such as neural networks with an extremely large number of parameters, all of these aspects (representation, optimization, generalization) seem to be intricately linked with each other. For e.g. it appears that in many cases the optimization algorithms that we use can influence the generalization behavior of the model.*

2.1 An example: Linear regression

Let us see these concepts for the case of linear regression. We first define the setup.

- $\mathcal{X} = \{x : x \in \mathbb{R}^d, ||x||_2 \leq 1\}$ (unit ball)
- $\mathcal{Y} = \mathbb{R}$

– $\ell(h(x), y) = (1/2)(h(x) - y)^2$

– $\mathcal{H} = \{h_w(x) = \langle w, x \rangle, w \in \mathbb{R}^d\}$

Let us look at the terms from our error decomposition in this context.

- **Representation error**: If $y = \langle w^*, x \rangle$ for some $w^* \in \mathbb{R}^d$, then $\hat{R}_S(h_{S,ERM}) = 0$ and there is no representation error. Otherwise, the true model is not linear and we'll incur an additional penalty because we restrict ourselves to linear functions. In this case we may want to use a richer function class to reduce the representation error.
- **Optimization error**: The empirical risk $\hat{R}_S(w)$ for any predictor w can be written as follows,

$$\hat{R}_S(w) = \frac{1}{2n} \sum_{i=1}^{n} (\langle w, x_i \rangle - y_i)^2$$

  – In this linear regression setting we can bring the optimization error to 0 since we can compute the ERM using a closed-form solution. To find the closed-form solution, we can differentiate the empirical risk above with respect to w and set the gradient to be 0.

$$\begin{aligned} 0 = \frac{\partial \hat{R}_S(w)}{\partial w} &= \frac{1}{n} \sum_{i=1}^{n} x_i (\langle w, x_i \rangle - y_i) \\ \sum x_i y_i &= \sum_{i=1}^{n} x_i x_i^T w \\ w &= \left(\sum x_i x_i^T \right)^{-1} \sum x_i y_i \\ \implies w &= \left(X^T X \right)^{-1} X^T y \end{aligned}$$

  where $X \in \mathbb{R}^{n \times d}$ is the matrix obtained by stacking all the data points x_i together, and $y \in \mathbb{R}^n$ is the vector of outputs y_i for all datapoints x_i.

  An exercise from your algorithms class will tell you that the **running time** needed to compute the ERM solution is $O(nd^2 + d^3)$. Similarly, apart from the memory required to store the input, the additional auxially **memory** need to compute the ERM solution is $O(d^2)$.[1]

  – What if we want to be more computationally efficient than this closed-form solution? One option is to use **gradient descent**. Gradient descent starts with an initialization, and then iteratively updates the estimate based on the gradient with some appropriate step size η:

$$\begin{aligned} w_0 &\leftarrow 0 \\ w_{t+1} &\leftarrow w_t - \eta \nabla \hat{R}_S(w_t). \end{aligned}$$

[1] More technically, we are assuming the input is given to us in read-only memory, and $O(d^2)$ is the additional read/write space that the algorithm needs.

For linear regression, gradient descent updates are as follows.

$$w_{t+1} = w_t - \eta \left(\frac{1}{n} \sum_{i=1}^{n} \underbrace{x_i(\langle w_t, x_i \rangle - y_i)}_{O(d) \text{ time}} \right).$$

What is the time complexity of doing gradient descent?

* The per step time complexity is $O(nd)$. Apart from the memory required to store the input, the additional memory needed to store the current iterate and the gradient is $O(d)$.
* We can show that for this problem, in T steps gradient descent can find a $O(e^{-T})$ sub-optimal solution compared to the closed-form solution (as long as the data-points x_i are 'nice'—more formally they need to be well-conditioned). Hence the optimization error is $O(e^{-T})$ for T steps of gradient descent.

– Therefore, there appears to be a tradefoff here. Gradient descent could be computationally more efficient, but we could have some residual training/optimization error if we are not finding the optimal closed-form solution.

- **Generalization error**: Our analysis so far was to find a solution which has small training error, how many samples do we need to ensure that this solution also has small error on the data generating distribution? If this generalization error is large, there are two remedies (1) get more data, (2) restrict to a smaller function class (*regularization* is one technique to do this).

3 PAC learning (Probably Approximately Correct Learning)

With the basic nomenclature in place, we will now define formal model of learning so that we can investigate when learning is possible. PAC learning [3] provides such a framework. We will mostly focus on binary classification with the zero-one loss:

- Label space: $\mathcal{X} \in \mathbb{R}^d, \mathcal{Y} = \{0, 1\}$
- Loss function: $\ell(h(x), y) = \mathbf{1}(h(x) \neq y)$

There are two crucial points to remember in the PAC learning setup.

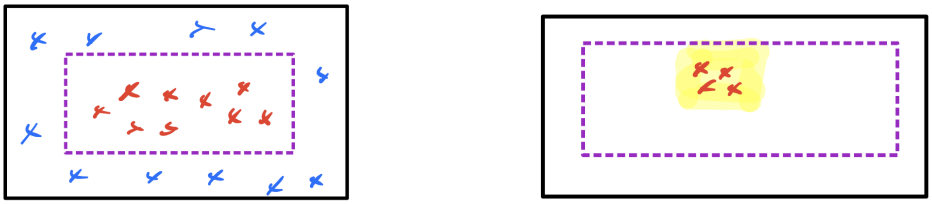
1. Let D be the distribution of the unlabelled datapoints x (in terms of our previous notation, D is just the marginal of x in the joint distribution P of (x, y)). There is no assumption on the marginal distribution D. However, the algorithm is also only required to do well in terms of the loss on datapoints drawn from the same distribution D.
2. We assume that the labels $y = h(x)$ for some $h \in \mathcal{H}$. This assumption is known as the **realizability assumption**, and says that it is possible to exactly predict the labels of every datapoint using the hypothesis class $\mathcal{H}$.

Therefore in terms of the joint distribution P over (x, y) we are placing no assumption on the marginal distribution of x but a strong assumption on the conditional distribution of y given x. We now define PAC learning formally.

Definition 1 (PAC learnability)**.** *A hypothesis class $\mathcal{H}$ is **PAC-learnable** with $m_H(\epsilon, \delta)$ samples if there exists a learning algorithm with the following property: for every $\epsilon, \delta \in (0, 1)$, every distribution D over $\mathcal{X}$ and every $h \in \mathcal{H}$, when the algorithm is given $m_H(\epsilon, \delta)$ samples drawn from D and labeled by h, the algorithm produces a hypothesis $\hat{h}$ such that with probability $1-\delta$, $R(\hat{h}) \leq \epsilon$. (This probability is over randomness in training sets and any internal randomness of the algorithm.)*

Example: Axis-aligned rectangles (also refer to figure in whiteboard notes)

Let us consider a simple example (also see Fig. 1a). Let $\mathcal{X} = [0, 1] \times [0, 1]$, $\mathcal{H}$ = rectangles and D be the uniform distribution on $\mathcal{X}$.



(a) The red points inside the purple, dashed rectangle are labelled as 1, the blue points outside as 0.

(b) D over $\mathcal{X}$ could be such that we cannot recover the true hypothesis (here it is only supported in the yellow region), but prediction is easy.

Figure 1: Axis-aligned rectangles.

- Both the requirements of being only approximately correct and correct with high probability cannot be strengthened without making learning impossible:
  - Failure probability δ: With some small probability we could get bad set of samples (say supported on only one corner of the cube), and cannot do anything.
  - Approximation parameter ϵ: We can only hope to be approximately correct since we cannot hope to recover the true hypothesis h^* exactly given random samples.
- Note that we can also hope to show learnability for any distribution D over x:
  - The reason is that the distribution D could be such that we cannot learn the ground truth h^*, but we can still predict well on that distribution. For example, in the case where the distribution D and the ground truth hypothesis h^* is such that we only see examples with label 1, it may not be possible to learn any approximation to h^* in terms of its parameters, but prediction is trivial in this case (also see Fig. 1b).

4 PAC bound for finite hypothesis classes

Now that we have our definition of learning, we can ask how many samples are required for learning. Our first learnability result shows that finite hypothesis classes are PAC learnable with a sample

complexity only depending logarithmically on the number of hypotheses in the class.

Theorem 2 (PAC bound for finite hypothesis class)**.** *Let $\mathcal{H}$ be a hypothesis class with finite size $|\mathcal{H}|$. Then $\mathcal{H}$ is PAC-learnable with*

$$m_{\mathcal{H}}(\epsilon, \delta) = O\left(\frac{\log(|\mathcal{H}|/\delta)}{\epsilon}\right)$$

samples.

Proof. We will show that an algorithm which finds ERM PAC-learnable $\mathcal{H}$. The empirical risk $R(h)$ and the popularion risk $\hat{R}(h)$ for the zero-one loss are as follows:

$$R(h) = \mathbb{E}_{(x,y)\sim D}\mathbb{1}(h(x) \neq y),$$
$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}(h(x) \neq y).$$

We first note that due to realizability, there is some $h^* \in \mathcal{H}$ such that $R(h^*) = 0$ (Note this also implies that $\hat{R}_S(h^*) = 0$.) We define two sets:

$$\mathcal{H}_{\text{bad}} = \{h \in \mathcal{H} \mid R(h) \geq \epsilon\} \textbf{ (bad hypothesis)}$$
$$S_{\text{bad}} = \{S \in (\mathcal{X} \times \mathcal{Y})^n \mid \exists h \in \mathcal{H}_{\text{bad}}, \hat{R}_S(h) = 0\} \textbf{ (bad training sets)}.$$

Note that $S = S_{\text{bad}} + S_{\text{good}}$, by disjointness. If we can bound the probability of drawing a training set S from S_{bad}, then w.h.p. it must be the case that low empirical risk $\hat{R}$ will correspond to low population risk R.

So our goal is to upper bound probability of getting training set from S_{bad}:

$$S_{\text{bad}} = \bigcup_{h\in\mathcal{H}_{\text{bad}}} \left\{S \in (\mathcal{X} \times \mathcal{Y})^n \mid \hat{R}_S(h) = 0\right\}.$$

We can write,

$$\begin{aligned}
\Pr_S[S \in S_{\text{bad}}] &= \Pr_{S\sim D}\left[\bigcup_{h\in\mathcal{H}_{\text{bad}}} \{\hat{R}_S(h) = 0\}\right] \\
&\leq \sum_{h\in\mathcal{H}_{\text{bad}}} \Pr_{S\sim D}[\hat{R}_S(h) = 0] \textbf{ (union bound)} \\
&= \sum_{h\in\mathcal{H}_{\text{bad}}} \Pr_{S\sim D}\left[\{h(x_i) = h^*(x_i)\ \forall i \in \{1, \ldots, n\}\}\right] \textbf{ (realizability)} \\
&= \sum_{h\in\mathcal{H}_{\text{bad}}} \prod_{i=1}^{n} \Pr_{x_i\sim D}[h(x_i) = h^*(x_i)] \textbf{ (i.i.d.)} \\
&\leq \sum_{h\in\mathcal{H}_{\text{bad}}} \prod_{i=1}^{n} (1-\epsilon) \\
&\leq |\mathcal{H}|(1-\epsilon)^n \leq |\mathcal{H}|e^{-\epsilon n} \text{ (since } 1 - x \leq e^{-x}).
\end{aligned}$$

Assume that the bad event of drawing a training set from S_{bad} happens with bounded probability δ. Then by setting $\delta = |\mathcal{H}|e^{-\epsilon n}$, we can solve for n and say that if $n \geq \left\lceil \frac{\log(|\mathcal{H}|/\delta)}{\epsilon} \right\rceil$, the probability of failure in PAC learning is at most δ, and hence we succeed w.p. $1-\delta$.

■

5 Bias-Complexity Tradeoff

Recall our error decomposition, and assume that the optimization error is zero.

$$R(h_S) = \underbrace{\hat{R}_S(h_{S,ERM})}_{\text{representation error}} + \underbrace{R(h_S) - \hat{R}_S(h_{S,ERM})}_{\text{generalization error}}.$$

A natural question to ask is if we can always make representation error small? Or equivalently, can we learn arbitrarily rich classes?

The following result shows that the answer to the above questions is no.

Theorem 3 (No-free-lunch theorem)**.** *Let A be a learning algorithm for binary classification over $\mathcal{X}$ and let $n \leq |\mathcal{X}|/2$. Then there exists a distribution P over $\mathcal{X} \times \{0,1\}$ such that*

1. *$R(h^*) = 0$.*
2. *With probability at least $1/7$ over training set S of size n, we have $R_P(A(S)) \geq 1/8$.*

Proof. Let the set $\mathcal{C}$ of size $2n$ be given, where $\mathcal{C}$ is a subset of the domain: $\mathcal{C} \in \mathcal{X}^{2n}$.

$$\mathcal{C} = x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_{2n}$$

Next, consider all possible $T = 2^{2n}$ labelling functions which map from $\mathcal{C}$ to $\{0,1\}$, where $|\mathcal{C}| = 2n$.

$$\begin{array}{llllll} f_1 & 0 & 0 & 0 & \cdots & 0 \\ f_2 & 0 & 0 & 0 & \cdots & 1 \\ f_3 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & & & & & \\ f_T & 1 & 1 & 1 & \cdots & 1 \end{array}$$

For each f_i, define a distribution P_i over (x, y) such that the marginal distribution over x is uniform over $\mathcal{C}$ and the label is given by f_i. Denote P_i^n to be the distribution of n i.i.d. samples from P_i.

It is simple to show that $R_{P_i}(f_i) = 0$.

Our goal will be to show that

$$\max_{i \in [T]} E_{S \sim P_i^n} \left[R_{P_i}(A(S)) \right] \geq 1/4. \tag{1}$$

This is sufficient to prove our theorem by the following simple claim.

Lemma 4. (1) *implies that* $\exists\, i \in [T]$ *such that with probability at least* $1/7$ *over training set* $S \sim P_i^n$, *we have* $R_{P_i}(A(S)) \geq 1/8$.

Proof. This follows by applying Markov's inequality to the random variable $1 - R_{P_i}(A(S)$. If

$$\begin{aligned} & E_{S\sim P_i^n}\left[R_{P_i}(A(S))\right] \geq 1/4 \\ \implies & E_{S\sim P_i^n}\left[1 - R_{P_i}(A(S))\right] \leq 3/4 \\ \implies & \Pr\left[1 - R_{P_i}(A(S)) \geq 7/8\right] \leq 6/7 \\ \implies & \Pr\left[R_{P_i}(A(S)) \leq 1/8\right] \leq 6/7. \end{aligned}$$

■

We will now prove (1). Since max is greater than average, we can rewrite (1) as follows.

$$\max_{i\in[T]} \mathbb{E}_{S\sim P_i^n}\left[R_{P_i}(A(S))\right] \geq \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}_{S\sim P_i^n}\left(R_{P_i}(A(S))\right)$$

Let $S_1, \ldots, S_K$ be set of all possible unlabelled training examples ($K = (2n)^n$). Let S_j^i be the labelled training dataset derived by labelling the examples in S_j using the function f_i. Note that in the expectation over S in the previous equation, the only randomnesss comes from the choice of the unlabelled dataset S_j, since the labels are deterministic after fixing f_i. Therefore by swapping the order of the expectation and summation, and then the using the fact that average is greater than minimum we can write,

$$\begin{aligned} \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}_{S\sim P_i^n}\left(R_{P_i}(A(S))\right) &= \mathbb{E}_{S_j} \frac{1}{T}\sum_{i=1}^{T}\left(R_{P_i}(A(S_j^i))\right) \\ &\geq \min_{S_j\in\mathcal{C}^n} \frac{1}{T}\sum_{i=1}^{T}\left(R_{P_i}(A(S_j^i))\right). \end{aligned}$$

Next, fix any S_j of size n. Then there are $p \geq n$ samples $v_1, \ldots, v_p \in \mathcal{C}$ that do not appear in S_j.

$$\begin{aligned} R_{P_i}(h) &= \frac{1}{2n}\sum_{x\in\mathcal{C}} 1(h(x) \neq f_i(x)) \\ &\geq \frac{1}{2p}\sum_{\ell=1}^{p} 1(h(v_\ell) \neq f_i(v_\ell)). \end{aligned}$$

Thus,

$$\begin{aligned} \frac{1}{T}\sum_{i=1}^{T} R_{P_i}(A(S_j^i)) &\geq \frac{1}{T}\sum_{i=1}^{T}\frac{1}{2p}\sum_{\ell=1}^{p} 1\left(A(S_j^i)(v_\ell) \neq f_i(v_\ell)\right) \\ &\geq \frac{1}{2}\min_{r\in[p]} \frac{1}{T}\sum_{i=1}^{T} 1\left(A(S_j^i)(v_\ell) \neq f_i(v_\ell)\right). \end{aligned}$$

Partition f_i into $T/2$ pairs, where each pair $(f_i, f_{i'})$ agrees on everything except v_ℓ. Every pair $(f_i, f_{i'})$ produces same labelled datasets $S_j^i, S_j^{i'}$. Therefore

$$1\left(A(S_j^i)(v_\ell) \neq f_i(v_\ell))\right) + 1\left(A(S_j^i)(v_\ell) \neq f_{i'}(v_\ell)\right) = 1.$$

Therefore, we can write

$$\frac{1}{T}\sum_{i=1}^{T} R_{P_i}(A(S_j^i)) \geq \frac{1}{2}\cdot\frac{1}{T}\cdot\frac{T}{2} = \frac{1}{4}$$

which completes the proof. ■

As a corollary, we have that the space of all possible functions on an infinite domain is not PAC-learnable with a finite number of samples.

Corollary 5. *Let $\mathcal{X}$ be an infinite domain set (such as $\mathbb{R}^d$) and let $\mathcal{H}$ be all possible functions from $\mathcal{X} \to \{0, 1\}$. Then, $\mathcal{H}$ is not PAC-learnable.*

6 Further reading

You can read sections 2.1-2.3, 3.1-3.2 and 5.1, 5.2, 5.3 of [4]. The introduction of Chapter 3 of [5] has a nice introduction to supervised ML. You can also look at Les Valiant's original paper if you're interested [3], which also nicely summarizes a lot of our motivation for studying ML. These short videos of him talking about this paper and learning more broadly result for his Turing award citation are also fun viewing: https://amturing.acm.org/award_winners/valiant_2612174.cfm (the first two videos are relevant to the class).

References

[1] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[2] Zheyan Shen, Jiashuo Liu, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*, 2021.

[3] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[4] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[5] Moritz Hardt and Benjamin Recht. Patterns, predictions, and actions: A story about machine learning. *arXiv preprint arXiv:2102.05242*, 2021.