Lecture 11: Computational Learning Theory

PAC Learning
Computational Learning Theory

- So far our questions were of the form — Given $m$ samples iid from a distribution $\mathbb{P}_x$, can we upper bound the error as $\epsilon = O(m^{-\alpha})$?
Computational Learning Theory

• So far our questions were of the form — *Given* $m$ samples iid from a distribution $\mathbb{P}_x$, can we upper bound the error as $\epsilon = O(m^{-\alpha})$?

• In particular, we have discussed how to use properties of the distribution to give better statistical “rates”.
Computational Learning Theory

- So far our questions were of the form — Given $m$ samples iid from a distribution $\mathbb{P}_x$, can we upper bound the error as $\epsilon = O(m^{-\alpha})$?

- In particular, we have discussed how to use properties of the distribution to give better statistical “rates”.

- [Today] In Computational Learning Theory, we will treat the learning algorithm as a computational process. This requires making a few things explicit (Discuss in pairs what they can be):
Computational Learning Theory

• So far our questions were of the form — **Given** $m$ samples iid from a distribution $P_x$, can we upper bound the error as $\epsilon = O(m^{-\alpha})$?

• In particular, we have discussed how to use properties of the distribution to give better statistical “rates”.

• [Today] In Computational Learning Theory, we will treat the learning algorithm as a computational process. This requires making a few things explicit (Discuss in pairs what they can be):
  
  • How does the algorithm “interact” with the distribution?
Computational Learning Theory

• So far our questions were of the form — Given $m$ samples iid from a distribution $\mathbb{P}_x$, can we upper bound the error as $\epsilon = O(m^{-\alpha})$?

• In particular, we have discussed how to use properties of the distribution to give better statistical “rates”.

• [Today] In Computational Learning Theory, we will treat the learning algorithm as a computational process. This requires making a few things explicit (Discuss in pairs what they can be):
  
  • How does the algorithm “interact” with the distribution?
  
  • What information does the algorithm have about the “learning problem”? 
Computational Learning Theory

• So far our questions were of the form — Given $m$ samples iid from a distribution $\mathbb{P}_x$, can we upper bound the error as $\epsilon = O(m^{-\alpha})$?

• In particular, we have discussed how to use properties of the distribution to give better statistical “rates”.

• [Today] In Computational Learning Theory, we will treat the learning algorithm as a computational process. This requires making a few things explicit (Discuss in pairs what they can be):
  • How does the algorithm “interact” with the distribution ?
  • What information does the algorithm have about the “learning problem” ?
  • How “long” is the algorithm allowed to run etc ?

Finally, this proposes an answer to the question what can be “learned” under various restrictions.
Computational Learning Theory

- So far our questions were of the form — **Given** $m$ samples iid from a distribution $\mathbb{P}_x$, can we upper bound the error as $\epsilon = O(m^{-\alpha})$?

- In particular, we have discussed how to use properties of the distribution to give better statistical “rates”.

- **[Today]** In Computational Learning Theory, we will treat the learning algorithm as a computational process. This requires making a few things explicit (Discuss in pairs what they can be):
  - How does the algorithm “interact” with the distribution?
  - What information does the algorithm have about the “learning problem”?
  - How “long” is the algorithm allowed to run etc?
  - How will the output estimator be “represented”?
Computational Learning Theory

• So far our questions were of the form — *Given* $m$ samples iid from a distribution $\mathbb{P}_x$, *can we upper bound the error as* $\epsilon = O(m^{-\alpha})$?

• In particular, we have discussed how to use properties of the distribution to give better statistical “rates”.

• [Today] In Computational Learning Theory, we will treat the learning algorithm as a computational process. This requires making a few things explicit (Discuss in pairs what they can be):
  
  • How does the algorithm “interact” with the distribution?
  
  • What information does the algorithm have about the “learning problem”?
  
  • How “long” is the algorithm allowed to run etc?
  
  • How will the output estimator be “represented”?

Finally, this proposes an answer to the question what can be “learned” under various restrictions.
Binary classification Setting

Some terminologies

• Instance space: $\mathcal{X}$ e.g. $\mathbb{R}^2, \{0, 1\}^d, \mathbb{R}^d$ etc.

• Label space: $\mathcal{Y} = +1, -1$

• Hypothesis/Concept classes are represented by : $C, \mathcal{H}, \mathcal{F}$. They are sets of maps from $\mathcal{X}$ to $\mathcal{Y}$. (In other words, classes of labelling functions) E.g.
  - CONJUNCTIONS e.g. $x_1 \land x_3 \land x_5$
  - DISJUNCTIONS e.g. $x_2 \lor x_3 \lor x_5$
  - Linear halfspaces e.g. $\sum_{i=1}^{d} w_ix_i \geq b$

• Data Distribution $\mathbb{P}_x$ over $\mathcal{X}$

• Example Oracle: An oracle $\text{Ex}(c; \mathbb{P}_x)$ that samples $x \sim \mathbb{P}_x$ and returns $(x, c(x))$.

• Target Concept Refer to $c$ as the “target concept” (ground truth).
Learning Algorithm

- **Learning algorithm** An algorithm $\mathcal{A}$
  - for learning concept class $\mathcal{C}$
  - with hypothesis class $\mathcal{H}$
  - can call the example oracle $\text{Ex}(c; P_X)$ many times
  - and must return some $h \in \mathcal{H}$.

- **Two sources of randomisation**:
  - **Randomness from data** Inherently, due to the randomisation of $\text{Ex}(c; P_X)$, $\mathcal{A}$ is always randomised. This randomness is from $P_X$.
  - **Randomness from algorithm** After receiving data from $\text{Ex}(c; P_X)$, $\mathcal{A}$ can flip an unbiased coin and introduce further randomness into the algorithm. Let the joint distribution over $P_X$ and internal coin flips of $\mathcal{A}$ be $P$. 

A concept class $\mathcal{C}$ is PAC learnable with hypothesis class $\mathcal{H}$ if there exists a learning algorithm $\mathcal{A}$ such that for all distributions $P_x$, concept $c \in \mathcal{C}$, and $\epsilon, \delta > 0$, if $\mathcal{A}$ is given access to $Ex(c; P_x)$ and knows $\epsilon, \delta$, $\mathcal{A}$ returns $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, over inner randomisation of $Ex(c; P_x)$ and $\mathcal{A}$ we have that $P_x[h(x) \neq c(x)] \leq \epsilon$. Further, the number of calls made to $Ex(c; P_x)$ must be polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}$.

- If $\mathcal{A}$ runs in time $\text{poly}\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)$ then $\mathcal{C}$ is **efficiently PAC learnable**.

- If $\mathcal{C}$ is learnable with $\mathcal{H} = \mathcal{C}$, then we say $\mathcal{C}$ is **proper learnable**. Otherwise, it is referred to as **improper learnable**. We will focus on proper PAC learnability for now.

- Number of times $\mathcal{A}$ calls $Ex(c; P_x)$ is equal to the sample size $m$. So far, we have written $\epsilon$ as function of $m$ i.e. $\epsilon(m, \delta)$ is the statistical error rate.
Understanding the definition

What are some things or questions that stand out to you about learnability in this definition?

- **Efficiency**
  - What is one unit of time?
  - What are possible reasons of inefficiency?
  - What kinds of computational constraints are required on $h$?

- **Available information to $A$**
  - What does $A$ know and what does $A$ not know?
  - What are some possible changes to $\text{Ex} \left( c; \mathbb{P}_x \right)$ that can simulate real environments? How can they change a class’ learnability?

Discuss in pairs
Understanding the definition

• Efficiency.
  • What is one unit of time?
    Call to $\text{Ex}(c; \mathbb{P}_x)$ takes unit time. The algorithm is run on a turing machine.
  • What are possible reasons of inefficiency?
    Exponential sample complexity or exponential running time.
  • What kinds of computational constraints are required on $h$?
    $h$ needs to be poly evaluable, otherwise trivial

• Available information
  • What does $\mathcal{A}$ know and what does $\mathcal{A}$ not know?
    Knows $C$ but not $c$. Does not know $\mathbb{P}_x$.
  • What are some changes to $\text{Ex}(c; \mathbb{P}_x)$ that can simulate real environments?
    Noisy Oracle (RCN, Massart, Tsybakov), Positive/Negative only, Membership Query, Statistical Query

• It attempts to separate the two things
  • Having sufficient data
  • Being able to compute the estimator/hypothesis from the data
Learning Axis-Aligned Rectangles

- Let $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{+1, 0\}$
- $\mathcal{C}$ is the class of Axis-Aligned Rectangle Classifiers. A concept $c \in \mathcal{C}$ labels $x \in \mathcal{X}$ positive (+1) if $x$ lies inside the rectangle and 0 o.w.

**Theorem**

The concept class of axis aligned rectangles is efficiently proper PAC learnable.

**Proof:**

- Algorithm $A$ chooses $m = \frac{4}{\epsilon} \log \left( \frac{4}{\delta} \right)$, queries $\mathbb{E} \mathbb{X} (c; \mathbb{P}_x)$ $m$ times and outputs the smallest axis-aligned rectangle $R'$ that contains all +ve points.
- Let $R$ be the target rectangle. Choose 4 regions $T_1, T_2, T_3, T_4$ along the inner sides of $R$ such that each region has mass $\frac{\epsilon}{4}$ under $\mathbb{P}_x$. Note that if $\mathbb{E} \mathbb{X} (c; \mathbb{P}_x)$ returns at least one point in all of these regions with probability greater than $1 - \delta$, it suffices for us.
- Let $A_i$ be the event that $\mathbb{E} \mathbb{X} (c; \mathbb{P}_x)$ upon $m$ calls does not return any point in $T_i$. Show $\mathbb{P} [\bigcup_i A_i] \leq 4 \exp \left( -\frac{me}{4} \right)$
- Setting $m = \frac{4}{\epsilon} \log \left( \frac{4}{\delta} \right)$ completes the proof.
Issues: Previous definition does not account for the size of the concept class or the instance space.

- **Representation scheme for concept class**: \( \rho : (\Sigma \cup \mathbb{R})^* \rightarrow C \) is a representation scheme for \( C \). e.g. \( \rho((x_1, y_1), (x_2, y_2)) = \) axis-aligned rectangle with bottom left corner at \((x_1, y_1)\) and top right corner in \((x_2, y_2)\). (Unit cost to represent alphabets in \( \Sigma \) and numbers in \( \mathbb{R} \))

- **Size of representations**: The function \( \text{size} : (\Sigma \cup \mathbb{R})^* \rightarrow \mathbb{N} \) measures the size of a representation in \( (\Sigma \cup \mathbb{R})^* \).

- **Size of concept**: A size of a concept is the minimum size over all representations in that representation scheme
  \[
  \text{size}(c) = \min_{\sigma : \rho(\sigma) = c} \text{size}(\sigma)
  \]

What are some examples where the choice of \( \rho \) affects the size of a concept?

- **Instance size**: Instances \( x \in \mathcal{X} \) also has an associated size e.g. memory to store. We denote \( \mathcal{X}_d \) as an instance space where all \( x \in \mathcal{X}_d \) has size \( d \).

Often these are clear from context but sometimes need further thought.
For $d \geq 1$, let $C_d$ be a concept class over $X_d$. Consider instance space $X = \bigcup_{d=1}^{\infty} X_d$ and the corresponding concept class $C = \bigcup_{d=1}^{\infty} C_d$.

**Definition (PAC learning)**

A concept class $C$ is PAC learnable with hypothesis class $\mathcal{H}$ if there exists a learning algorithm $A$ such that for all $d > 0$, all distributions $P_x$ over $X_d$, concept $c \in C_d$, and $\epsilon, \delta > 0$, if $A$ is given access to $\text{Ex}(c; P_x)$ and knows $\epsilon, \delta, \text{size}(c)$, and $d$, $A$ returns $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, over inner randomisation of $\text{Ex}(c; P_x)$ and $A$ we have that $P_x[h(x) \neq c(x)] \leq \epsilon$. Further, the number of calls made to $\text{Ex}(c; P_x)$ should be polynomial in $\text{size}(c), d, \frac{1}{\epsilon}, \frac{1}{\delta}$.

**Efficient PAC learnability:** $A$ should run in time polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}, \text{size}(c)$, and $d$. Usually $\text{size}(c)$ is bounded by some polynomial in $d$ and hence can be ignored.
Learning CONJUNCTIONS

Now we will see an example of PAC Learning Attempt II

- Let $\mathcal{X}_d = \{0, 1\}^d$, $\mathcal{Y} = \{0, 1\}$
- CONJUNCTIONS$_d$ over $d$ boolean variables $z_1, \ldots, z_d$
  - literal is a variable or its negation
  - conjunction is an AND of literals.
- A conjunction can be represented with two sets $P, N \subseteq [d]$
  $$c_{P,N} = \bigwedge_{i \in P} z_i \land \bigwedge_{j \in N} \overline{z}_j$$
- The class of CONJUNCTIONS$_d$ is the set of all conjunctions.
  $$\text{CONJUNCTIONS}_d = \{ c_{P,N} | P, N \subseteq [d] \}$$
- Note an efficient representation scheme: size $c_{P,N} \leq d$

Theorem (learning conjunctions)

The concept class $\mathcal{C} = \bigcup_{d \geq 1} \text{CONJUNCTIONS}_d$ is efficiently PAC learnable.
Proof of learnability of CONJUNCTIONS

Let $c^*$ be the target concept.

Proof First, we state the algorithm and then prove the guarantees

Algorithm Fix $m \geq \frac{2d}{\varepsilon} \log \left( \frac{2d}{\delta} \right)$ and run the following algorithm. Start with $P, N = [d], [d]$;

• For $i = 1 \ldots m$
  • Call $\text{Ex} \left( c^*; \mathcal{D} \right)$ and let $(x, y)$ be the output.
  • If $y = +1$, eliminate all literals from $P, N$ that cause $c_{P,N}(x) = 0$.
    • i.e. $P = P \setminus \{ j : x_j = 0 \}$, $N = N \setminus \{ j : x_j = 1 \}$
  • Denote the resultant conjunction as $h = c_{P,N}$. Return $h$.

Convince yourself that

• the returned conjunction is the largest conjunction that is accurate on the $m$ observed data samples.
• All eliminated literals are also not present in $c^*$. 
Proof of learnability of CONJUNCTIONS (Continued)

Approximately Correct  For a literal $\ell$ and an instance $x \in \{0, 1\}^d$, let $\ell(x)$ denote the assignment of the literal $\ell$ on the instance $x$. i.e. if $\ell = z_i$ then $\ell(x) = x_i$. If $\ell = \neg z_i$, then $\ell(x) = 1 - x_i$.

- A literal $\ell$ is “bad” if $\mathbb{P}_x[c^*(x) = 1 \land \ell(x) = 0] \geq \frac{\epsilon}{2d}$.
- Note by construction, $\mathbb{P}_x[h(x) \neq c^*(x)] = \mathbb{P}_x[h(x) = 0 \land c(x) = 1]$.
- Let $B$ be the set of bad literals and $h$ contain no literals in $B$. Then, $\mathbb{P}_x[h(x) = 0 \land c(x) = 1] \leq \sum_{\ell \in \bar{B}} \mathbb{P}_x[h(x) = 0 \land \ell(x) = 1] \leq \epsilon$

Probably Correct  Now, we need to prove that $h$ contains no bad literals. Let $A_\ell$ be the event that $\ell$ is not eliminated by the algorithm after $m$ calls

- Bound $\mathbb{P}[A_\ell] \leq (1 - \frac{\epsilon}{2d})^m \leq \exp(-\frac{em}{2d})$.
- $\mathbb{P}$ [at least 1 “bad” literal remain] $\leq \mathbb{P} \left[ \bigcup_{\ell \in B} A_\ell \right] \leq \sum_{\ell=0}^{2d} \exp(-\frac{em}{2d})$
- Use $m \geq \frac{2d}{\epsilon} \log \left( \frac{2d}{\delta} \right)$ to show that all bad literals are eliminated with probability $1 - \delta$. 