CIS 419/519: Applied Machine Learning
Lecture 5: Explaining ML Generalization

Professor: Dan Roth

1 Computational Learning Theory

In this lecture note, we will be discussing generalization guarantees that try to quantify the success of a learning algorithm on a given learning task. We seek a theory that will relate the probability of successful learning, the number of training examples, the complexity of the hypothesis space, the accuracy to which the target concept is approximated, and how training examples are presented. Such a theory should be able to answer the following fundamental questions:

- What learning problems can be solved?
- When can we trust the output of a learning algorithm?
- What general laws constrain inductive learning?

In the previous lecture, we focused on quantifying the performance of a learning algorithm using Mistake Bound Theorem [1]. In this lecture, we will use probabilistic intuition to discuss the number of examples one needs to see before we can say that our learned hypothesis is good. To start we will pose the same learning conjunction problem introduced in Lecture 4 to refresh your mind.

1.1 Learning Conjunction

Assume that you want to learn the following hidden monotone conjunction [2]:

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Without going into any details think about these questions: How many examples are needed in order to learn the function? How is it learned from these examples? There are multiple possible learning protocols, we will cover the following three:

- **Protocol I:** The learner proposes instances as queries to the teacher[3].
- **Protocol II:** The teacher provides training examples. Since the teacher knows the hidden function f, it provides good training sets, that allow the learner to learn quickly.
- **Protocol III:** Some random source (e.g. Nature) provides training examples; the teacher (Nature) provides the labels ($f(x)$)

[1] Refer to Lecture 4 notes

1.1.1 Protocol III

This protocol is most often studied in machine learning, partially because it's more natural and is easier to analyze. In the this protocol, some random source provides training examples and a teacher provides the labels ($f(x)$). Some of the given examples can be as follows:

$$< (1,1,1,1,1,1,\ldots,1,1),1 >$$
$$< (1,1,1,0,0,0,\ldots,0,0),0 >$$
$$< (1,1,1,1,1,0,\ldots,0,1,1),1 >$$
$$< (1,0,1,1,1,0,\ldots,0,1,1),0 >$$
$$< (1,1,1,1,1,0,\ldots,0,0,1),1 >$$
$$< (1,0,1,0,0,0,\ldots,0,1,1),0 >$$
$$< (1,1,1,1,1,1,\ldots,0,1),1 >$$
$$< (0,1,0,1,0,0,\ldots,0,1,1),0 >$$

One algorithm we can use in this protocol is elimination. Start with the set of all literals as candidates, if we see a positive example, and some of the literals are zeros in that example, we can conclude that those 0 literals are unimportant (given that our conjunction is monotone). These variables with value 0 can thus be eliminated. But we are not able to conclude anything when the example is negative.

Table 1: Queries for Protocol III

Example	Conclusion
$< (1,1,1,1,1,1,\ldots,1,1),1 >$	$f = x_1 \land x_2 \land x_3 \land x_4 \land x_5 \land \ldots \land x_{100}$
$< (1,1,1,0,0,0,\ldots,0,0),0 >$	learned nothing
$< (1,1,1,1,1,0,\ldots,0,1,1),1 >$	$f = x_1 \land x_2 \land x_3 \land x_4 \land x_5 \land x_{99} \land x_{100}$
$< (1,0,1,1,0,0,\ldots,0,0,1),0 >$	learned nothing
$< (1,1,1,1,1,0,\ldots,0,0,1),1 >$	$f = x_1 \land x_2 \land x_3 \land x_4 \land x_5 \land x_{100}$

In this learning approach, each example, either modifies the hypothesis (by eliminating a variable in some positive examples) or doesn't change learner's hypothesis (negative examples). The key difference between this protocol and the previous two protocols is that the input in this protocol in random (naturally generated). This means that it is not guaranteed to learn the hidden conjunction exactly. Given that our target function is a conjunction, however, we can still say something meaningful about the behavior of the output. We cannot simply use the number of examples to determine the amount of time it takes before learning a satisfying function since there would be 2^{100} possible examples. We can analyze the time to learn something useful using two following approaches:

- **Probabilistic Intuition:** We consider the probability of one variable in the hidden conjunction never appearing in the example set. This probability is very small, so we can argue that the learned concepts perform well on future data that is distributed similarly to our training data. This approach is a key idea of the Probably Approximately Correct (PAC) framework that we will go over in this lecture note.

- **Mistake Driven Learning:** We can say something important about the performance of our learned hypothesis whenever making a mistake on a given example. Intuitively, if we correct the hypothesis for the mistakenly classified example, we can assume that the modified hypothesis performs better on future data. Using this approach, we can focus on the number of mistakes we are going to make until we are satisfied with our hypothesis as a measure of performance. We already discussed this in lecture 4.

1.2 Prototypical Concept Learning

Before the start we need to define a couple of notations:

- Instance space: X, examples (described by binary attributes)
- Concept space: C, the set of possible target functions where $f \in C$ is the hidden target function (for example all n-conjunctions or n dimensional linear functions)
- Hypothesis space: H, the set of possible hypotheses
- Training instances: $S_x\{0,1\}$, positive and negative examples of the target concept $f \in C$.

$$< x_1, f(x_1) >, < x_2, f(x_2) >, ..., < x_n, f(x_n) >$$

Training instances are generated by a fixed unknown probability distribution D over X.

We need to determine a hypothesis $h \in H$ such that $h(x) = f(x)$ for all $x \in S_x$ and a hypothesis $h \in H$ such that $h(x) = f(x)$ for all $x \in X$.

$$h = \underline{x_1} \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

This hypothesis $h \in H$ will estimate f and be evaluated by its performance on subsequent instances $x \in X$ drawn according to D (we have an inherent assumption that training and testing instances are drawn according to the same distribution).

1.3 Probably Approximately Correct (PAC) Learning - Intuition

In the hypothesis below:

$$h = \underline{x_1} \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Since feature x_1 was active in all of the positive examples $x_+ \in S_x$, it is very likely that it will also be active in future positive examples $x_+ \in X$ drawn from D. If not, in any case, feature x_1 is active only in a small percentage of the examples so our error will be small. Error is defined as:

$$Error_D = Pr_{x \in D}[f(x) \neq h(x)]$$

One way the set of predictions using $f(x)$ and $h(x)$ can be arranged is shown in Figure 1 (positive prediction are inside the set)

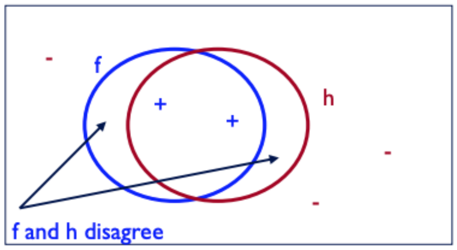

Figure 1: One possible arrangement of sets predictions on $f(x)$ and $h(x)$

Now the question becomes, given what we know about training instances S_x, can we bound the error?

1.3.1 Learning Conjunctions – PAC Analysis

If we let z be a literal and $p(z)$ be the probability that z is false in positive examples sampled from D. If z is in the target concept then $p(z) = 0$. Otherwise, $p(z)$ is the probability that z is deleted from h in a randomly chosen positive example. Which means we have:

$$Error(h) \leq \sum_{z \in h} p(z)$$

In this case, then h will make mistakes only on positive examples. A mistake is made only if a literal z that is in h but not in f is false in a positive example. In this case, h will predict negative, while the example is actually positive. Thus $p(z)$ is the probability that z causes h to make a mistake on a randomly drawn example from D. There may be overlapping reasons for mistakes, but the sum clearly bounds it. Figure 2 shows this case's set configuration.

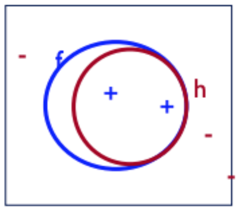

Figure 2: Set configuration for analysis 1

Lets call a literal z (in the hypothesis h) bad if $p(z) > \frac{\epsilon}{n}$. A bad literal is a literal that is not in the target concept and has a significant probability to appear false with a positive example. There are two possibilities:

- There are no bad literals: Since $Error(h) \leq \sum_{z \in h} p(z)$ then $Error(h) \leq \epsilon$
- There are some bad literals:

  Let z be a bad literal. The probability that it will not be eliminated by a given example is:

$$\begin{aligned} Pr(z \text{ survives one example}) &= 1 - Pr(\text{z is eliminated by one example}) \\ &\leq 1 - p(z) \\ &< 1 - \frac{\epsilon}{n} \end{aligned}$$

Now the probability that z will not be eliminated by m independent examples is, therefore:

$$Pr(z \text{ survives m independent examples}) \leq (1 - p(z))^m < (1 - \frac{\epsilon}{n})^m$$

There are at most n bad literals, so the probability that some bad literal survives m examples is bounded by $n(1 - \frac{\epsilon}{n})^m$.

We want $Pr(z \text{ survives m independent examples})$ to be small. So we want to choose a large enough m such that the probability that some z survive m examples is less than δ (that z remains in h, and makes it different from the target function f). This means we want:

$$n(1 - \frac{\epsilon}{n})^m < \delta$$

Using the fact that for $x > 0$ we have $1 - x < e^{-x}$ it is sufficient to require that:

$$ne^{-m\epsilon/n} < \delta$$

By rearranging we have that we need at minimum:

$$m > \frac{n}{\epsilon} \{ln(n) + ln(1/\delta)\}$$

examples to guarantee a probability of failure ($Error(h) > \epsilon$) of less than δ. So we have the theorem that with $m > \frac{n}{\epsilon} \{ln(n) + ln(1/\delta)\}$ then with probability $> 1 - \delta$, either:

- There are no bad literals

  or equivalently

- $Error(h) < \epsilon$

For example to have an error of less than %10 ($\epsilon = 0.1$) while having $\delta = 0.1$ and $n = 100$, we need 6907 examples at a minimum. With $\delta = 0.1$, $\epsilon = 0.1$, and $n = 10$, we only need 460 examples (for $\delta = 0.01$ we only need 690).

1.4 Formulating Prediction Theory

Before we continue to generalize our intuition lets reiterate some definitions and define some new notations:

- Instance space: X, input to the Classifier

- Output space: $Y = \{-1, +1\}$
- Current hypothesis: h, current hypothesis function $h : Y \leftarrow X$
- Concept space: C, the set of possible target functions where $f \in C$ is the hidden target function
- Hypothesis space: H, the set of possible hypotheses
- D: an unknown distribution over $X \times Y$
- Training instances: S, a set of examples drawn independently from D
- Sample size: $m = |S|$

We now can define two different errors:

- True Error: $Error_D = Pr_{(x,y) \in D}[h(x) \neq y]$
- Empirical Error: $Error_S = Pr_{(x,y) \in S}[h(x) \neq y] = \sum_{1,m}[h(x_i) \neq y_i]$

Now we can explore to answer the following questions:

- Can we describe/bound $Error_D$ given $Error_S$?
- Is it possible to learn a given function in C using functions in H, given the supervised protocol?

1.5 Requirements of Learning

We cannot expect a learner to learn a concept exactly since there will generally be multiple concepts consistent with the available data (which represents a small fraction of the available instance space.) Also, unseen examples could potentially have any label. We ”agree” to misclassify uncommon examples that do not show up in the training set.

We cannot always expect to learn a close approximation to the target concept since sometimes (only in rare learning situations we hope) the training set will not be representative (will contain uncommon examples). Therefore, the only realistic expectation of a good learner in that with high probability it will learn a close approximation to the target concept.

1.6 Probably Approximately Correct (PAC)

In Probability Approximately Correct (PAC) learning, one requires that given small parameters ϵ and δ, with probability at least $(1 - \delta)$, a learner produces a hypothesis with an error at most ϵ. Consistent Distribution allows us to hope for such a guarantee.

1.6.1 PAC Learnability

Consider a concept class C defined over an instance space X (containing instances of length n), and a learner L using a hypothesis space H.

"C is PAC learnable by L using H if for all $f \in C$, for all distribution D over X, and fixed $0 < \epsilon, \delta < 1$, L, given a collection of m examples sampled independently according to the distribution D produces with probability at least $(1 - \delta)$ a hypothesis $h \in H$ with error at most ϵ $(Error_D = Pr_D[f(x) \neq h(x)])$ where m is a polynomial in $1/\epsilon, 1/\delta, n$ and $size(H)$"

We can also say the following about efficient learnability:

"C is efficiently learnable if L can produce the hypothesis in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, n and $size(H)$"

In general, there are two limitations:

- Polynomial sample complexity (a condition on m; information-theoretic constraint): Is there enough information in the sample to distinguish a hypothesis h that approximate f?
- Polynomial time complexity (a condition on the efficiency of L; computational complexity): Is there an efficient algorithm that can process the sample and produce a good hypothesis h?

To be PAC Learnable there must be a hypothesis $h \in H$ with arbitrarily small error for every $f \in C$. We generally assume $H \supseteq C$ (properly PAC learnable if $H = C$).

The algorithm must meet its accuracy:

- For every distribution (the distribution free assumption)
- For every target function f in the class C

1.6.2 Occam's Razor

Using Occam's Razor [4] we can prove the claim that:

The probability that there exists a hypothesis $h \in H$ that:

- is consistent with m examples and
- satisfies $Error(h) > \epsilon$ $(Error_D(h) = Pr_D[f(x) \neq h(x)])$ is less than $|H|(1 - \epsilon)^m$

[4] Occam's Razor is the problem-solving principle that "entities should not be multiplied without necessity. When presented with competing hypotheses about the same prediction, one should select the solution with the fewest assumptions"

We can prove the claim using the following logic:

Let h be such a bad hypothesis. The probability that h is consistent with one example of f is

$$Pr_{x \in D}[f(x) = h(x)] < 1 - \epsilon$$

Since the m examples are drawn independantly of each other the probability that h is consistent with m examples of f is less than $(1-\epsilon)^m$. The probability that *some* hypothesis in H is consistent with m examples is less than $|H|\,(1-\epsilon)^m$. We want this probability to be smaller than δ, that is:

$$\begin{aligned} |H|\,(1-\epsilon)^m &< \delta \\ \ln(|H|) + m\ln(1-\epsilon) &< \ln(\delta) \\ \frac{1}{-\ln(1-\epsilon)}\left\{ln(|H|) + ln(1/\delta)\right\} &> m \\ m &> \frac{1}{\epsilon}\left\{ln(|H|) + ln(1/\delta)\right\} \end{aligned}$$

Last step is true because $e^{-x} = 1 - x + x^2/2 + \ldots$; $e^{-x} < 1 - x$ thus $ln(1-\epsilon) < -\epsilon$. It is called Occam's razor because this indicates a preference toward small hypothesis spaces.

1.6.3 Consistent Learners

Using the definitions from earlier, we get the following general scheme for PAC learning (to be a consistent leaner):

Given a sample D of m examples:

- Find some $h \in H$ that is consistent with all m examples:
  - Show that if m is large enough a consistent hypothesis must be close enough to f.
  - Check that we don't have too many examples (polynomial in the relevant parameters), that h we showed that the closeness to the target function guarantee requires m that satisfies $m > \frac{1}{\epsilon}\left\{ln(|H|) + ln(1/\delta)\right\}$
- Show that the consistent hypothesis $\in H$ can be computed efficiently

We have a consistent learner in the case of our conjunction example mentioned earlier. In the conjunction case since we showed that if we have sufficiently many examples (polynomial in the parameters), then h is close to the target function. We also used the Elimination algorithm to find a hypothesis h that is consistent with the training set (easy to compute).

1.6.4 Examples

Here are some examples:

- Conjunction:

  The size of the hypothesis space is 3^n since there are 3 choices for each feature (appear as a positive, appear as a negative, not appear at all).

$$\begin{aligned} m >& \frac{1}{\epsilon}\left\{\ln(3^n) + \ln(\frac{1}{\delta})\right\} \\ & \frac{1}{\epsilon}\left\{n \ \ln 3 + \ln(\frac{1}{\delta})\right\} \end{aligned}$$

  If we want to guarantee a 95% chance of learning a hypothesis of at least 90% accuracy, with $n = 10$ Boolean variable, $m > 140$. Minimum number of examples increases to 1130 when we change $n = 100$. Since the number of examples grows logarithmic with δ, changing the confidence to 99% only increases the number of required examples to 1145. These results hold for any consistent learner.

- K-CNF

  We want to show that the class of K-CNF functions is PAC learnable. Here is an example of a member of this class of functions:

$$f = \bigwedge_{i=1}^{m}(l_{i_1} \vee l_{i_2} \vee \ldots \vee l_{i_k})$$

  To develop an Occam Algorithm (Consistent Learner algorithm) for a hidden $f \in$ K-CNF we draw a sample D of size m:

  – Determine sample complexity:
    $f = C_1 \wedge C_2 \wedge ... \wedge C_m$ where $C_i = l_1 \vee l_2 \vee ... \vee l_k$:

$$\begin{aligned} \ln(|\textbf{K-CNF}|) \approx& \ln(2^{(2n)^k}) \\ & O(2n)^k \end{aligned}$$

    which means that $\log(|H|)$ is polynomial in n since k is a fix (constant) number. Due to the sample complexity result, h is guaranteed to be a PAC hypothesis if we can use the m examples to learn a consistent hypothesis.

  – Find a hypothesis h that is consistent with all the examples in D:
    Define a new set of features (literals), one for each clause of size k:

$$y_j = l_{i_1} \vee l_{i_2} \vee \ldots \vee l_{i_k}; j = 1, 2, \ldots, n_k$$

    use the algorithm for learning monotone conjunctions, over the new set of literals. We know that the algorithm is efficient. For example for $n = 4$ and $k = 2$ we have: $y_1 = x_1 \vee x_2, y_2 = x_1 \vee x_3, y_3 = x_1 \vee x_4, y_4 = x_2 \vee x_3, y_5 = x_2 \vee x_4, y_6 = x_3 \vee x_4$ which means we do the following transformation:

Original examples: $(0000, 1), (1010, 1), (1110, 1), (1111, 1)$
New examples: $(000000, 1)(111101, 1)(111111, 1)(111111, 1)$

1.6.5 Why Should We Care?

We now have a theory of generalization in which we know what the important complexity parameters are. We also understand the dependence in the number of examples and in the size of the hypothesis class. In addition, we have a generic procedure for learning that is guaranteed to generalize well:

1. Draw a sample of size m
2. Develop an algorithm that is consistent with it
3. It will be good.

Finally, we have tools to prove that some hypothesis classes are learnable and some are not which is quite powerful.

1.6.6 Negative Results for Learning

There are two types of nonlearnability results:

- Complexity theoretic: It is about showing that various concept classes cannot be learned based on well accepted assumptions from computational complexity theory. For example, determining that C cannot be learned unless P=NP. Examples:
  - k-term DNF, for $k > 1$ (k-clause CNF, $k > 1$)
  - Neural Networks of fixed architecture (3 nodes; n inputs)
  - Boolean formulas that are "read-once"
  - Quantified conjunctive concepts
- Information-Theoretic: The concept class is sufficiently rich that a polynomial number of examples may not be sufficient to distinguish a particular target concept. The proof shows that a given class cannot be learned by algorithms using hypotheses from the same class. Examples:
  - DNF Formulas
  - CNF Formulas
  - Deterministic Finite Automata
  - Context-Free Grammars

Both unlearnability types involve "representation dependent" arguments. Usually, proofs are for EXACT learning but apply for the distribution free case.

1.7 Agnostic Learning

In many cases, you may not be able to find a hypothesis that is completely consistent with the training data. We still want to be able to say something rigorous about accuracy over unseen data. This is where Agnostic learning comes in. Assume we are trying to learn a concept f using hypotheses in H, but $f \notin H$. In this case, our goal should be to find a hypothesis $h \notin H$, with a small training error on our training data X_T (since we won't be able to find a hypothesis completely consistent with the training data):

$$Error_{\text{TRAIN}}(h) = \frac{1}{m} |\{x \in X_T; f(x) \neq h(x)\}|$$

We want a guarantee that a hypothesis with a small training error will have a good performance on unseen examples,$Error_D(h) = Pr_{x \in D}[f(x) \neq h(x)]$. Hoeffding bounds allow us to get such guarantee since it characterizes the deviation between the true probability of some event and its observed frequency over m independent trials (where p is the underlying probability of the binary variable being 1 and p_{emp} is the probability of what we observe empirically):

$$Pr[p > p_{emp} + \epsilon] < e^{-2m\epsilon^2}$$

Therefore, the probability that an element in H will have a training error off by more than ϵ can be bounded as follows:

$$Pr[Error_D(h) > Error_{\text{TRAIN}}(h) + \epsilon] < e^{-2m\epsilon^2}$$

Using the same analysis as explained in section 1.3.1, with $\delta = |H|e^{-2m\epsilon^2}$, we get a generalized bound – a bound on how much the true error, $Error_D$ will deviate from the observed error, TRAIN. For any distribution D generating training and test instances, with probability at least $1 - \delta$ over the choice of the training set of size m, (drawn ——D), for all $h \in H$:

$$Error_D(h) < Error_{\text{TRAIN}}(h) + \sqrt{\frac{\log|H| + log(\frac{1}{\delta}}{2m}}$$

An agnostic learner which makes no commitment to whether f is in H and returns the hypothesis with the least training error over at least m number of examples can guarantee with a probability of at least $(1 - \delta)$ that its training error is not off by more than ϵ from the true error. We then have the following bound on the number of examples m (notice that learnability depends upon the log of the size of the hypothesis space similar to the consistent learners discussed earlier):

$$m > \frac{1}{2\epsilon^2} \left\{ \ln(|H|) + \ln(\frac{1}{\delta}) \right\}$$

1.8 Infinite Hypothesis Space

So far all of our analysis has been on finite hypothesis spaces. The way our bounds used the size of the hypothesis space as a measure of expressiveness is not going to hold valuable in the case of infinite hypothesis spaces. Some infinite hypothesis spaces are more expressive than others. For example, hypothesis space of 17 side convex polygons is more expressive than space of rectangles.

To be able to update our bounds for cases in which hypothesis space is infinite, we need to come up with a measure that can determine expressiveness of various infinite hypothesis spaces. The Vapnik-Chervonenkis dimension (VC dimension) provides such a measure. Analogous to $|H|$, there are bounds for sample complexity using $VC(H)$.

1.8.1 Shattering

We can measure $VC(H)$ of hypothesis space H using a concept called Shattering. We say that a set S of examples is shattered by a set of functions H if for every partition of the examples in S into positive and negative examples there is a function in H that gives exactly these labels to the examples. The intuition here is that a rich set of functions shatters large sets of points (thus is more expressive).

1.8.2 Examples

Here are some example concept classes with explanation on the max set number of points they can shatter:

- Left bounded intervals on the real axis: $[0, a)$, for some real number $a > 0$. In here set of all possible a that define our hypothesis space is infinite.

  Sets of two points cannot be shattered (we mean: given two points, you can label them in such a way that no concept in this class will be consistent with their labeling) while sets of one point can be shattered. Figure 3 shows why sets of two points cannot be shattered by this concept class.

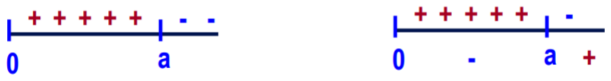

Figure 3: Left image shows that sets of one point can be shattered while the right image shows that sets of two cannot

- Intervals on the real axis: $[a, b]$, for some real numbers $b > a$:

  Sets of three points cannot be shattered (we mean: given three points, you can label them in such a way that no concept in this class will be consistent with their labeling) while sets of one and two points can. Figure 4 shows why sets of three points cannot be shattered by this concept class.

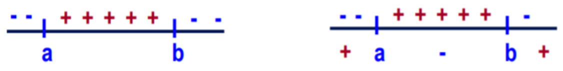

Figure 4: Left image shows that sets of two point can be shattered while the right image shows that sets of three cannot

- Half-spaces in the plane:

  In this case sets of four points cannot be shattered while sets of up to three can be shattered. Figure 4 shows why sets of four points cannot be shattered by this concept class. This is because the fourth point doesn't necessarily have to be in the convex hull of the other three points.

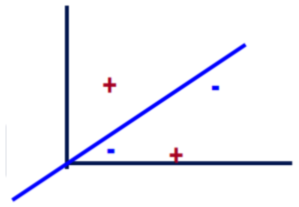

Figure 5: A counter example that shows sets of four cannot be shattered by half-space in the plane concept class

1.8.3 VC Dimension

The VC Dimension of hypothesis space H over instance space X is the size of the largest finite subset of X that is shattered by H. This means that there are two steps to proving that $VC(H) = d$:

- If there exists a subset of size d that can be shattered, then $VC(H) \geq d$
- If no subset of size $d + 1$ can be shattered, then $VC(H) < d + 1$

This means that the hypothesis examples H given above have VC dimensions as below:

- Left bounded intervals on the real axis: $VC(H) = 1$
- Intervals on the real axis: $VC(H) = 2$
- Half-spaces in the plane: $VC(H) = 3$

Since an unbiased hypothesis space H shatters the entire instance space X, i.e, it is able to induce every possible partition on the set of all possible instances, the larger the subset of X that can be shattered, the more expressive a hypothesis space is, i.e., the less biased. This way we can see that $VC(H)$ can be used as a measure of expressiveness of hypothesis space.

1.8.4 Sample Complexity Bounds

Using $VC(H)$ as a measure of expressiveness we have an Occam algorithm for infinite hypothesis spaces. Given a sample D of m examples, find some $h \in H$ that is consistent with all m examples if,

$$m > \frac{1}{\epsilon}\left\{8VC(H)\log\frac{13}{\epsilon} + 4\log(\frac{2}{\delta})\right\}$$

Then with probability, at least $(1 - \delta)$, h has an error less the ϵ (that is, if m is polynomial we have a PAC learning algorithm. To be efficient we need to produce the hypothesis h efficiently. Notice that to shatter m examples it must be that $|H| > 2^m$, which means $\log(|H|) \geq VC(H)$.

1.8.5 Learning Axis Parallel Rectangles

Consider axis-parallel rectangles in the real plane. We want to determine whether this infinite hypothesis is PAC learnable or not. We need to determine its VC dimension first:

- There is a subset of 4 points that can be shattered (we need to consider 16 different rectangles here.). This means $VC(H) \geq 4$
- But there is no subset of size five that can be shattered. You can see one subset in which the subset is not shattered by the hypothesis space in Figure 6

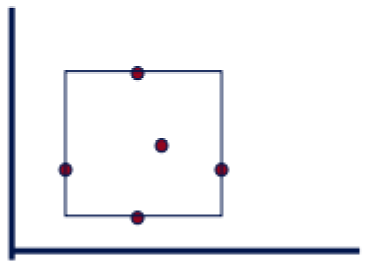

Figure 6: A counter example that shows sets of five cannot be shattered by the defined rectangles concept class

The two points above mean that $VC(H) = 4$. Since the VC dimension is finite, as far as sample complexity, the hypothesis space is guaranteed to be PAC learnable. Now we have to check if we can find an algorithm to efficiently learn the target function. we can use the following algorithm: Find the smallest rectangle that contains the positive examples (necessarily, it will not contain any negative example, and the hypothesis is consistent). Using this algorithm we can see that axis parallel rectangles are efficiently PAC learnable.

1.8.6 Sample Complexity Lower Bound

Similar to the finite hypothesis case, there is a general lower bound on the minimum number of examples necessary for PAC learning in the general case. Consider any concept class C such that $VC(C) > 2$, any learner L and small enough ϵ, δ. Then there exists a distribution D and a target function in C such that if L observes less than

$$m = \max[\frac{1}{\epsilon}\log(\frac{1}{\delta}), \frac{VC(C) - 1}{32\epsilon}]$$

examples, then with probability at least δ, L outputs a hypothesis having $Error(h) > \epsilon$. Ignoring the constant factors, the lower bound is the same as the upper bound, except for the extra $\log(\frac{1}{\epsilon})$ factor in the upper bound.

1.9 Computational Learning Theory Conclusions

The PAC framework provides a reasonable model for theoretically analyzing the effectiveness of learning algorithms. The sample complexity for any consistent learner using the hypothesis space, H, can be determined from a measure of H's expressiveness ($|H|$ for the finite case, $VC(H)$ for the infinite case). If the sample complexity is tractable, then the computational complexity of finding a consistent hypothesis governs the complexity of the problem. Sample complexity bounds given in this case are far from being tight, but they do separate learnable classes from non-learnable classes (and show what's important). Computational complexity results also exhibit cases where information-theoretic learning is feasible, but finding a good hypothesis is intractable.

The theoretical framework allows for a concrete analysis of the complexity of learning as a function of various assumptions (e.g., relevant variables). Many additional models have been studied as extensions of this basic framework and they include: learning with noisy data, learning under specific distributions, learning probabilistic representations, learning neural networks, learning finite automata, active learning (learning with queries) and, models of teaching. One important extension that we didn't talk about in this lecture is the PAC-Bayesians theory. PAC-Bayesians theory, in addition to the Distribution Free assumption of PAC, makes also an assumption of a prior distribution over the hypothesis the learner can choose from.

Theoretical results also shed light on important issues to consider in learning. These issues include the importance of bias (representation), sample and computational complexity, interaction, etc. The Bounds we talked about in this lecture can guide model selection even when they seem impractical at the first sight. A lot of the recent work on the Computational Learning Theory has been on finding data-dependent bounds that are usually tighter. The impact the theory of learning has had on practical learning systems in the last few years has been very significant since it has helped to develop algorithms such as SVMs, Winnow, and popularized concepts such as Boosting and Regularization.