Machine Learning Theory (CSC 482A/581B) - Lectures 3 and 4

Nishant Mehta

1 Towards Statistical Learning: the Consistency Model

We now begin our study of the first major component of this course: *statistical learning theory.* Our starting point will be classification in the statistical learning setting (recall the protocol introduced in Lecture 1) under the realizability assumption.

Let's review the statistical learning protocol in the realizable case. Let $\mathcal{C} \subset 2^{\mathcal{X}}$ be a fixed and known concept class.

1. Nature selects a distribution P over the input space $\mathcal{X}$ and selects a concept $c \in \mathcal{C}$.
2. Learner is given a training set of n labeled examples $(x_1, y_1), \ldots, (x_n, y_n)$, where each x_j is drawn i.i.d. from P and each $y_j = c(x_j)$.
3. Learner wishes to select a hypothesis f in some hypothesis space $\mathcal{F}$ which obtains low risk with respect to 0-1 loss, i.e., for which the misclassification probability $\Pr_{X \sim P}(f(X) \neq c(X))$ is small.

A natural approach for Learner is to take $\mathcal{F} = \mathcal{C}$ and select any hypothesis $f \in \mathcal{F}$ which is consistent with the training sample.

Definition 1. A hypothesis $f : \mathcal{X} \to \mathcal{Y}$ is *consistent* with training sample $(x_1, y_1), \ldots, (x_n, y_n)$ if $f(x_j) = y_j$ for all $j \in [n]$.

If we are lucky and have a training sample that is representative of the actual distribution, then it will be the case that the *empirical risk*

$$\frac{1}{n} \sum_{j=1}^{n} \mathbf{1}[f(x_j) \neq c(x_j)]$$

is close to the true risk

$$R(f) = \mathsf{E}_{X \sim P}[\mathbf{1}[f(X) \neq c(X)]] = \Pr_{X \sim P}(f(X) \neq c(X)).$$

In this lucky event, minimizing the empirical risk is a good proxy for minimizing the actual risk.

For now, we will consider the algorithmic question of how to find a hypothesis consistent with the training sample. Once that is settled, at least in a few scenarios, we will directly address the question of how much data we need before the empirical risk is "close enough" to the true risk; answering this question is critical if we wish to justify algorithms that attempt to minimize the empirical risk.

Definition 2. An algorithm $\mathcal{A}$ learns a class $\mathcal{C}$ in the *consistency model* if, for any training set $(x_1, y_1), \ldots, (x_n, y_n)$, the algorithm outputs a concept $f \in \mathcal{C}$ consistent with the training set if one exists and otherwise outputs `False` (indicating that no such concept exists in $\mathcal{C}$).

As before, our primary interest is in efficient algorithms, whose runtime is polynomial in the number of examples n and the size of an example (typically the dimension of the data, d). Note that unlike the mistake bound model, our algorithm must also handle the situation where the examples were not labeled according to any concept $c \in \mathcal{C}$.

2 Examples of learning in the consistency model

2.1 Monotone conjunctions

When $\mathcal{C}$ is the class of monotone conjunctions over $\{0,1\}^d$, we can reuse the key idea for learning monotone conjunctions in the mistake bound model:

> Start with the conjunction of all variables $x_1 \wedge x_2 \wedge \ldots \wedge x_d$, and kick out any feature which takes the value of 0 in some positive example. If this conjunction is consistent with the negative examples, output the conjunction; otherwise, output `False`.

Analysis. The resulting hypothesis, call it $\hat{f}$, only kicks out features when necessary to be consistent with the positive examples, and thus subject to this constraint, it tries to label as many examples as negative as possible. Therefore, if there is a monotone conjunction consistent with the data, then $\hat{f}$ can never make a mistake on a negative example and hence is consistent with the data. If $\hat{f}$ does make a mistake on a negative example, then no monotone conjunction is consistent with the data, and so the algorithm should indeed output `False`.

Efficiency. The above algorithm is efficient; its runtime is $O(dn)$, since it only needs to inspect each feature of each example at most once.

Extensions. The class of monotone disjunctions, concepts of the form $x_{j_1} \vee x_{j_2} \vee \ldots \vee x_{j_k}$ for $k \in \{0, 1, \ldots, d\}$, can be learned in an identical way if we first transform the data as follows:

- Flip the sign of each label.
- For each example x, replace each feature x_j by its negation $\bar{x}_j = 1 - x_j$.

Learning monotone disjunctions over the original data is now equivalent to learning monotone conjunctions on the transformed data.

We leave as a simple exercise the task of efficiently learning (not necessarily monotone) conjunctions over $\{0,1\}^d$ in the consistency model: a conjunction is of the form (e.g.) $\bar{x}_2 \wedge x_4 \wedge x_7$. Another good exercise is to design an algorithm for efficiently learning k-CNF, the class of formulas in *conjunctive normal form* for which each clause has at most k terms. For instance, $x_1 \wedge (\bar{x_1} \vee x_2) \wedge (x_3 \vee \bar{x}_4)$ is an example of a concept in 2-CNF. A related exercise is the task of learning k-DNF, the class of formulas in *disjunctive normal form* for which each clause has at most k terms. For instance, $\bar{x}_1 \vee (x_1 \wedge \bar{x}_2) \vee (\bar{x}_3 \wedge x_4)$ is an example of a concept in 2-DNF.

2.2 Linear separators

Suppose that we are learning homogeneous linear separators in the realizable case, where there exists a hypothesis w^* obtaining margin $\gamma > 0$ over the training sample $(x_1, y_1), \ldots, (x_n, y_n)$. We later will see a simple way to find a w consistent with the data using the Perceptron algorithm. However, rather than using Perceptron, it is possible to try and find such a w directly.

Without loss of generality, assume that w^* has unit norm. Since w^* obtains margin γ over the training sample, it holds that

$$y_j \langle w^*, x_j \rangle \geq \gamma \text{ for all } j \in [n].$$

Therefore, taking $\tilde{w} = \frac{w^*}{\gamma}$, we have

$$y_j \langle \tilde{w}, x_j \rangle \geq 1 \text{ for all } j \in [n].$$

Clearly, this vector $\tilde{w}$ induces a linear separator consistent with the data, and so we need only find some $w \in \mathbb{R}^d$ which satisfies the linear constraints

$$y_j \langle w, x_j \rangle \geq 1 \text{ for all } j \in [n].$$

We can find such a vector using linear programming. A linear program is an optimization problem which has a linear objective and linear constraints. In our case, we do not have an objective, and so we simply set the parameter for the objective to be equal the zero vector $\mathbf{0}$. Framed as a linear program, our task becomes

$$\begin{aligned} \underset{w \in \mathbb{R}^d}{\text{minimize}} \quad & \langle \mathbf{0}, w \rangle \\ \text{subject to} \quad & \langle w, y_j \, x_j \rangle \geq 1, \; j = 1, \ldots, n. \end{aligned}$$

There is a rich theory for solving linear programs, which includes efficient (polynomial time) algorithms. Thus, it is possible to learn linear separators efficiently in the consistency model.

3 The PAC Model

In statistical learning, the goal is to minimize the expected loss with respect to a probability distribution P, but rather than having knowledge of P, Learner instead only has random samples drawn from P. Consequently, whenever Learner outputs a hypothesis, it cannot with complete certainty guarantee that the hypothesis obtains zero risk, or even risk bounded by some level $\varepsilon < 1$. In many situations, what Learner *can* guarantee is that with some confidence, say probability $1 - \delta$ with respect to the sample of data, the hypothesis has risk at most ε.

The PAC model of learning formalizes this style of guarantee. The acronym PAC stands for Probably Approximately Correct; informally, an algorithm learns in the PAC model if it *Probably* (with confidence $1 - \delta$) outputs a hypothesis which is *Approximately Correct* (has risk at most ε).

Before providing the formal definition of the PAC model, let's see an example of a PAC-style (ε, δ)-style guarantee.

3.1 A first PAC-style guarantee

Consider the lowly problem of learning monotone conjunctions. Suppose that we run our algorithm for learning monotone conjunctions in the consistency model, thereby obtaining a conjunction $\hat{f}$ consistent with the training sample $(x_1, y_1), \ldots, (x_n, y_n)$. We then wonder how large the true risk of $\hat{f}$ might be. The following guarantee holds for *any* finite class $\mathcal{C}$, which includes the class of monotone conjunctions as a special case.

Theorem 1. *Let $\mathcal{C}$ be a finite concept class, let P be a fixed distribution over $\mathcal{X}$, and consider a learning algorithm $\mathcal{A}$ that, given a training sample, outputs a hypothesis $\hat{f} \in \mathcal{C}$. For any $\varepsilon > 0$ and any $\delta \in (0, 1)$, if*

$$n \geq \frac{\log |\mathcal{C}| + \log \frac{1}{\delta}}{\varepsilon},$$

then, with probability at least $1-\delta$ over the draw of $X_1, \ldots, X_n$ from P, it holds that if $\hat{f}$ is consistent with the training sample $(X_1, c(X_1)), \ldots, (X_n, c(X_n))$, then $\hat{f}$ has risk at most ε.

Before giving the proof, note that applying the above result in the case of monotone conjunctions immediately yields the following corollary, since for this class we have $|\mathcal{C}| = 2^d$.

Corollary 1. *Let $\mathcal{C}$ be the class of monotone conjunctions over $\{0, 1\}^d$, let P be a fixed distribution over $\mathcal{X}$, and consider a learning algorithm $\mathcal{A}$ that, given a training sample, outputs a hypothesis $\hat{f} \in \mathcal{C}$. For any $\varepsilon > 0$ and any $\delta \in (0, 1)$, if*

$$n \geq \frac{d \log 2 + \log \frac{1}{\delta}}{\varepsilon},$$

then, with probability at least $1-\delta$ over the draw of $X_1, \ldots, X_n$ from P, it holds that if $\hat{f}$ is consistent with the training sample $(X_1, c(X_1)), \ldots, (X_n, c(X_n))$, then $\hat{f}$ has risk at most ε.

Proof (of Theorem 1). The idea is to first bound the probability that a *fixed* hypothesis $f \in \mathcal{C}$ is consistent with the data but has risk greater than ε. This probability is equivalent to the probability of drawing zero successes in n independent Bernoulli trials when the success probability exceeds ε; this latter probability is of course strictly less than $(1 - \varepsilon)^n$.

Next, we bound the probability that $\hat{f}$ is consistent with the training sample but has risk greater than ε. In the below, the probability is taken with respect to $(X_1, \ldots, X_n)$.

$$\begin{aligned}
&\Pr\Big(\Big(R(\hat{f}) > \varepsilon\Big) \textbf{ and } \Big(\hat{f}(X_j) = c(X_j) \text{ for all } j \in [n]\Big)\Big) \\
&\leq \Pr\Big(\exists f \in \mathcal{C} \text{ such that } (R(f) > \varepsilon) \textbf{ and } (f(X_j) = c(X_j) \text{ for all } j \in [n])\Big) \\
&\leq \sum_{f \in \mathcal{C} \text{ with } R(f) > \varepsilon} \Pr(f(X_j) = c(X_j) \text{ for all } j \in [n]).
\end{aligned}$$

The first inequality holds because $\hat{f}$ takes values in $\mathcal{C}$, and the second inequality is the union bound. Now, the last line above can be coarsely be upper bounded by $|\mathcal{C}|(1 - \varepsilon)^n$.

Thus, the probability of the "bad event" that $\hat{f}$ is consistent with the training sample, yet has risk exceeding ε, is at most $|\mathcal{C}|(1 - \varepsilon)^n$. Using the inequality $1 - x \leq e^{-x}$, this probability is at most $|\mathcal{C}|e^{-n\varepsilon}$. It remains to set $\delta = |\mathcal{C}|e^{-n\varepsilon}$ and solve for n, yielding $n = \frac{\log |\mathcal{C}| + \log \frac{1}{\delta}}{\varepsilon}$. □

Here is an intuitive way to think about the above result: suppose that there is a bad hypothesis with risk greater than ε. What is the probability that this hypothesis gets zero error on the training data and thus tricks us into thinking it is a good hypothesis. The proof first bounds this probability of being tricked. Now, if there are many bad hypotheses (perhaps as many as $|\mathcal{C}| - 1$ of them in the worst case), we try to control the probability that *any* of them trick us. A coarse way to do this is to just via a union bound, which scales with the number of hypotheses. Thus, the larger $|\mathcal{C}|$ is, the more data we need to avoid being tricked, as otherwise we might overfit.

3.2 PAC Learning

Definition 3. We say that $\mathcal{C}$ is *PAC learnable* if there exists an algorithm $\mathcal{A}$ which, for any concept $c \in \mathcal{C}$, for every distribution P over $\mathcal{X}$, and for all $\varepsilon > 0$ and $\delta \in (0, 1)$, satisfies the following guarantee: if $\mathcal{A}$ is given access to examples drawn from P and labeled according to c, then with probability at least $1 - \delta$, $\mathcal{A}$ outputs a hypothesis $\hat{f} \in \mathcal{C}$ with risk $\Pr(\hat{f}(X) \neq c(X)) \leq \varepsilon$.

We say that $\mathcal{C}$ is *efficiently PAC learnable* if, in addition,

- $\mathcal{A}$ uses a number of examples polynomial in d, $\frac{1}{\varepsilon}$, and $\frac{1}{\delta}$;
- $\mathcal{A}$ runs in time polynomial in d, $\frac{1}{\varepsilon}$, and $\frac{1}{\delta}$.

In the above definition of efficient PAC learnability, d refers to the dimension of $\mathcal{X}$. If $\mathcal{C}$ is (efficiently) PAC learnable using algorithm $\mathcal{A}$, then we say that $\mathcal{A}$ *(efficiently) PAC learns* $\mathcal{C}$.

Remarks:

- Computational learning theorists also demand efficient PAC learnability, so that the algorithm runs in polynomial time.
- The original definition of efficient PAC learnability, due to Leslie Valiant, also requires that the number of examples and runtime be polynomial in a quantity $|c|$ known as the *representation size* of c; this quantity may be thought of as the number of bits used to represent concept c in some representation/encoding of $\mathcal{C}$. As this is more crucial in computational learning theory rather than statistical learning theory, we will not discuss this point further and have not referred to $|c|$ in our definition of efficient PAC learnability. From a statistical perspective, $|c|$ will not matter. For those interested, this point is covered in detail in the computational learning theory book of Kearns and Vazirani (1994).

In a last crucial remark, we introduce the fundamental notion of sample complexity.

Definition 4. The *sample complexity* $n_{\mathcal{C}} \colon (0, 1)^2 \to \mathbb{N}$ of learning a concept class $\mathcal{C}$ is the function which, given any $\varepsilon \in (0, 1)$ and any $\delta \in (0, 1)$, specifies the minimum number of examples $n_{\mathcal{C}}(\varepsilon, \delta)$ needed to PAC learn $\mathcal{C}$ to risk ε with confidence $1 - \delta$.

The sample complexity of learning $\mathcal{C}$ is a purely statistical notion: it does not take into account the runtime of the algorithm. In statistical learning theory, our primary interest is providing upper and lower bounds on the sample complexity. As we will see when we study the VC dimension, a class $\mathcal{C}$ is PAC learnable *only if* its sample complexity is polynomial in $\frac{1}{\varepsilon}$ and $\log \frac{1}{\delta}$.

4 Connections between learning models

We have now seen three models of learning:

- the mistake bound model;
- the consistency model;
- the PAC model.

There are strong connections between these models. We will explore four connections. The first relates the mistake bound model to the consistency model.

Theorem 2. *If $\mathcal{C}$ is learnable in the mistake bound model, then $\mathcal{C}$ is learnable in the consistency model.*

Proof. It will be useful to assume that the learning algorithm is a *conservative learner*, meaning that the algorithm changes its current hypothesis only when it makes a mistake. This assumption is not problematic, because given any algorithm $\mathcal{A}$ which makes at most M mistakes on any sequence of examples, we can convert this algorithm into a conservative learner $\mathcal{A}'$ which has the same mistake bound, as follows:

> Algorithm $\mathcal{A}'$ behaves exactly as $\mathcal{A}$ except when the latter updates its current hypothesis after predicting correctly, in which case $\mathcal{A}'$ does not update its current hypothesis.

To analyze the number of mistakes made by $\mathcal{A}'$, observe that the behavior of this algorithm (in terms of updates and mistakes) is the same as the behavior of $\mathcal{A}$ were it run only on the subsequence examples on which it makes mistakes. Since $\mathcal{A}$ makes at most M mistakes on any sequence, it holds that $\mathcal{A}'$ satisfies this same mistake bound.

Now, let $\mathcal{A}'$ be a conservative algorithm that learns $\mathcal{C}$ in the mistake bound model. Then, for any sequence of examples labeled according to some concept $c \in \mathcal{C}$, $\mathcal{A}'$ will make at most finitely many mistakes.

Next, consider a training set $(x_1, y_1), \ldots, (x_n, y_n)$ labeled according to some $c \in \mathcal{C}$. The following algorithm obtains a hypothesis consistent on the training set:

1. Run $\mathcal{A}'$ on the sequence of examples in the training set.
2. If $\mathcal{A}'$ made at least one mistake in this run, then go back to 1. Otherwise, output the current hypothesis of $\mathcal{A}'$

The above algorithm eventually terminates because $\mathcal{A}'$ only makes a finite number of mistakes on any sequence, which includes as a special case the sequence formed by multiple concatenations of the training set with itself. When the algorithm terminates, no mistakes were made over the training set, and so the current hypothesis is consistent with the training set. □

The next connection is one that we essentially already proved via Theorem 1.

Theorem 3. *If $\mathcal{C}$ is finite and learnable in the consistency model, then $\mathcal{C}$ is PAC learnable.*

Proof. If $\mathcal{A}$ learns $\mathcal{C}$ in the consistency model, then applying Theorem 1 with this choice of $\mathcal{A}$ yields the desired PAC-style guarantee. □

The above two theorems immediately imply the following corollary.

Corollary 2. *If $\mathcal{C}$ is finite and learnable in the mistake bound model, then $\mathcal{C}$ is PAC learnable.*

Requiring $\mathcal{C}$ to be finite is a serious restriction, and in fact, this restriction can be dispensed with entirely.

Theorem 4. *If $\mathcal{C}$ is learnable in the mistake bound model, then $\mathcal{C}$ is PAC learnable.*

Proof. Let $\mathcal{A}$ be an algorithm that learns $\mathcal{C}$ in the mistake bound model, and suppose that it makes at most M mistakes on any sequence of examples. Without loss of generality, we will restrict our analysis to conservative learners; let $\mathcal{A}'$ be the conservative learner corresponding to $\mathcal{A}$ (recall the construction from the proof of of Theorem 2). Therefore, $\mathcal{A}'$ also makes at most M mistakes on any sequence of examples, and it updates its current hypothesis only when it makes a mistake.

Now, let's consider a few possibilities.

- Suppose that the first hypothesis f_0 used by $\mathcal{A}'$ (the one prior to making any mistakes) correctly classifies the first $\tilde{n} = \frac{\log \frac{1}{\delta'}}{\varepsilon}$ examples, for some $\delta' \in (0,1)$. If the risk of f_0 is greater than ε, this can only happen with probability at most $(1-\varepsilon)^{\tilde{n}}$. Thus, with probability at least $1-\delta'$, it holds that f_0 has risk at most ε, and we are done.

- If instead, f_0 made a mistake on one of the first $\tilde{n}$ examples, then we make the same argument with f_1 and another $\tilde{n}$ examples.

- This argument eventually terminates, since $\mathcal{A}'$ makes at most M mistakes: after $M\tilde{n}$ examples we are guaranteed to have a hypothesis which correctly classifies at least $\tilde{n}$ examples.

The above argument motivates the following PAC learning algorithm $\mathcal{A}_{\text{PAC}}$:

1. Draw a training set from P (and labeled according to $c \in \mathcal{C}$) of size $n = \frac{M \log \frac{M}{\delta}}{\varepsilon}$.

2. Feed the sequence of examples in the training set into $\mathcal{A}'$ and stop when either of the following hold:

   - the current hypothesis correctly classifies $\frac{\log \frac{M}{\delta}}{\varepsilon}$ examples;
   - $\mathcal{A}'$ has made M mistakes.

3. Output the algorithm's current hypothesis.

From the mistake bound, the algorithm produces a sequence of hypotheses $h_1, h_2, \ldots$ of length at most $M+1$. If the algorithm stops on a hypothesis h_m for $m \leq M$, then h_m was consistent with $\frac{\log \frac{M}{\delta}}{\varepsilon}$ examples; this can only happen with probability $\frac{\delta}{M}$ if in fact $R(h_m) > \varepsilon$. Therefore, the probability that one of the first M hypotheses is returned but actually has risk exceeding ε is at most δ. If h_{M+1} is returned, the algorithm will never make a mistake again, and so h_{M+1} certainly (with probability 1) cannot have risk exceeding ε

So, the above algorithm satisfies the following guarantee:

> With probability at least $1-\delta$, the hypothesis output by $\mathcal{A}_{\text{PAC}}$ when run on the training set has risk at most ε.

□

5 Relaxing the PAC model

Improper learning The PAC model of learning is a nice first model, but it can be unnecessarily restrictive if our ultimate goal is prediction rather than identifying the correct concept $c \in \mathcal{C}$; the latter is sufficient, but hardly necessary. In the PAC model as defined above, we took the hypothesis space $\mathcal{F}$ equal to the concept class $\mathcal{C}$; this version of PAC learning is called *proper* PAC learning. We can and will instead also consider *improper* PAC learners; these learning algorithms use a hypothesis space $\mathcal{F}$ which strictly contains $\mathcal{C}$ and hence can be much richer.

Is there any advantage to using improper learners? The answer is yes, at least under the widely believed assumption that[1] $RP \neq NP$. Under this assumption, the concept class $\mathcal{C}$ of 3-term DNF formulas is not PAC learnable by any proper learner, but it is PAC learnable by an improper one. Here, the input space is $\{0,1\}^d$, and the concept class consists of disjunctions of three terms,

[1] RP (randomized polynomial time) is a complexity class.

with each term being a conjunction of an arbitrary number of literals. For example, the following formula is an instance of this class:

$$(x_1 \wedge \bar{x}_2 \wedge \bar{x}_3 \wedge x_4) \vee (\bar{x}_1 \wedge x_3 \wedge \bar{x}_5 \wedge x_6 \wedge \bar{x}_7) \vee (x_2 \wedge x_7).$$

The hardness result involves showing that a graph is 3-colorable if and only if a certain training set is consistent with some 3-term DNF formula. Thus, properly PAC learning the class of 3-term DNFs is as hard as the graph 3-coloring decision problem.

On the other hand, there is a very simple improper PAC learner for this class. Observe that from the distributive law, every 3-term DNF formula can be expressed as a concept in 3-CNF. Thus, any proper PAC learner for 3-CNF can be used to PAC learn 3-term DNF.

Further relaxations. In the next lecture, we will begin our study of *agnostic learning* by relaxing two very strong assumptions inherent to the PAC model:

- that our hypothesis space contains c;
- that there even exists a perfect hypothesis.

References

Michael J Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory.* MIT press, 1994.