Statistical Techniques in Robotics (16-831, F10) Lecture#10 (Thursday September 23)

General Weighted Majority, Online Learning as Online Optimization

Lecturer: Drew Bagnell *Scribe:Anca Drăgan* [1]

1 General Loss Function Weighted Majority

1.1 Recap

In the previous lectures, we have developed on the rather philosophical statement that induction is a provably near-optimal strategy. More precisely, the predictions as a function of time come near the optimal strategy.

What this means in fact is not that we can prove any bounds on the difference between reality and our predictions (loss), but rather that we can make the regret of our predictions small enough: we can make the difference between our loss and the loss of the best expert (in retrospect) small.

$$R = \sum_t l_t(alg) - l_t(e^*) \tag{1}$$

To find such an algorithm with small regret, we have introduced "Weighted Majority", a method that penalizes experts that made incorrect predictions. By a simple analysis, we were able to show that the number of mistakes this method makes is bounded by:

$$m \leq 2.4(m^* + \log N) \tag{2}$$

where N is the number of experts and m^* is the number of mistakes the best expert makes over the entire time horizon.

1.2 Weighted Majority with a General Loss Function

The classical "Weighted Majority" mentioned above uses a binary loss function: $l \in \{0, 1\}$. What follows is a strict generalization of this, where we allow a more flexible form of the loss function.

1. Set $w_i^0 = 0$.
2. At time t, predict expert e_i in proportion to its weight, w_i.
3. Adjust the weight of experts: $w_i^{t+1} = w_i^t e^{-\epsilon l_t(e_i)}, \forall i$, with $l_t(i) \in \{0, 1\}$

This algorithm has many applications: computational geometry, Adaboost (where the experts are the data points), and even Yao's XOR lemma (see [2] for more details) in complexity theory.

[1] Some content adapted from previous scribes: Jared Goerner.

The bound on the regret of this algorithm becomes

$$E[R] \leq \epsilon \sum_t l_t(e^*) + \frac{1}{\epsilon} \ln N \tag{3}$$

Ideally, we would like this algorithm to be what is called "No Regret", defined as the average regret over time converging to 0:

$$\frac{R_T}{T} \to 0 \tag{4}$$

To do so, we need to make sure that $\epsilon(T)$ decays in such a way that the regret grows less than linearly as a function of T. Since $l(e^*) \leq 1$ by definition, we have that $\sum_t l_t(e^*) \in O(T)$. Therefore, applying this to (3), we get:

$$E[R] \in O(\epsilon T + \frac{1}{\epsilon} \ln N) \tag{5}$$

Setting $\epsilon = \frac{1}{\sqrt{T}}$, we get that

$$E[R] \in O(\sqrt{T} + \sqrt{T} \ln N) \tag{6}$$

Not only does this grow less than linearly as required by (4) for obtaining No Regret, but it also makes the ration approach 0 the fastest (up to constants).

Note: *This is the point where Drew says how this algorithm can solve any problem in the universe. For more information on General Weighted Majority, refer to the original paper by Arora et.al[1].*

1.3 Example: Linear Classifiers

The general form of a binary linear classifier is

$$\theta^T f \geq 0 \Rightarrow 1$$

$$\theta^T f < 0 \Rightarrow -1$$

Here, f is a feature vector and θ is the weight vector. Like any other problem, we can come up with a weight vector using Weighted Majority: simply set the experts to be sampled θs, and the loss to be whether the linear classifier using θ as a weight vector made the right binary prediction. The time steps in Weighted Majority become the data points.

The problem with this approach is that we need to discretize this n-dimensional space (where n is the number of features), which requires having $O(e^n)$ experts. This makes the algorithm very impractical.

On the other hand, the approach allows us to derive useful bounds on linear classifiers, by looking at the scaling in terms of the number of features. Plugging in the number of experts into (3), we get that

$$E[R] \leq \epsilon \sum_t l_t(e^*) + \frac{1}{\epsilon} \ln e^n = \sum_t l_t(e^*) + \frac{1}{\epsilon} n \tag{7}$$

Therefore, the regret scales linearly in the dimension of the feature space. It turns out it is generally true that we need $O(n)$ samples for a linear classifier. (When introducing the constant it becomes about $10n$).

2 Online Learning as Online Optimization

Motivation: we can solve convex problems efficiently.

2.1 What are Convex Sets and Functions?

A convex set is a set such that any linear combination of two points in the set is also in the set: if $A \in C, B \in C$, then

$$\theta A + (1 - \theta)B \in C \tag{8}$$

For example, a disc in 2D is convex, but a circle isn't. Likewise, a star-shaped set would not be convex, since if you picked a point in one star tip and a point in another star tip, not all of the linear combinations of those two points would be in the set. An interesting property of convex sets is that their intersection is convex.

Convex functions are the functions for which the epigraph (the area above the curve) is a convex set. Intuitively, you can think about it as if the function will hold water if it was poured in from the top of the graph. The property corresponding to (8) for functions is:

$$f(A)\theta + (1 - \theta f(B) \geq f(\theta A + (1 - \theta)B) \tag{9}$$

This directly generalizes to Jensen's inequality:

$$if \sum_i \theta_i = 1$$

$$then\ f\left(\theta_1 x_1 + \ \cdots + \ \theta_n x_n\right) \leq \ \sum_i \theta_i f(x_i)$$

2.2 Subgradients

Convex functions have subgradients at every point in their domain. A subgradient $\nabla f(x)$ is a subgradient at x if it is a line s.t. the entire function is above it:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x), \forall y \tag{10}$$

This property simply re-states that any point in the domain of the function, the value of the function at that point needs to be higher than the value of the subgradient line at the same point.

If a function is differentiable at a point, then it has a unique subgradient at that point. Furthermore, convex functions are the max over all subgradients. This is an interesting property that will be used later in the class, because the maximum of convex functions is convex. This is an equivalent statement to the one about the intersection of convex sets, because the maximum of two functions is a function whose epigraph is the intersection of the epigraphs of the two original functions.

2.3 The Online Convex Programming Problem - Intro

Online Convex Programming was proposed by Martin Zinkevich[3] in 2003. It is framed in the same context of time steps, loss function, experts and weighted majority, preserving all the same qualities form WM, while being computationally feasible.

The idea is that the experts are elements of some convex set, and that the loss at time t is convex over the set of experts and thus has a subgradient. At every time step, we need to predict an expert x_t and receive the loss $l_t(x_t)$ and $\nabla l_t(x_t)$.

Example: for the case of the linear classifier, where the experts are $x_t = \theta_t$ in some convex set, the loss function could be

$$l_t(\theta_t) = (\theta_t^T f_t - y_t)^2$$

where y_t is the actual label for the data point f_t, in $\{-1, 1\}$. This loss is convex and is in fact a parabola in terms of θ_t.

The next lecture will formalize the Online Convex Programming Problem better, and explain its applications to No Regret Portfolio creation.

References

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale, "The multiplicative weights update method: A meta algorithm and its applications." Technical report, The Princeton University

[2] O Goldreich, N Nisan, A Wigderson, "On Yao's XOR-lemma"

[3] Martin Zinkevich, "Online Convex Programming and Generalized In nitesimal Gradient Ascent", ICML 2003