TTIC 31250: An Introduction to the Theory of Machine Learning

Avrim Blum May 2, 2018

---

1 Computational Hardness of Learning

Today we will talk about some computational hardness results for learning. Generally speaking, there are two different types of such results. The first focus on the concept class (or representation) used by the learning algorithm. For example, given a labeled sample S, how hard is it to find a consistent $h \in C$ whenever one exists? Or, given a labeled sample S, how hard is it to find the $h \in C$ of minimum empirical error (especially if there is no $h \in C$ whose empirical error is 0)? The second type of result focuses on the intrinsic difficulty of prediction. For example, given access to data from some distribution D labeled by some target $c^* \in C$, how hard is it to find a prediction rule h of error at most, say, $1/4$ when we do not require that $h \in C$ and only require that it be some polynomial-time procedure?

The first type of hardness is often called "representation-dependent" hardness, or hardness of "proper learning". The second type is often called "representation-independent" hardness, or hardness of "improper learning".

2 Representation-dependent hardness

We've seen that the consistency problem can be efficiently solved for linear threshold functions (using linear programming) and for disjunctions (using a simple list-and-cross-off algorithm). How about for the intersection of 2 halfspaces, i.e., the AND of two linear threshold functions, or for a 2-clause CNF, i.e., the AND of two disjunctions? These consistency problems turn out to be NP-hard.

2.1 Hardness for learning an intersection of two halfspaces

Let's consider the intersection of two halfspaces. We will prove that solving the consistency problem is NP-hard by performing a reduction from the NP-hard problem "hypergraph 2-coloring". First of all, what is the hypergraph 2-coloring problem? Here you are given a hypergraph, which is just a set V of n nodes (call them $1, 2, \ldots, n$) and m subsets $s_1, s_2, \ldots, s_m \subseteq V$ which are sometimes called "hyperedges". Your goal is to give each node a color, Red or Blue, such that no subset s_j is monochromatic. That is, every S_j has at least one Red node and at least one Blue node. This problem is NP-hard.

To perform our reduction, we want to create a set S of positive and negative examples in R^n (in fact, they will be in $\{0,1\}^n$) such that the set S is consistent with an intersection of two halfspaces (i.e., an AND of two linear threshold functions) if and only if the given instance of hypergraph 2-coloring has a solution. We can do this as follows.

1. First, we label the origin as positive.

2. Next, for each coordinate i, we label the unit vector e_i (having a 1 in coordinate i and a 0 in all other coordinates) as negative.

3. Finally, for each set s_j we label the indicator-vector for that set (having a 1 in coordinate i for each $i \in s_j$ and a 0 in the rest) as positive.

Claim 1 *The labeled examples produced by this reduction are consistent with an intersection of two halfspaces (i.e., an AND of two linear threshold functions) if and only if the given instance of hypergraph 2-coloring has a solution.*

Proof: First, suppose the labeled examples are consistent with the AND of two linear threshold functions. For each negative example: if threshold 1 says negative, color its associated node Red; if threshold 2 says negative, color its associated node Blue. (For points on the negative side of both, their associated nodes can be colored anything). We claim this is a legal solution to the given hypergraph 2-coloring instance. The reason is that suppose some set s_j had all its nodes the same color, say Red. Let's look at the Red LTF: $w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq w_0$, where $w_0 \geq 0$ since the origin is positive. For each $i \in s_j$, we know $w_i > w_0$ since e_i is negative. But this means that $\sum_{i \in s_j} w_i > |s_j|w_0$, and this in turn is $\geq w_0$ since $w_0 \geq 0$. So, this means the indicator vector for s_j would be mistakenly labeled as negative, a contradiction.

In the other direction, suppose you are given a solution to the hypergraph 2-coloring instance. Then just create the "Red" LTF $w_1x_1 + \ldots w_nx_n \leq 1/2$ where $w_i = 1$ if i is Red and $w_i = -n$ if i is Blue, and create the "Blue" LTF $w'_1x_1 + \ldots w'_nx_n \leq 1/2$ where $w'_i = 1$ if i is Blue and $w'_i = -n$ if i is Red. Each negative example e_i is correctly separated from the origin by one of the two LTFs (Red points separated by the Red LTF and Blue points by the Blue LTF). But each of the positives will be correctly classified since the large $-n$ value will counteract any number of $+1$ values of the other color, and each set has at least one element of each color. ■

2.2 Hardness for learning an AND of two OR-functions (2-clause CNF)

We saw we could easily learn a single OR function, like $x_1 \vee x_3 \vee x_5$. How about an AND of two OR functions, like $(x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_4 \vee x_6)$? This turns out to be NP-hard. These are called "2-clause CNF formulas".

To make things easier to think about, let's define our class of interest C to be the class of *monotone* 2-clause CNFs; i.e., no variables are negated.

Claim 2 *The consistency problem for monotone 2-clause CNF is NP-complete.*

Proof: We can do a reduction from hypergraph 2-coloring just like before. For each $i = 1, 2, \ldots, n$ create the negative example e_i (with $x_i = 1$ and the rest of the features set to 0). Notice that this implies that no variable can be in both clauses. Next, for each set s_j we label the indicator vector for s_j as positive.

Let's think about what this means. Suppose we have a consistent 2-clause CNF. Create a coloring where variables in clause 1 are the red nodes, variables in clause 2 are the blue nodes, and nodes whose variables are not in either clause can be either color (recall that no variable is in both clauses).

This will be a legal coloring since the fact that the indicator vector for s_j is positive means that s_j must have at least one feature set to 1 inside each clause (to make their AND be positive). In the other direction, given a legal coloring, we just create one clause for each. This will correctly output negative on the negatives and correctly output positive on the positives. ■

What about the non-monotone case? Actually, the above reduction still works (assuming $n \geq 3$). In particular, if there exists a consistent 2-clause CNF at all, it will have to be monotone. The reason is that if some $\bar{x}_i$ is in clause 1, then clause 2 can't have any $\bar{x}_j$ (else e_k will be positive for $k \notin \{i, j\}$) and clause 2 can't have any x_j for $j \neq i$ (else e_j would be positive). So the only option for clause 2 is to equal x_i. But ANDing clause 1 with x_i is the same as just removing $\bar{x}_i$ from clause 1.

2.3 Learning 2-clause CNF using 2-DNF

Interestingly, you *can* learn 2-clause CNF formulas using the class of 2-DNF formulas (an OR of ANDs where each AND is of size at most 2). The reason is that (a) any 2-clause CNF can be written as a 2-term DNF by just distributing out the AND, and (b) you can learn 2-DNF in polynomial time and $O(n^2/\epsilon)$ examples by reduction to learning a single OR function.

Interestingly, no similar trick is known for an intersection of 2 halfspaces.

2.4 Hardness of finding the LTF of minimum empirical error

We know that we can find a single LTF consistent with the data when one exists. However, when there is no consistent LTF, the problem of finding the LTF of minimum empirical error is NP-hard.

We can show this using a very similar reduction, based on the Maximum Independent Set problem in graphs. Given a graph $G = (V, E)$, an independent set is a set of nodes having no edges between them. Finding the maximum independent set in a graph is NP-hard.

We can do a reduction by using the same idea as before. For sake of intuition, let's flip signs this time. Specifically, the origin will be negative, each coordinate unit vector e_i will be positive, and then for each edge (i, j) in the graph, we will make the indicator vector for that edge negative. Notice that the maximum independent set in G is exactly the largest set of positive examples that can be separated from all the negatives. In other words, imagine you are not allowed to make mistakes on any negative examples and you goal is to get as many positive examples correct as possible. Specifically, if S is an independent set, then the associated LTF is

$$w_1 x_1 + \ldots + w_n x_n \geq 1/2$$

where we set $w_i = 1$ if $i \in S$ and $w_i = -1$ if $i \notin S$. Notice that this guarantees that for each edge (i, j), the example x corresponding to that edge has $w \cdot x \leq 0$. In the other direction, if we have an LTF $w_1 x_1 + \ldots + w_n x_n \geq w_0$, where $w_0 > 0$, and we let $S = \{i : w_i \geq w_0\}$, then this has to be an independent set, because if $w_i \geq w_0$ and $w_j \geq w_0$ then $w_i + w_j \geq 2w_0 \geq w_0$, so there can't be a negative example in the indicator vector for that pair.

Now, what about minimizing empirical error, i.e., you are allowed to make mistakes of both kinds? To address this, we just replicate each negative example enough times (more than the total number positive examples) so that even making a mistake on a single negative example would be suboptimal.

3 Representation-independent hardness

We now consider hardness results that do not depend on the form of the hypotheses used by the learning algorithm. Specifically, our goal is to say that the class C contains functions that are so "complicated" that for some distributions D, no polynomial-time algorithm can predict much better than just random guessing.

3.1 Cryptographic assumptions

A classic cryptographic result is that if factoring is hard, then one can create functions f called pseudorandom generators (PRGs) with the following properties. First of all f is a function from $\{0,1\}^n$ to $\{0,1\}^m$ where $m > n$. An algorithm A is said to "break" f if

$$|\Pr_{v\sim\{0,1\}^m}[A(v)=1] - \Pr_{x\sim\{0,1\}^n}[A(f(x))=1]| \geq 1/poly(n).$$

In other words, A can (at least slightly) distinguish a random m-bit string from the result of running f on a random n-bit string. Now the claim is that given an n-bit number N, for any $m = poly(n)$, one can create such a function f such that if f can be broken by a polynomial-time algorithm A, then this gives a polynomial-time algorithm for factoring N. (Formally, there is a general efficient procedure for creating such functions f_N from any given N).

It turns out one can use such constructions to show that learning DFAs (finite automata) is hard, under the assumption that factoring is hard.[1] Specifically, if you could learn an n-state DFA to error less than $1/4$ (say), with at most n^k samples and polynomial time, then you could get a polynomial-time factoring algorithm. Here is the idea. Let $m = 10n^k$, and suppose we are given a string of m bits $b_1 b_2 \cdots b_m$ and need to decide if it's truly random or is the output of a PRG f on a random input of size n. What we'll do is create a probability distribution that is uniform over $m = 10n^k$ examples $x_1, x_2, \ldots, x_m$ (chosen in a specifc way that I won't get into), where the label of x_i is b_i. If $b_1 b_2 \cdots b_m$ is truly random, then there's no way an algorithm from only $n^k = m/10$ samples will be able to get to error rate $1/4$. The best it could hope to do is have error rate $\frac{9}{10}\frac{1}{2} = \frac{9}{20} > \frac{1}{4}$. However, if $b_1 b_2 \cdots b_m$ is the result of running a PRG from some appropriate construction,[2] the claim is there exists an n-state DFA that can compute the b_i from the x_i and so our algorithm has to get error at most $1/4$. So our algorithm can break the PRG.

To make this all work, one has to show there are PRGs that are "simple" enough that the ith bit can be computed from the x_i using a DFA. This was proven by Kearns and Valiant.

3.2 Modern results using hardness of refuting random CSPs

Recently there has been a development of strong representation-independent hardness results for classes we encounter much more commonly in machine learning, based on the presumed difficulty of refuting certain kinds of random constraint satisfaction problems (CSPs). Here is a brief summary of a result of [2] on hardness for agnostic learning of linear separators.

[1] You can view a DFA as a Boolean function: given an input string x, we feed it into the DFA and output "positive" if the DFA accepts the string and "negative" if the DFA rejects the string.

[2] Technically, this will be a PRG on an input of size n' such that $n = poly(n')$, but that's fine since we still have $m = poly(n')$.

First, the claim is that under an appropriate hardness assumption (described below), for any constant c, there is no polynomial-time algorithm that given a sample S of n^c points in $\{-1,1\}^n$, can in general whp distinguish (a) the case that labels are just uniform random coin flips from (b) the case that there exists a linear separator of error at most 10%. (You can replace "10%" with any constant greater than 0). So, this is a pretty strong hardness claim. E.g., if you could even weak-learn under condition (b), this would allow you to distinguish it from case (a).

The assumption used to achieve this result is that "refuting random k-XOR formulas" is hard. A k-XOR formula is an AND of XORs, where XORs are allowed to be negated too. For instance, a 2-XOR might look like:

$$(x_1 \oplus x_3) \wedge (x_2 \oplus x_3) \wedge \neg(x_1 \oplus x_3) \wedge \ldots.$$

It is easy to find a satisfying assignment to such a formula if one exists, using Gaussian elimination. However, if there is no satisfying assignment, finding the one that satisfies as many of the XORs as possible is NP-hard. What the assumption states is that for $m \leq n^{\sqrt{k}\log k}$, it is in fact hard to distinguish the case (a) of a *random* k-XOR formula of m constraints from (b) a formula of m k-XOR constraints that has an assignment satisfying at least 90% of the constraints.

A few notes about this:

1. A random formula of $m \geq n/\epsilon^2$ k-XOR constraints will with high probability have the property that no assignment satisfies more than a $\frac{1}{2} + \epsilon$ fraction of the constraints. We can see this as follows. Fix an assignment x and *then* draw the m constraints at random (each selected by picking k variables at random, along with randomly choosing to negate the XOR or not). Each constraint is satisfied by x with probability $1/2$, independently, so by Hoeffding bounds, the probability that the specific assignment x satisfies more than a $\frac{1}{2} + \epsilon$ fraction of the constraints is at most $e^{-2m\epsilon^2}$. By the union bound, the chance that *any* assignment satisfies more than this many is at most $2^n e^{-2m\epsilon^2} \leq 2^n e^{-2n} < e^{-n} = o(1)$.

2. There were earlier results based on the assumption that this task was hard for other kind of constraint satisfaction problems, some of which were subsequently broken (i.e., shown not to be hard) by [1].

3. For the k-XOR problem, the best known polynomial-time algorithm requires $m = \Omega(n^{k/2})$ to solve the distinguishing problem.

References

[1] Sarah R. Allen, Ryan O'Donnell, and David Witmer. How to refute a random CSP. In *FOCS*, 2015.

[2] Amit Daniely. Complexity theoretic limitations on learning halfspaces. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 105–117, 2016.