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

General Weighted Majority, Online Learning as Online Optimization

Lecturer: Drew Bagnell *Scribe: Nathaniel Barshay*[1]

1 Generalized Weighted majority

1.1 Recap

In general online learning, we cannot hope to make guarentees about loss in any absolute sense. Instead, we use the notion of regret R to compare the loss of our algorithm (alg) against that of the best expert ($e^\star$) from some family of experts. We thus define regret as:

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

The first of such algorithms analyzed was "Weighted Majoriy", which works on 0/1 loss, and achieves a number of mistakes m 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 in retrospect makes over the entire time horizon.

Next we looked at "Randomized Weighted Majority", which is similar to WM but uses a weighted draw (rather then a weighted average), to make a prediction at a given timestep, and also introduces a learning rate β.

1.2 The Master Learning Algorithm: Generalized Weighted Majority

The RWM algorithm mentioned above assumes a binary loss function: $l \in \{0, 1\}$. We now generalize to any loss function with outputs in $[0, 1]$ (keeping RWM as a special case). The algorithm is:

1. Set $w_i^0 = 1$.
2. At time t, pick an expert e_i in proportion to its weight w_i, and let that expert decide.
3. Adjust the expert weights:

$$w_i^{t+1} \longleftarrow w_i^t e^{-\epsilon l_t(e_i)} \qquad \forall i$$

The bound on the regret of this algorithm becomes

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

[1]Content adapted from previous scribes: Anca Drăgan, Jared Goerner.

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

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

This grow sublinearly in T, thus the ratio in (4) tends towards $\frac{1}{\sqrt{T}}$, and we have shown a no regert algorithm.

Of course, this requires knowing T beforehand, it turns out one can also achieve no regret (via a harder proof) by varying ϵ with time:

$$\epsilon_t = \frac{1}{\sqrt{t}}$$

.

Note: *This is the point where Drew says that 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].*

This algorithm has many suprising 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.

1.3 Application: Self Superfized Learning

Suppose we have a robot (driving in 1-dimension) that wants to learn to identify objects at long range, given that it can identify objects perfectly at short range. Such a sensor model is quite common, we have far less information (and thus classification is far more difficult) when objects are far away. It might be desirable to learn such a model in a path planning setting.

Let us assume that every observed obstacle is either a Tree or a Giant Carrot (and thus the difficulty of classification is quite understandable). The formal online learning is as follows: we get features (from an object at range) and decide a class (Tree/Carrot) from the features available, then we drive close to the object and the world gives us the true classification. We will use 0/1 Loss (0 if correct, 1 if incorrect). Almost any family of classifiers can be used for our set of experts (decision trees, linear classfier). However, we must discretize the parameters of such learners to keep the number of experts finite.

1.4 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 vector of weights. Note that if we assert $||\theta|| = 1$, then theta essentially has only $d-1$ parameters, since the last is redundant (and each $\theta_i \in [-1, 1]$).

In order to have a finite family of experts, we might discretize each θ_i into b levels, and have an expert for each combinations of θ_i. In this case we have $N = (b-1)^d$, and we can run GWM verbatim.

Plugging the number of experts into (3), we get that:

$$E[R] \leq \epsilon \sum_t l_t(e^*) + \frac{1}{\epsilon} \ln\left(O(b^d)\right) = \epsilon \sum_t l_t(\theta^*) + \frac{1}{e} O(d \ln(b)) \tag{7}$$

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 $10n$).

This is great theoretically, but keep in mind we still need to track weights for each of $O(b^d)$ experts! Thus the algorithm is only practical for small d (upper bound at about 4).

2 Online Learning as Online Optimization

Motivation: "In fact, the great watershed in optimization isn't between linear and nonlinear, but between convexity and nonconvexity"

Most importantly, if we use:

1. Convex sets of experts
2. Convex loss functions

then we may be able to solve our online learning problem efficiently in the realm of online optimization.

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, the perimiter of a circle is not convex, because a linear combiation of two points is a chord, which passes through the enterior (which is not in the set). Examples of convex sets include:

- Unit ball in $\mathbb{R}^n$ under l_2 norm. $S = \{||x|| \leq R\}$
- Box in $\mathbb{R}^n$. $S = \{||x||_\infty \leq R\}$
- General unit ball. $S = \{||x||_i \leq R \qquad i \geq 1\}$
- Linear subspace
- Half space. $S = \{w_t x \leq b\}$
- Intersection of half spaces, I.E. polyhedron.
- Cone, I.E. all positive linear combinations of a set of vectors.

Convex functions are the functions for which the epigraph (the area above the curve) is a convex set. Defined rigerously we have:

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

This directly generalizes to Jensen's inequality:

$$\sum_i \theta_i = 1 \quad \implies \quad f\left(\theta_1 x_1 + \ \cdots + \ \theta_n x_n\right) \leq \ \sum_i \theta_i f(x_i)$$

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 the normal to some plane that touches $f(x)$ at x, and is below the rest of f. In symbols:

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

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.

Several key properties of convex functions follow:

- Any local minima is also a global minimum (not necessarily the unique global minimum). This is easy to see, the subgradient at a local minima sets a lower bound on the functions value.
- Local optimization never gets stuck (we can always follow a subgradient down, unless already at a global min)

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