Introduction to Machine Learning
Lecture 11
Online Learning

18.6.2015

Remark 1 *The main resource for this lecture is [3].*

1 Online Learning

- Batch learning:
  - Learning process is divided into two or three stages: Training, (validation) and testing. We assume that there is an underlying (unknown) distribution, $\mathcal{D}$, over $\mathcal{X} \times \mathcal{Y}$ such that both training and test examples are generated i.i.d. according to $\mathcal{D}$.
  - Good for stationary environments (vision, speech) in which the distribution does not change too often.
- In some cases, this model is not suitable. For example, in online routing or portfolio selection, one needs to adapt its behavior and make decisions in an online manner.
- Online learning:
  - The learner makes decisions in a sequential manner. At each time t, the learner makes a decision (e.g., prediction) and pays a cost which is determined by the environment.
  - Learning is described as a *repeated game* between a learner and the environment.
  - No statistical assumptions. In particular, the environment can be adversarial.

The online model seems to be more challenging than the batch model. Indeed, we will find several evidences for this intuition throughout this lecture.

2 Online Classification

Let $\mathcal{X}$ be a domain, $\mathcal{Y} = \{-1, 1\}$ and let $\mathcal{H} \subseteq \{-1, 1\}^{\mathcal{X}}$ be a hypothesis class. For simplicity, we focus on finite hypothesis class with N hypotheses. In the online literature, the hypothesis are often called *experts*. This name stems from the fact that at each time t, the learner receives the advice of the experts, and use this advice to decide on its next prediction. Formally, online classification is a repeated game between a learner, $\mathcal{A}$, and an environment, $\mathcal{E}$. At each time t, the game follows the following protocol:

1. The environment $\mathcal{E}$ chooses an example x_t and reveals it to the learner.
2. The algorithm $\mathcal{A}$ decides on a label $\hat{y}_t \in \{-1, 1\}$.
3. The enviornment $\mathcal{E}$ chooses a label y_t and reveals it to the learner.
4. The instantaneous loss (cost) of the learner is defined by $\hat{\ell}_t = \mathbf{1}_{[\hat{y}_t \neq y_t]}$.

The number of rounds is[1] denoted by T. The ultimate goal of the learner is to minimize the cumulative loss, which is defined by

$$\hat{L}_T = \sum_{t=1}^{T} \hat{\ell}_t \ .$$

Of course, without making any further assumptions on the environment, we have no hope to succeed in this task. Thus, similarly to the PAC model, we look at the relative suboptimality of the learner w.r.t. to the best hypothesis in $\mathcal{H}$. Namely, we define the regret of the learner by

$$R_T := R_T(\mathcal{A}) = \hat{L}_T - \min_{i \in [N]} L_T(i) \ , \tag{1}$$

where $L_T(i) = \sum_{t=1}^{T} \ell_t(i) = \sum_{t=1}^{T} \mathbf{1}_{[h_i(x_t) \neq y_t]}$. The goal of the learner is to minimize the average regret w.r.t every possible environment. This can be defined using minimx terminlogy. Denote the regret of A w.r.t. an environment $\mathcal{E}$ by $R_T^E(\mathcal{A})$. When we simply refer to the regret of A, we actually refer to $\max_E R_T^{\mathcal{E}}(\mathcal{A})$. Following this notation, our ultimate goal is to design design an (hopefully efficient) algorithm that approximately attains the minimax (optimal) regret

$$\min_{\mathcal{A}'} R_T(\mathcal{A}') = \min_{A'} \max_{\mathcal{E}} R_T^E(\mathcal{A}') \ .$$

The average regret is defined by $\frac{1}{T} R_T$. An online learning is considered to be learnable if the optimal average regret tends to zero as T tends to infinity. This captures the idea that we would like to perform as well as the best fixed hypothesis in hindsight.

2.1 Realizable setting

As in the PAC model, we start out tour by considering a simplified model in which we assume that there exists (at least one) hypothesis $h \in \mathcal{H}$ which classifies all the examples correctly. We next consider two general algorithms for this setting.

It is tempting to mimic the ERM algorithm. Indeed, the *Consistent* algorithm picks, at each time t, a hypothesis which is consistent with the examples seen so far. That is, at each time t, we follow the following rule:

$$\hat{h}_t \in \mathcal{H}_t := \{h_i \in \mathcal{H} : L_{t-1}(i) = 0\} \quad , \quad \hat{y}_t = \hat{h}_t(x_t) \ .$$

[1] We assume for simplicity that both players know the number of rounds

Theorem 1 *The algorithm* Consistent *makes at most $N-1$ mistakes. Thus,*

$$R_T \leqslant N-1 \ .$$

Proof Whenever the algorithm errs the size of $\mathcal{H}_t$ is reduced by at least 1. Also, by realizability, the set $\mathcal{H}_t$ never becomes empty. Since $|\mathcal{H}| = N$, it follows that the algorithm might err at most $N-1$ times. ■

The above proof suggests a more clever method to choose $\hat{h}_t$. Namely, instead of eliminating only one hypothesis for each mistake, we can eliminate the majority. This is done by choosing h_t according to the Majority rule. The resulting algorithm is called *Halving.* It follows the following rule.

$$\hat{y}_t = \begin{cases} 1 & \sum_{i:h_i \in \mathcal{H}} h_i(x_t) \geqslant 0 \\ -1 & \text{otherwise} \end{cases}$$

It can be seen that whenever the algorithm errs, at least half of the hypotheses are eliminated. Thus, we conclude the following bound.

Theorem 2 *The algorithm* Halving *makes at most $\lfloor \log(N) \rfloor$ mistakes. Thus,*

$$R_T \leqslant \lfloor \log(N) \rfloor \ .$$

2.2 General case

We now consider the general (agnostic) case. In this setting, it is not reasonable to remove hypothesis that errs. Still, we can apply the ERM rule. That is, we can apply the rule

$$h_t \in \arg\min_{h_i \in \mathcal{H}} L_{t-1}(i) \quad , \quad \hat{y}_t = \hat{h}_t(x_t) \ .$$

The resulting algorithm is called *Follow The Leader.* As we show next, this algorithm performs miserably. In particular, it fails to ensure a vanishing average regret. In fact, this negative result applies for any deterministic algorithm.

Theorem 3 *Assume that $\mathcal{H} \supseteq \{h_+, h_-\}$, where $h_+(x) = 1$ and $h_-(x) = -1$ for all x. Let $\mathcal{A}$ be a deterministic algorithm. Then, there exists an environment for which the average regret of $\mathcal{A}$ is at least* $1/2$.

Proof Consider the environment that returns $y_t = 1 - \hat{y}_t$ for all t. Consequently, $\hat{L}_t = T$. Since at each time t, exactly one of the hypotheses h_+ and h_- can err, we have

$$\min_{i \in N} L_T(i) \leqslant \min\{L_T(h_-), L_T(h_+)\} \leqslant T/2 \ .$$

Thus,

$$R_T \geqslant T - T/2 = T/2 \Rightarrow \frac{1}{T} R_T \geqslant 1/2 \ .$$

■

2.3 The role of randomization

To sidestep the above negative result, we now allow the learner to randomize its predictions. That is, we assume that the learner maintains a probability vector $p_t = (p_t(1), \ldots, p_t(N))) \in \{p \in \mathbb{R}^N_{\geqslant 0} : \sum_{i=1}^N p(i) = 1\}$ and predicts $\hat{y}_t$ according to p_t. Of course, this change has no effect if the environment is allowed to observe the random bits used by the learner. Thus, we will assume that the environment only knows the vector p_t when it decides on y_t. The cost of the learner is now defined to be the experted cost according to p_t. That is, we define

$$\hat{\ell}_t = \sum_{t=1}^{N} p_t(i)\ell_t(i) \ .$$

The regret is defined as in Equation (1). We next describe a successful algorithm for this setting. The algorithm is called *Multiplicative Weights* (MW)[2]. As discussed in a recent survey ([1]), this algorithm has many applications (e.g., boosting and solving zero-sum games approximately).

The idea of the algorithm is very simple. Instead of eliminating hypotheses that err, we maintain weights over the hypotheses. The weight of the i-th hypothesis at time t is defined by $w_t(i)$. These weights are updated according to the losses of the hypotheses. The probabilities for following each hypothesis is proportional to its weight. Initially, we define all weights to be equal. There are various successful methods for updating the weights. The most known are:

$$w_{t+1}(i) = w_t(i)(1 - \eta\ell_t(i)) \tag{2}$$

$$w_{t+1}(i) = w_t(i)\exp(-\eta\ell_t(i)) \ . \tag{3}$$

Both methods enjoy similar bounds on the average regret. We next analyze the first choice. A pseudo-code is given in Algorithm 12.

Algorithm 1 Multiplicative Weights

```
  **Input:** $N$ hypotheses (experts)
  **Parameter:** $\eta > 0$
  **Initialize:** $w_1 = (1, \ldots, 1) \in \mathbb{R}^N$
  **for** $t = 1$ **to** $T$ **do**
    Receive $x_t$
    $W_t = \sum_{i=1}^N w_t(i)$
    $p_t = w_t / W_t$
    choose expert $i$ with probability $p_t(i)$
    set $\hat{y}_t = h_i(x_t)$
    receive the loss vector $\ell_t = (\ell_t(i))_{i=1}^N = (\mathbf{1}_{[h_i(x_t) \neq y_i]})_{i=1}^N$
    update the weights: $(\forall i)$ $w_{t+1}(i) = w_t(i)(1 - \eta\ell_t(i))$
  **end for**
```

[2]The algorithm is also known as Hedge or Weighted Majority.

Theorem 4 *Let $\eta \in (0, 1/2)$. Assuming that $T \geqslant 4\ln(N)$, the regret of Algorithm 12 bounded by*

$$R_T \leqslant \frac{\ln(N)}{\eta} + \eta T \ .$$

Substituting $\eta = \sqrt{\ln(N)/T}$, we obtain that the average regret is at most $2\sqrt{\frac{\ln(N)}{T}}$.

The idea of the proof is to relate the "potential" W_t both to the cumulative loss of the learner and the cumulative loss of the best expert. In the first part of the proof we will show that if the loss of the learner is large, then the potential is small. In the second part we will show that if the potential is small, the loss of the best expert must be large.

Proof

First part:

$$\begin{aligned} W_{t+1} &= \sum_{i=1}^{N} w_{t+1}(i) = \sum_{i=1}^{N} w_t(i)(1 - \eta\ell_t(i)) \\ &= W_t - \eta \sum_{i=1}^{N} w_t(i)\ell_t(i) = W_t - \eta W_t \sum_{i=1}^{N} \frac{w_t}{W_t}(i)\ell_t(i) \\ &= W_t - \eta W_t \sum_{i=1}^{N} p_t(i)\ell_t(i) = W_t(1 - \hat{\ell}_t) \leqslant W_t \exp(-\hat{\ell}_t) \ . \end{aligned}$$

Thus,

$$W_{T+1} \leqslant W_1 \prod_{t=1}^{T} \exp(-\hat{\ell}_t) = N \exp(-\eta \hat{L}_T) \ .$$

Taking ln, we get

$$\hat{L}_T \leqslant \frac{1}{\eta}(\ln(N) - \ln(W_{t+1})) \tag{4}$$

Second part: Fix some expert i. Since weights are positive (and since ln is monotonically increasing), we have

$$\begin{aligned} \ln(W_{T+1}) &\geqslant \ln(w_{T+1}(i)) = \ln\left(w_1(i) \cdot \prod_{t=1}^{T}(1 - \eta\ell_t(i)) \right) \\ &= \ln(1) + \sum_{t=1}^{T} \ln(1 - \eta\ell_t(i)) \geqslant \sum_{t=1}^{T}(-\eta\ell_t(i) - \eta^2\ell_t(i)^2) \\ &\geqslant -\eta(L_T(i) + \eta T) \ , \end{aligned}$$

where the first inequality follows from the inequality

$$(\forall x \leqslant 1/2) \ln(1 - x) \geqslant -x - x^2 \ ,$$

and the second inequality follows from the fact that $\ell_t(i)^2 \leqslant 1$ for all t. Combining this bound on $\ln(W_{T+1})$ with Equation (4), we obtain

$$\hat{L}_T \leqslant \frac{1}{\eta}(\ln(N) + \eta(L_T(i) + \eta T)) = L_T(i) + \frac{\ln(N)}{\eta} + \eta T \ .$$

Note that the last inequality holds for every $i \in [N]$. Substituting $\eta = \sqrt{\ln(N)/T}$, we obtain that

$$R_T \leqslant 2\sqrt{T\ln(N)} \ .$$

Dividing by T, we get

$$\frac{R_T}{T} \leqslant 2\sqrt{\frac{\ln(N)}{T}} \ .$$

■

One can show that the regret bound above is asymptotically optimal, i.e., this is the minimax regret for this setting.

2.4 Online-to-Batch

We argued before that the online setting is more challenging than the batch setting. We now formalize this idea by showing a reduction from the batch setting to the online setting.

Thus, we consider the batch (PAC) setting in which the sequence $(x_1, y_1), \dots, (x_T, y_T)$ are generated i.i.d. according to some distribution $\mathcal{D}$. The idea is to run an online algorithm $\mathcal{B}$ that produces a sequence of hypotheses, $\hat{h}_1, \dots, \hat{h}_T$. For the reduction, we require that the choice of $\hat{h}_t$ does not depend on (x_t, y_t) (which is indeed the case for all the algorithms discussed above). Based on its decisions, we decide on one hypothesis $\hat{h}$. Maybe the most simple rule is to choose one of the hypotheses $\hat{h}_1, \dots, \hat{h}_T$ uniformly at random. We thus described an algorithm $\mathcal{A}$ for the batch setting that uses the online algorithm $\mathcal{B}$ as a black-box. We next relate the sample complexity of $\mathcal{A}$ to the regret of $\mathcal{B}$. For simplicity (and also since we already know how to boost the confidence), we discuss sample complexity only in terms of the accuracy parameter ϵ. That is, $m_{\mathcal{A}}(\epsilon)$ tells us how many examples are need in order to ensure that $\mathbb{E}L_{\mathcal{D}}(\hat{h}) \leqslant \inf_{h\in\mathcal{H}} L_{\mathcal{D}}(h) + \epsilon$.

Theorem 5 *Denoting the regret of $\mathcal{B}$ by R_T (and assuming that $\frac{R_T}{T} \to 0$ as T tends to infinity), we have*

$$\mathbb{E}_{\mathcal{D}}(\hat{h}) - \inf_{h\in\mathcal{H}} L_{\mathcal{D}}(h) \leqslant \frac{R_T}{T} \ .$$

Proof (sketch) The main idea is that since $\hat{h}_t$ is independent of (x_t, y_t), and since the sequence is i.i.d., the loss $\hat{\ell}_t$ forms an unbiased estimate of $L_{\mathcal{D}}(\hat{h}_t)$. The rest of the proof is left as an exercise. ■

For example, the MW algorithm attains an average regret of $O(\sqrt{T\ln N})$. The above theorem tells us that in the statistical model, if we run MW for T rounds, then the expected suboptimality, $\mathbb{E}_{\mathcal{D}}(\hat{h}) - \inf_{h\in\mathcal{H}} L_{\mathcal{D}}(h)$, of the algorithm $\mathcal{A}$ (which is equipped with MW as a black-box) is at most $\sqrt{\ln(N)/T}$. Thus, in order to ensure that this term is at most ϵ, we need to perform $T \geqslant \ln(N)/\epsilon^2$ iterations. This coincides with the bounds we know for agnostic PAC learning.

2.5 Online Perceptron

Originally, the Perceptron algorithm has been presented as an online algorithm. Namely, the algorithm maintains a weight vector $w_t \in \mathbb{R}^d$ which is used to produce a prediction $\hat{y}_t$

according to

$$\hat{y}_t = \mathrm{sgn}(\langle w_t, x_t \rangle) \ .$$

In the batch setting, we established two possible bounds on the complexity of linear separation using halfspaces. In the PAC model, where we make no assumptions on the distribution, we concluded a bound on the sample complexity that scales with the VC-dimension, d. If we assume that the data is separated with margin γ, then the number of required examples scales with $1/\gamma^2$ (that might be much smaller than d).

In the online setting we have a direct analogue result for the second case (i.e., the online percepron makes at most $1/\gamma^2$ mistakes if the margin is at least γ) but for the first case (in which we do not assume that the margin is large), as we show next, for any (deterministic) algorithm, we can generate a **realizable** sequence for which the algorithm errs on every round. Thus the regret of the algorithm w.r.t. the zero-one loss is T. The idea is very simple. Since thresholds on the line can be reduced to learning homogeneous halfspaces in $\mathbb{R}^2$ (by adding a bias coordinate), we describe an instance of of the thresholds problem for which the learner errs on every round.

1. Let $x_1 = 1/2$.
   (a) If $\hat{y}_1 = 1$, then choose $y_1 = -1$, $x_2 = 3/4$.
      i. If $\hat{y}_2 = 1$, then choose $y_2 = -1$, $x_3 = 7/8$.
      ii. If $\hat{y}_2 = -1$, then choose $y_2 = 1$, $x_3 = 5/8$.
   (b) If $\hat{y}_1 = -1$, then choose $y_1 = 1$, $x_2 = 1/4$.
      i. If $\hat{y}_2 = 1$, then choose $y_2 = -1$, $x_3 = 3/8$.
      ii. If $\hat{y}_2 = -1$, then choose $y_2 = 1$, $x_3 = 1/8$.
2. Proceed in the same manner. That is, at each time t, decide on y_t and x_{t+1} such that the following hold:
   - $y_t \neq \hat{y}_t$
   - There are two consistent ways to label x_{t+1}.

3 Online Convex Optimization

Similarly to the the batch model, we can discuss online convex optimization problems in which the hypothesis set forms a convex set and the loss functions at each time t are convex. We refer to a recent survey ([2]) on this topic.

Optional Exercises

See the exercises in [3][Chapter 21]. In particular, exercise 21.4 shows how to avoid the assumption that the algorithm knows the number of rounds, T. Exercise 21.5 discusses the Online-to-batch conversion.

References

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[2] S. Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2011.

[3] Shai Shalev-Shwartz and Shai Ben-David. Understanding machine learning. 2014.