Lecture 14: Online Learning

Akshay Krishnamurthy
akshay@cs.umass.edu

October 27, 2017

1 Recap

So far we have mostly studied *statistical learning theory*, which focuses on batch learning under probabilistic assumptions. More specifically, we have mostly assumed that the learning algorithm has access to a training set S of n examples drawn iid from a distribution $\mathcal{D}$, and we would like the algorithm to produce a hypothesis or predictor that has low error (measured in some way) on fresh samples generated from $\mathcal{D}$ (i.e. low generalization error). We saw how to prove generalization bounds based on properties like uniform convergence and covering numbers, and we also saw some algorithms like boosting and SVM for the batch setting.

Today and for the next several weeks we will study a different learning paradigm called *online learning* or *sequential prediction*. This setting makes no probabilistic assumptions and hence does not care much about generalization. Nevertheless we will see connections to statistical learning and to game theory, as well as many new algorithms.

2 Motivation and Formalism

Consider the following example of weather prediction

Example 1. *We would like to learn how to make weather forecasts. For simplicity let us assume that we are only interested in predicting* $\{rain, no\ rain\}$*, so we are making a binary decision. Every night we make a prediction about the next day (using whatever information we have available), and the next day we get to see if it rained or not, so we can verify our prediction. We would like to predict well over the course of the year.*

The first observation here is that there is no training data. Instead, on every example (day), we must make a prediction without seeing the label, and only then is the true label available to us. At this point we can use the example for updating our model. So in some sense every example is first a test example and then a training example. The other main point is that modeling the days as iid would be a terribly unrealistic assumption, since the weather changes dramatically with the seasons. How should we go about relaxing the iid assumption? There are at least two options that make some sense.

1. Model the temporal nature of the seasons with some probabilistic time-series process. This does happen.
2. Try to relax all probabilistic assumptions and instead just model the sequences of days/weather as adversarial.

The former can make sense we when have a good sense of how the environment behaves, or when our probabilistic model is good. On the other hand, the latter involves no assumptions so it is always a good fit for the application, but can we learn in such an adversarial setting? The answer is yes, and such adversarial sequential prediction problems play a central role in online learning.

Formalism. In each round of a sequential prediction problem, the learner receives an instance $x_t \in \mathcal{X}$ and must make a prediction $\hat{y}_t \in \mathcal{Y}$, the learner then sees the true label y_t and receives a loss $\ell(\hat{y}_t, y_t)$. So far this is quite similar to statistical learning. The main difference is that now we remove any distribution assumptions and instead say that (x_t, y_t) are chosen by an adversary. This adversary is all powerful, and can choose (x_t, y_t) based on

everything that has happened so far in the learning process, but (something that will be important in a bit) does not have acces any randomness that the learner uses on round t.

The first question you should be asking is: how is it possible to learn anything at all when the adversary is so powerful? Indeed the adversary can force you to make a mistake on every round, so nothing interesting is going on. One way to say something nontrivial is to restrict the adversary. For example, let us make a restriction based on **realizability**, which we have seen before.

3 Realizability

Definition 1 (Mistake bound model)**.** *Let $\mathcal{H} \subset \mathcal{X} \to \{\pm 1\}$ be a hypothesis class and let A be an online learning algorithm. Given any sequence $S = (x_1, h^\star(x_1), \ldots, x_T, h^\star(x_T))$ of T labeled examples where $h^\star \in \mathcal{H}$, we define $M_A(S, h^\star)$ to be the number of mistakes that algorithm A makes on sequence S. We denote by $M_A(\mathcal{H})$ the supremum of $M_A(S, h^\star)$ over all sequences and all labeling functions $h^\star$. If there exists an algorithm A that satisfies a* mistake bound *of the form $M_A(\mathcal{H}) \leq B < \infty$, we say that $\mathcal{H}$ is online learnable in the mistake bound model.*

The main restriction we have made on the adversary is that labels must be generated by a labeling function $h^\star \in \mathcal{H}$ and the learner is aware of this hypothesis class. We are now asking that the number of mistakes that the learner makes is bounded by some quantity B that is independent of the length of the sequence. This seems sensible, it means that asymptotically (as $T \to \infty$) the average number of mistakes is going to zero, which seems to suggest that we are learning. Note however that this does not say that after some point t we will make no mistakes, it just says that over the sequence the total number of mistakes is bounded.

In a sense this is quite different from statistical learning. In this model you cannot say anything about the generalization performance of the learned model. For example it could be the case that for the first half of the rounds you actually make no mistakes (because the adversary gives you easy examples), but then the adversary starts giving you the harder examples so in the latter half you make a bunch of mistakes.

We will also see how to relax the realizability assumption. First let us turn to an algorithm. Since we are in the realizable model, a natural thing to do is to eliminate any hypothesis that has made a mistake so far. The only remaining design decision is to decide how to make a prediction. Start with $V_1 = \mathcal{H}$ and for $t = 1, \ldots, T$

1. Receive x_t
2. Predict $\hat{y}_t = \text{argmax}_{r \in \{-1,1\}} |\{h \in \mathcal{V}_t : h(x_t) = r\}|$.
3. Recieve true label $y_t = h^\star(x_t)$
4. Update $V_{t+1} \leftarrow \{h \in V_t : h(x_t) = y_t)\}$.

Theorem 2. *The Halving algorithm enjoys a mistake bound of $M(\mathcal{H}) \leq \log_2(|\mathcal{H}|)$.*

Proof. We track $|V_t|$. Clearly $|V_t| \geq 1$ since we will never eliminate $h^\star$. Moreover $|V_0| = |\mathcal{H}|$. Finally, since we predict with the majority, every time we make a mistake we must eliminate at least $1/2$ of the surviving hypothesis. Thus we get $|\mathcal{V}_{t+1}| \leq |\mathcal{V}_t|/2$. Putting this together, we get $1 \leq |\mathcal{V}_T| \leq |\mathcal{H}|2^{-M}$, which implies that $M \leq \log_2(|\mathcal{H}|)$. □

Littlestone dimension Just as we saw in the statistical learning setting, we have seen that in the online learning setting, we can learn finite classes in the presence of realizability. A natural question that you should all be asking is, can we also learn VC classes in mistake bound model? Unfortunately the answer is no, but there is another parameter called the **Littlestone dimension** that characterizes online learnability.

The intuition here is that the adversary can change the examples in a way to force more mistakes. Formally this can be modeled as a rooted binary tree where each node is associated with an example, and each edge is based on the prediction of the learner. So if the adversary shows you x_1 and you predict 1, the adversary will declare a mistake and traverse the right branch. The process repeats as we navigate down the tree. For realizability the adversary must only guarantee that every path you might traverse can be labeled by some $h^\star \in \mathcal{H}$. This motivates a combinatorial definition based on these trees.

Definition 3. *A shattered tree of depth d is a sequence of instances $v_1, \ldots, v_{2^d-1} \in \mathcal{X}$ such that for every labeling $y_1, \ldots, y_d \in \{0,1\}^d$ there exists some $h \in \mathcal{H}$ such that for all $t \in [d]$ we have $h(v_{i_t}) = y_t$ where $i_t = 2^{t-1} + \sum_{j=1}^{t-1} y_j 2^{t-1-j}$. The* Littlestone dimension *of $\mathcal{H}$, Ldim($\mathcal{H}$), is the largest T such that there exists a tree of depth T that is shattered by $\mathcal{H}$.*

The formal definition doesn't matter too much but the intuition is more important. The point is that the adversary may change the sequence on the fly based on your predictions, and all he must do is make sure that he satisfies the realizability property.

Theorem 4. *1. For any online learning algorithm, we must have $M_A(\mathcal{H}) \geq Ldim(\mathcal{H})$.*

2. There exists an algorithm (similar to halving) with $M_A(\mathcal{H}) \leq Ldim(\mathcal{H})$.

3. $VCdim(\mathcal{H}) \leq Ldim(\mathcal{H})$, this inequality can be strict and there can be arbitrarily large separation.

4 The Unrealizable setting

We can also say something non-trivial in the unrealizable setting. However, when we leave the realizable setting the mistake bound model no longer makes sense, since the adversary can now force us to make arbitrarily many mistakes. Instead a natural thing to do since we are using a hypothesis class $\mathcal{H}$ is to compare ourselves to the number of mistakes made by the best hypothesis in $\mathcal{H}$. This is captured by the definition of regret

$$\text{Regret}_A(h, T) = \sup_{(x_1,y_1),\ldots,(x_T,y_T)} \left[\sum_{t=1}^{T} \mathbf{1}\{\hat{y}_t \neq y_t\} - \sum_{t=1}^{T} \mathbf{1}\{h(x_t) \neq y_t\} \right]$$

The regret relative to $\mathcal{H}$ is just $\text{Regret}_A(\mathcal{H}, T) = \sup_{h \in \mathcal{H}} \text{Regret}_A(h, T)$. Let us unpack the definition a bit

1. If the sequence is realizable, then the definition reduces to the mistake bound definition, since there exists a hypothesis that makes no mistakes.

2. On the other hand, if the problem is not realizable, then we are asking for the learner's predictions to be just about as good as the best possible hypothesis. In a sense, we are learning which h is the best in hindsight, which is similar to the statistical learning setting, where we are interested in excess risk bounds. In particular if $\text{Regret}_A(\mathcal{H}, T) = o(T)$ then we are learning about the class $\mathcal{H}$.

3. Unfortunately, it is easy to see that for a deterministic algorithm we always have $\text{Regret}_A(\mathcal{H}, T) \geq T/2$, even when $|\mathcal{H}| = 2$. The idea is that the adversary, being all powerful, knows what prediction you are going to make and will force you to make a mistake. On the other hand, with two hypothesis (one predicting always 1 and the other always 0), one of them will make at most $T/2$ mistakes.

However, for randomized algorithms this is not possible, since the adversary won't know your prediction. Thus we will allow the algorithm to make randomized predictions, which we denote with $p_t \in \Delta(2)$. We also will slightly change the notation and say that on every round t, the learner receives an example x_t, makes a prediction p_t and recieves a loss vector $\ell_t \in [0,1]^2$ (where $\ell_t(y) = \mathbf{1}\{y_t \neq y\}$ for $y \in \{0,1\}$). The regret for the learner is

$$\text{Regret}(T) = \sum_{t=1}^{T} \langle p_t, \ell_t \rangle - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell_t(h(x_t))$$

Next time we'll see an algorithm that achieves sublinear regret.