CS-621 Theory Gems September 20, 2012

Lecture 2

Lecturer: Aleksander Mądry *Scribes: Carsten Moldenhauer and Robin Scheibler*

1 How Can We Teach a Computer to Learn?

Traditionally, whenever we want to use a computer to solve a new task, we need to provide it with very explicit and precise description of what it is exactly that we want it to do. This requirement often is not too problematic – after all, there is a lot of tasks that even not too experienced programmer (let alone a team of experienced ones) can make computers perform. However, there still is a lot of scenarios in which just coming up with a description of the task at hand that is sufficiently explicit and precise for a computer might be the core of the problem.

A canonical example here is email spam filtering. Namely, if you see an email message in your inbox, you can immediately tell if it is a spam or not. On the other hand, specifying – in a precise, succinct, and computer-readable way – what it means for an email message to be a spam might be rather challenging.

Let us take a closer look at this problem. We can view each email message as an object that is described by a set of *features*: the *From* address, *Reply-to* address, Subject, presence of certain keywords in its body, etc. Now, the traditional approach to dealing with spam messages would be based on looking at the pool of the emails we got so far and providing our email system with a set of filtering rules that would help distinguish the emails in this pool that are spam from the ones that are not. For example, we could setup rules like:

- if the email contains keyword 'viagra' $\longrightarrow$ SPAM
- if the email contains keyword 'sex' and *From* is not 'kinkyfriend@gmail.com' $\longrightarrow$ SPAM
- and so on...

However, as the pool of the emails becomes bigger and the spamers more inventive, coming up with new, more accurate rules can quickly become quite tedious and difficult. Also, note that some of the good rules might be rather non-obvious to us (for instance, a good indication of a message being a spam might be that the *From* and *Reply-to* addresses are different). Even more crucially, it is not clear at all that a set of rules that correctly classifies/labels the emails that we have already seen, will still do equally good job at classifying/labeling the messages that we will get in the future. (After all, good performance on yet-unseen messages is what we really care about here.)

All of the above makes us wonder if we could somehow automate this whole process. In other words, can we make the computer "learn" to recognize the spam – or any other concept – on its own while getting only minimal guidance from us?

This challenge can be rephrased in two fundamental questions:

1. Can an algorithm efficiently infer correct prediction rules from a collection of labeled examples?
2. How well do these inferred rules generalize (i.e., how well they work on new, unlabeled examples)?

Providing an answer to these two questions is a cornerstone of the field of Machine Learning and underlies many of the spectacular successes of Computer Science in the last two decades.

2 PAC-Learning Model

Before we go further, we need to make things a bit more precise. In particular, we need to introduce a formal model of learning. The model we will use is the Probably Approximately Correct (PAC) model.

In this framework, we have a universe X of "objects" and a distribution $\mathcal{D}$ over these objects. Furthermore, there is a set L of possible *labels* and a function f (sometimes called the *concept*) that

classifies each object in X by assigning a label to it. (So, in our spam detection example, X is the set of all possible emails that could be written, $\mathcal{D}$ is the distribution that captures the pattern of emails that we might get, L consists only of two labels, say -1 and 1, and f assigns a label of 1 to a message that is a spam, and yields -1 otherwise.)

Now, the way the learning process proceeds is that we are presented with a set $S = \{(x^1, l^1), \ldots, (x^m, l^m)\}$ of m *independent* samples from X (according to the distribution $\mathcal{D}$) together with their correct labels (i.e., $l^j = f(x^j)$ for each j). Our goal is to use this information to output a *hypothesis* h that with good probability approximates the function f on the whole universe X (with respect to the distribution $\mathcal{D}$). Formally, we want h to be a (ε, δ)-*PAC learner* for f, which means that for some (small) parameters $\varepsilon, \delta > 0$, we want that with probability at least $(1 - \delta)$ (over the choice of the set of samples S)

$$Pr_{x \in \mathcal{D}}[h(x) \neq f(x)] < \varepsilon,$$

i.e., h mis-classifies a random sample chosen from $\mathcal{D}$ with probability at most ε.

Note that in this model coming up with the hypothesis h corresponds to an answer to the first fundamental question that we posed earlier. On the other hand, requiring h to be the (ε, δ)-PAC learner ensures that it also has the generalization property that the second question asks for.

Finally, observe that this model makes an assumption that the samples in the set S were chosen independently – depending on application, this might or might not be realistic.

3 Finding a "Good" Hypothesis

One crucial aspect of the learning process that the PAC model is not specifying (and that is exactly the key challenge in every learning scenario) is the choice of the hypothesis h. So, we want to think a bit about this problem.

A natural requirement for h would be to ensure that it classifies in the first place all the labeled samples in S. This is definitely a reasonable requirement – after all, if h is not able to do good job on the samples that we are given then why should it do well on the yet-unseen ones – but there are some pitfalls here.

To see the problem, note that there is always a hypothesis that would classify all the available samples: a look-up table. Given the set $S = \{(x^1, l^1), \ldots, (x^m, l^m)\}$ we can just take our hypothesis h to be

$$h(x) = \begin{cases} l^1 & \text{if } x = x^1 \\ l^2 & \text{if } x = x^2 \\ \vdots & \\ l^m & \text{if } x = x^m \\ \text{arbitrary label} & \text{otherwise} \end{cases}.$$

Clearly, such a hypothesis is easy to come up with and does a great job at classifying all the data in S. However, we do not really expect it to generalize well. What is a good choice of h then?

It turns out that the best way to go about the choice h is to follow so-called *Occam's razor* principle that roughly tells us that we should always strive to take h to be the "simplest" hypothesis that explains our labeled data. Of course, there is many notions of simplicity that one might think of and, in case of PAC learning, there is actually a very precise and elegant answer based on the concept of *VC-dimension*.

However, just to illustrate why simplicity might be a good guiding principle here, we show that h is guaranteed to have good generalization properties whenever it is coming from a relatively small set H of possible classifiers. That is, we show that if there is a set of classifiers H such that: (1) we are always able to find a hypothesis h in H that explains all our labeled data; (2) $|H|$ is relatively small; then the resulting h has good generalization properties (as long as, the number of labeled examples that we have is reasonably large).

To this end, let us fix some family H of classifiers, as well as, parameters $\varepsilon, \delta > 0$ and let us denote by H_{BAD} the subset of H that consists of the ones among the classifiers that do not generalize sufficiently well. That is, let

$$H_{\text{BAD}} \subset H \text{ s.t. } \forall h \in H_{\text{BAD}} \quad Pr_{x \in \mathcal{D}}[h(x) \neq f(x)] \geq \varepsilon.$$

Let us focus on some $h' \in H_{\text{BAD}}$. What is the probability that it might fool us, i.e., that it will explain correctly all the examples in S? As each sample is independent and h', by definition, has at least ε probability of mis-classifying it, this happens with probability of at most $(1-\epsilon)^{|S|}$. (Note that this is the moment when we crucially use the sample independence assumption from the PAC model.)

Therefore, taking a union bound over all elements of H_{BAD}, we get

$$\begin{aligned} &Pr[\text{any } h \in H_{\text{BAD}} \text{ labels all examples correctly}] \\ &\qquad \leq \sum_{h \in H_{\text{BAD}}} Pr[h \text{ answers correctly all examples}] \\ &\qquad \leq |H_{\text{BAD}}|\,(1-\epsilon)^{|S|}. \end{aligned}$$

In the context of (ε, δ)-PAC learning, we want this quantity $|H_{\text{BAD}}|(1-\epsilon)^{|S|}$ to be at most δ. So, by using that $|H_{\text{BAD}}| \leq |H|$, we get a lower bound on the size of $|S|$

$$|S| \geq \frac{1}{\epsilon}\left(\ln|H| + \ln\frac{1}{\delta}\right).$$

So, as long as S is at least that large, we are guaranteed to have the (ε, δ)-PAC learning property.

Note that the above bound also hints at why taking a look-up table that we have seen before, as your hypothesis might not be a good idea. The size of all the possible look-up table that correctly classifies a set of m labeled examples is 2^m. So, the above argument fails to provide any generalization bound as plugging $|H| = 2^{|S|}$ into the lower-bounding inequality leads to contradiction. (One should keep in mind, however, that this is *not* a proof that this type of hypothesis will not generalize well - it is just an indication of some potential problem.)

Finally, it is worth pointing out again that there is a certain tension in the choice of H here. On one hand, we want it to be small/simple enough so the arguments along the ones presented above, yield good generalization bounds. But, on the other hand, H should be rich enough to be able to explain reasonably well all the sets of labeled examples that we might be presented with. This means that choosing the right set of classifiers to work with is an integral and delicate part of solving a learning problem.

4 Learning Linear Classifiers

As we discussed above, choosing the right set of classifiers is a crucial decision when tackling a learning problem. However, even once we have made a good choice of our set H of possible classifiers, there is still one problem that we need to solve: finding, for a given set of samples S, a classifier in H that correctly labels them. (Note that at this point this is a purely algorithmic problem. We already know that there exists some correct classifier in H, we just need to find it efficiently.)

As one might expect, algorithms for this kind of task are usually specialized to the particular set of classifier one wants to work with. Therefore, from now on, we will focus our attention on a simple, but very powerful and popular class: linear separators.

In this settings, our objects are points in high-dimensional space, say $\mathbb{R}^n$, where each of n coordinates encodes (as a real number) one feature of the object. Also, our labels can be either 1 or -1. (Note that, at this moment, it is not important to us what $\mathcal{D}$ or even f is.) Our goal is to find (knowing that it exists), for a given a set of m labeled points $(x^1, l^1), \ldots, (x^m, l^m)$, a vector $w \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that

$$\text{sign}(w \cdot x^j - \theta) = l^j \qquad \forall j = 1, \ldots, m.$$

Geometrically, the above condition means that the hyperplane that is orthogonal to w and is at the offset θ from the origin, separates all the x^js with label 1 from the ones with label -1.

Before we proceed, we notice that without loss of generality we can assume that $\theta = 0$ (so our hyperplane goes through the origin) and that all labels l^j are positive (i.e., equal to 1). To do the former, note that we can incorporate θ as a part of w if we add one more dimension to our space and set each x^j to have -1 on this dummy coordinate. To do the latter, we just map each x^js with $l^j = -1$ to a vector $-x^j$ with a label 1. Clearly, if our solution separates the data after this transformation then it also was separating the original one.

Also, we can always assume that all the data is normalized, as normalization does not affect separation condition (recall that our hyperplane goes through the origin now). Thus, in what follows, we will assume that all x^j have an l_2-norm of one.

In the light of above, our task simplifies now to:

$$\begin{gathered}\text{given } x^1, \ldots, x^m \in \mathbb{R}^n \text{ with each } \|x^j\|_2 = 1, \\ \text{find } w \in \mathbb{R}^n \text{ such that} \\ w \cdot x^j > 0 \qquad \forall j = 1, \ldots, m.\end{gathered}$$

Note that this kind of task captures a pretty general case of Linear Programming (LP) (we will discuss LPs later in the course). Thus, in principle, one could use the general LP algorithms to solve it. However, we want to describe instead a simple (and usually much more efficient) algorithm for this problem called *Perceptron* algorithm.[1]

Perceptron Algorithm:

1. Start with $w^0 \leftarrow (0, \ldots, 0)$
2. In round t:
   (a) check if $\exists j_t$ with $w^t \cdot x^{j_t} \leq 0$
   (b) if not: output w^t
   (c) otherwise: set $w^{t+1} \leftarrow w^t + x^{j_t}$ and repeat;

So, this algorithm in each round checks if there is a sample (x^{j_t}) that is not yet classified correctly. If there is no such sample then clearly we have found the correct classifiers. Otherwise, we add offending example to our classifier. Note that by doing this we will improve the classification of x^{j_t}, because

$$w^{t+1} \cdot x^{j_t} = w^t x^{j_t} + x^{j_t} x^{j_t} = w^t x^{j_t} + 1 > w^t \cdot x^{j_t}$$

We analyze now the number of iterations of this algorithm (assuming that there is a linear separator for the data).

Theorem 1 (Termination of Perceptron) *If w^* is a linear classifier that classifies all the samples correctly and $\gamma = \min_j \frac{w^* \cdot x^j}{\|w^*\|_2}$ then Perceptron finishes in at most $T \leq \frac{1}{\gamma^2}$ iterations.*

Proof The proof uses a potential function argument. For each round t let

$$N_t = \|w^t\|_2^2 \qquad \text{and} \qquad A_t = w^t \cdot w^*.$$

Clearly, $A_0 = N_0 = 0$. We have

$$N_{t+1} = \|w^{t+1}\|_2^2 = \|w^t + x^{j_t}\|_2^2 = \|w^t\|_2^2 + 2\, w^t \cdot x^{j_t} + \|x^{j_t}\|_2^2 \leq N_t + 1$$

[1] Interestingly, it is actually possible to use Perceptron algorithm to solve LP problem in full generality in polynomial time.

since $w^t \cdot x^{j_t} \leq 0$ (as we only add incorrectly classified samples) and $\|x^{j_t}\|_2 = 1$. Furthermore,

$$A_{t+1} = w^{t+1} \cdot w^* = w^t w^* + x^{j_t} w^* \geq A_t + \gamma \|w^*\|_2.$$

Hence, by induction, $N_T \leq T$ and $A_T \geq \gamma \|w^*\|_2 T$. Using the Cauchy-Schwarz inequality ($\|a\|^2 \|b\|^2 \geq (a \cdot b)^2$) we have

$$N_T \|w^*\|_2^2 = \|w^T\|_2^2 \, \|w^*\|_2^2 \geq (w^T \cdot w^*)^2 = (A_T)^2.$$

Hence,

$$T \geq N_T \geq (A_T)^2 / \|w^*\|^2 \geq \gamma^2 T^2 \qquad \Rightarrow \qquad T \leq \frac{1}{\gamma^2}.$$

■