CSCI699: Theory of Machine Learning Fall 2021

Lecture 8: Rademacher Complexity & Computational Complexity of Learning

Instructor: Vatsal Sharan *Scriber: Emir Ceyani*

Today

- Finish Rademacher Complexity
- Introduce a new topic: "Computational Complexity of Learning"

Recap (Rademacher Complexity)

Theorem 1 (Excess risk bounds using Rademacher)**.** *Assume that $\forall z$ and $\forall h \in H$, we have $|\ell(h,z)| \leq C$. Then, the probability at last $1-\delta$ over $S \sim \mathcal{D}^n$,*

1. $\sup_{h\in\mathcal{H}}(R(h) - \hat{R}_s(h)) \leq 2\,\mathbb{E}_{s'}\,\mathcal{R}(\ell \circ \mathcal{H} \circ S^{'}) + c\sqrt{\frac{2\log(\frac{1}{\delta})}{n}}$
2. $\sup_{h\in\mathcal{H}}(R(h) - \hat{R}_s(h)) \leq 2\mathcal{R}(\ell \circ \mathcal{H} \circ \mathcal{S}) + 3c\sqrt{\frac{2\log(\frac{2}{\delta})}{n}}$
3. *For* $h^* = \arg\min_{h\in\mathcal{H}} R(h)$

$$R(ERM_{\mathcal{H}}(S)) - R(h^*) \leq 2\mathcal{R}(\ell \circ \mathcal{H} \circ \mathcal{S}) + 4c\sqrt{\frac{2\log(\frac{4}{\delta})}{n}}$$

Lemma 2 (Contraction Lemma)**.** *For each $i \in [n]$, let $\phi_i : \mathbb{R} \to \mathbb{R}$ be a $\rho-$Lipschitz function, i.e. $|\phi_i(x) - \phi_i(y)| \leq \rho|x-y|, \quad \forall x, y \in \mathbb{R}$. $\forall a \in \mathbb{R}^n$, define $\phi(a) \in \mathbb{R}^n$ as $\phi(a) = [\phi_1(a_1), \ldots, \phi_n(a_n)]$. For a set A, let $\phi \circ A = \{\phi(a) : a \in A\}$. then, $\mathcal{R}(\phi \circ A) \leq \rho\mathcal{R}(A)$.*

Rademacher Complexity of Linear Classes

Lemma 3. *Let $S = (x_1, \ldots, x_n)$ and $\mathcal{H}_2 = \{h_w(x) =< w, x >: ||w||_2 \leq B_2\}$. Define the operation $\mathcal{H}_2 \circ S = \{< w, x_1 >, \ldots, < w, x_n >: ||w||_2 \leq B_2\}$. Then, the Rademacher complexity of an ℓ_2-bounded linear model is $\mathcal{R} \leq B_2\frac{\max_i ||x_i||_2}{\sqrt{n}}$.*

Lemma 4. *Let $S = (x_1, \ldots, x_n)$ and $\mathcal{H}_1 = \{h_w(x) =< w, x >: ||w||_1 \leq B_1\}$. Define the operation $\mathcal{H}_1 \circ S = \{< w, x_1 >, \ldots, < w, x_n >: ||w||_2 \leq B_2\}$. Then, the Rademacher complexity of an ℓ_1-bounded linear model is, $\mathcal{R} \leq B_1 \max_i ||x_i||_\infty \sqrt{\frac{2\log(2d)}{n}}$.*

1 Takeaways from Rademacher Complexity

1. If the loss function $\ell(h, z)$ is 1-Lipschitz (e.g. hinge loss or absolute value loss), then by the contraction lemma,
$$\mathcal{R}(\ell \circ \mathcal{H}_2 \circ S) \leq \mathcal{R}(\mathcal{H} \circ S)$$
Now using excess risk bound, it is possible to get generalization bounds directly.

2. Note that the VC-dimension bound for linear predictors in $\mathbb{R}^d$ is $O(d)$. The Rademacher bound **does not directly depend polynomially on dimension!** Hence, it can be much smaller than VC dimension.

3. Motivating idea of regularization:

   - Notice $\mathcal{H}_2$ bound depends on $B_2 \max_i ||x_i||_2$. So, if we want to learn over a small ℓ_2-norm ball, we would have better generalization. In addition, if the best possible weight has also bounded norm bounded by B_2, we also get a small approximation error. Thus, there is a need to choose the best possible B_2
   - Also, $\mathcal{H}_1$ bound depends on $B_1 \max_i ||x_i||_\infty$.
     Say $x_i \in \{\pm 1\}^d \implies \max_i ||x_i||_\infty = 1$ & $\max_i ||x_i||_2 = \sqrt{d}$. Also, $w \in \{-1, 0, 1\}^d$ and is also r-sparse. Then, $||w||_2 = \sqrt{r} \quad \& \quad ||w||_1 = r$.
     Thus , $B_1 \max_2 ||x_i||_\infty = R \quad \& \quad B_2 \max_i ||x_i||_2 = \sqrt{Rd}$. If $R << d$, working over ℓ_1 could be much better than working over ℓ_2 ball.

Now, we finish our studies over the *statistical complexity* of learning and move into *computational complexity* of learning.

2 Computational Complexity of Learning

1. What about the *running time of the algorithm*?
2. What can we learn *in polynomial time*?

Before answering these questions, let us recall PAC leanability:

Definition 5 (PAC learnability)**.** *A hypothesis class $\mathcal{H}$ is **PAC-learnable** if there exists a learning algorithm with the following property:*

$\forall \epsilon, \delta \in (0, 1), \forall \mathcal{D} \sim \mathcal{X}$ and $\forall h \in \mathcal{H}$, when the algorithm is given $n_{\mathcal{H}}(\epsilon, \delta)$ samples drawn from $\mathcal{D}$ and labelled by h, the algorithm produces a hypothesis $\hat{h}$ such that with probability $1 - \delta$, $R(\hat{h}) \leq \epsilon$.

Notice that the probability is over randomness in training set, and any internal algorithmic randomness.

Next, we need to answer the question: what does it mean to *learn in polynomial time*? Another question is **with respect to what parameter do we define** *polynomial*?

Question: Polynomial in size of training set?

This is a bad idea because there may be cases that algorithm receives much more (or much less) data than it needs to learn, but it runs in polynomial time with respect to the size of original data. Just using this notion does not take account to how much of the data we use.

To account the cost for training data, let us think that there is an entity (which is a black-box function) which outputs insightful advises to learner. Here, it's the datapoints. Formally speaking, we define *an example oracle* as follows:

Definition 6 (Example Oracle)**.** *For any distribution D over $\mathcal{X}$ & hypothesis $h(x) : \mathcal{X} \to \mathcal{Y}$, we define* $\mathrm{Ex}(\mathcal{L}, \mathcal{D})$ *as an* ***example oracle*** *executing the following steps in order:*

- *Draws* $x \sim \mathcal{D}$
- *Labels* $y = h(x)$
- *Outputs* $(x, h(x))$

With this definition in mind, we can define *sample complexity* as **the number of example oracle calls**! Each oracle call costs unit time to the learner.

Our notion of polynomial time is polynomial in the instance size. We will typically think of the instance space as being $\mathcal{X}^d$. where $\mathcal{X}$ is $\{\pm\}^d$ or $\mathbb{R}^d$. Therefore, we want the algorithm to run in time polynomial in the feature dimension d.

However, in reality, we want to also allow the runtime to depend polynomially on the *in representation size* of the hypothesis class.

Question: What is the representation size of hypothesis class?

Answer: **The number of bits required to write down any hypothesis in hypothesis class.** For example, in a feed-forward neural network with instance size d,

$$\text{representation size} = \#\text{edges} \times \#\text{bits required to store each weight}$$

However, for all hypothesis classes we consider in class,

$$\text{representation size} = \text{poly}(\text{instance size of datapoints})$$

Therefore for all problems that we consider,

$$\text{poly}(\text{representation size}, \text{instance size}) = \text{poly}(\text{instance size})$$

Therefore, we only consider **polynomial in instance size**.

Definition 7 (Efficient PAC Learning)**.** *A hypothesis class $\mathcal{H}$ is PAC-learnable if there exists a learning algorithm A with the following property:*

$\forall \epsilon, \delta \in (0,1), \forall \mathcal{D} \sim \mathcal{X}^d$ (where $\mathcal{X}$ is typically $0, 1 \in \mathcal{R}$) and $\forall h^* \in \mathcal{H}$, if A is given access to example oracle $\mathrm{Ex}(h^*, \mathcal{D})$, with probability $1-\delta$, it outputs a hypothesis $h \in \mathcal{H}$ with $R(\hat{h}) \leq \epsilon$.

$\mathcal{H}$ is efficiently PAC learnable if running time of A is ***polynomial*** *in $d, \frac{1}{\epsilon}, \frac{1}{\delta}$.*

Next, our focus will be on learning Boolean functions, with input domain $\mathcal{X}^d = \{0,1\}^d$.

Conjunctions

The class of all conjunctions on d Boolean literals $(x_1, x_2, \ldots x_d)$. As an example,

$$x_1 \wedge \bar{x}_3 \wedge x_4$$

represents the hypothesis class which is 1 if and only if $x_1 = 1, x_3 = 0,$ and $x_4 = 1$.

Question: Can we design a polynomial $(d, \frac{1}{\epsilon}, \frac{1}{\delta})$ time algorithm for learnable conjuctions?

Theorem 8. *The class of conjunctions on Boolean literals is efficiently PAC learnable.*

Algorithm 1 Algorithm to learn Boolean Conjunctions

Proof.
1: Set $h = x_1 \vee \bar{x}_1 \vee x_2 \vee \bar{x}_2 \ldots x_d \vee \bar{x}_d$
2: **for** $i = 1$ to n do **do**
3: $(a_i, y_i) \leftarrow \mathrm{Ex}(h^*, D)$
4: **if** $t_i = 1$ **then**
5: Drop each $\bar{x}_j$ from h if $(a_i)_y = 1$
6: Drop each x_j from h if $(a_i)_y = 0$
7: **end if**
8: **end for**

Claim: The algorithm given above is an ERM over the class of conjunctions.

We need to show that the algorithm gets 0 misclassification error over training examples $(a_1, y_1), \ldots, (a_n, y_n)$

Claim: The set of literals appearing in h at any time, contains the set of literals in target hypothesis h^*.

Proof. In the beginning, h contains all possible conjunctions. We remove a literal from h if it was get to 0 in an example. Such a literal cannot appear in h^*. □

As the set of literals appearing in h at any time, contains the set of literals in target hypothesis h^*: $h(a) = 1 \implies h^*(a) = 1, \forall a \in \{0,1\}^d$. h correctly classifies all training data labelled as 0.

Moreover, after getting any new data point a_i with $y_i = 1$, h updates to predict 1 on the datapoint and it will never be updated predict 0 on ai, as literals are only removed. Thus Algorithm 2 is an ERM.

Now, we use ERM result for finite hypothesis classes as $|\mathcal{H}| \leq 2^{2d}$. The result for learnability of finite hypothesis classes implies that we can learn with error ϵ with failure probability δ with $O(\frac{d\log(\frac{1}{\delta})}{\epsilon})$ samples.

□

Intractability of learning 3-Term Disjunctive Normal Forms(DNFs)

$$\text{3-Term-DNF}_\text{d} = \{T_1 \vee T_2 \vee t_3 | T_i \text{is a conjunction on } \{x_1, \ldots x_d\}\}$$

Theorem 9. *3-Term-DNF formulae are not efficiently PAC-learnable unless RP=NP.*

Before proving the theorem, let us recap the fundamental concepts in computational complexity.

2.1 Computational Complexity Review

Definition 10 (NP)**.** *A decision problem C is in NP if there exists a polynomial-time algorithm A such that for every instance x of C,*

- *If x evaluates to "yes", then $\exists y$, $|y| \leq \text{poly}(|x|), A(x,y) = 1$*
- *If x evaluates to "no", then $\forall y$, $|y| \leq \text{poly}(|x|), A(x,y) = 0$*

Intuition A: verifier , y: certificate / witness

Consider 3SAT

$$\begin{aligned}\text{3SAT}_\text{d} =&(x_1 \vee \bar{x}_2 \vee x_5) \wedge \\ &(x_5 \vee x_6 \vee \bar{x}_7) \wedge \\ &\vdots \\ &(x_{d-2} \vee \bar{x}_{d-1} \vee x_d)\end{aligned}$$

Certificate y: Satisfying assignment. it is easy to check if an assignment is valid, There always exists a certificate if instance is satisfiable, no certificate if not.

Definition 11 (RP)**.** *A decision problem C is in RP if there exists a randomized polynomial-time algorithm A such that for every instance x of C,*

- *If x evaluates to "yes", A outputs "yes" w.p. $\geq \frac{2}{3}$*
- *If x evaluates to "no", A outputs "no" w.p. 1.*

Intuition C is in RP if there exists a polynomial-time algorithm with one-sided error. 3SAT instance is satisfiable, output TRUE w.p. $\geq \frac{2}{3}$. otherwise, always output FALSE.

It is widely believed that RP $\neq$ NP.

Definition 12 (NP-Completeness)**.** *A decision problem C is in NP-Complete if,*

- *C is in NP*
- *Every decision problem C' in nP can be reduced to C in polynomial time.*