Statistical Techniques in Robotics (16-831, F12) Lecture#10 (Monday October 1)

Convex Optimization and Subgradients

Lecturer: Drew Bagnell *Scribe: Shervin Javdani*[1]

1 Recap - Randomized Weighted Majority

In general online learning, we cannot hope to make guarantees 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^*) from some family of experts. Thus, we define regret as:

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

The first of such algorithms analyzed was "Weighted Majority", 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 makes a prediction by randomly sampling one expert, with probability proportional to how correct that expert was in the past (instead of a weighted average). We introduced a learning rate β, which corresponds to how much we downweight experts for a mistake. As we are now making decisions probabilistically, we provide a bound on the expected number of mistakes:

$$\mathbb{E}[m] \leq \frac{m^* \ln(1/\beta) + \ln(n)}{1 - \beta} \tag{3}$$

where n is the number of experts.

2 Generalized Weighted Majority

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

1. Set $w_i^0 = 1$.
2. At time t, choose expert e_i with probability $\frac{w_i^t}{\sum_j w_j^t}$
3. Adjust all expert weights:
$$w_i^{t+1} \longleftarrow w_i^t e^{-\epsilon \ell_t(e_i)} \qquad \forall i$$

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

Where ϵ is some scaler (possibly different for each time step) which determines how quickly weights are adjusted based on the loss function. The bound on the regret of this algorithm becomes:

$$\mathbb{E}[R] \leq \epsilon \sum_t \ell_t(e^*) + \frac{1}{\epsilon} \ln N \tag{4}$$

Note that we can affect our bound based on how we select ϵ. Ideally, we would like this algorithm to be what is called "No Regret", defined as the average regret over time converging to 0:

$$\frac{R_T}{T} \to 0 \tag{5}$$

To do so, we need to select ϵ_t so that the regret grows sublinearly as a function of T. Since $\ell(e^*) \leq 1$ by definition, we have that $\sum_t \ell_t(e^*) \in O(T)$. Therefore, applying this to (4), we get:

$$\mathbb{E}[R] \in O(\epsilon T + \frac{1}{\epsilon} \ln N) \tag{6}$$

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

$$\mathbb{E}[R] \in O(\sqrt{T} + \sqrt{T} \ln N) \tag{7}$$

Which grows sublinearly in T. Thus the ratio in (5) tends towards $\frac{1}{\sqrt{T}}$, and we have shown this is a no regert algorithm!

Unfortunately, this requires knowing T beforehand. However, 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 surprising 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.

2.1 Application: Self Supervised Learning

Suppose we have a robot (driving in 1-dimension) that can accurately identify objects at short range, and wants to learn how to identify them at long range. Let us assume that every observed obstacle is either a Boulder or a Bush. The formal online learning is as follows: we get features (from an object at range) and decide a class (Boulder/Bush) from the features available When we drive close to the object, we get the class, and can thus update our algorithm for predicting at long range. We will use 0/1 Loss (0 if correct, 1 if incorrect). See Figure 1. But what is the "expert" in this case?

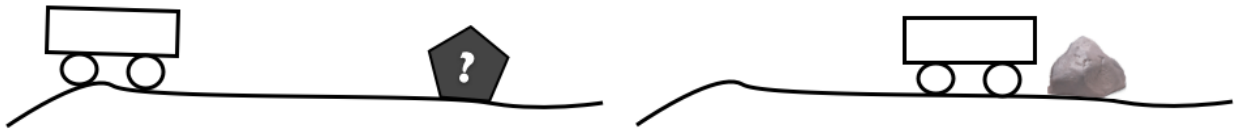

Figure 1: Self Supervised learning example. Far away, we would like to predict whether an object is a boulder or bush. After driving up close, we receive the object's true class. We formulate the construction of the long range predictor as an online learning problem, where our total loss is the number of times the long range predictor misclassified.

2.1.1 Linear Classifiers

We consider using a linear classifier, where our expert θ is essentially how we weight each feature in making our decision. That is:

$$\begin{aligned} \theta^T f \geq 0 \quad &\Rightarrow 1 \text{ (boulder)} \\ \theta^T f < 0 \quad &\Rightarrow 0 \text{ (bush)} \end{aligned}$$

Where f is feature vector with d features (such as height, spread, etc.). Note that there are infinitely many experts $\theta \in \mathbb{R}^d$. To deal with this, we discretize θ.

(sidenote - it's easy to see that the length of θ doesn't matter - only the direction does. We also assert that each $\theta_i \in [-1, 1]$.)

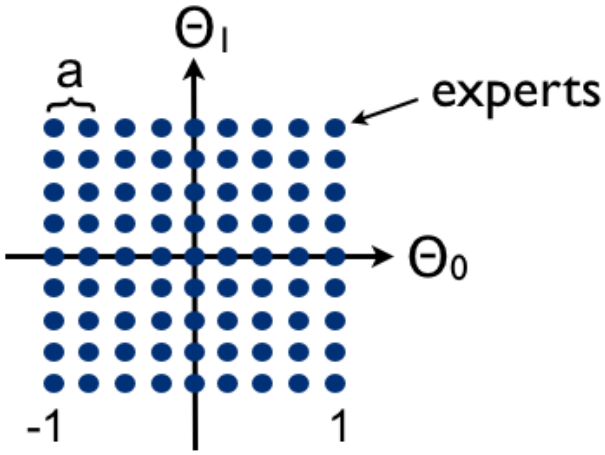

Figure 2: A discretized linear classifier with $d = 2$, with interval length a.

Consider discretizing our features at intervals of a for each θ_i, as in Figure 2. This results in $(\frac{2}{a}+1)^d$ experts. (we can actually do a little better by noting there is some redundancy in θ, since only the direction matters...). Plugging in this number of experts in our regret bound (4), we get:

$$\mathbb{E}[R] \leq \epsilon \sum_t \ell_t(\theta^*) + \frac{1}{\epsilon} \ln \left((\frac{2}{a} + 1)^d \right) \tag{8}$$

$$= \epsilon \sum_t \ell_t(\theta^*) + \frac{d}{\epsilon} \ln \left(\frac{2}{a} + 1 \right) \tag{9}$$

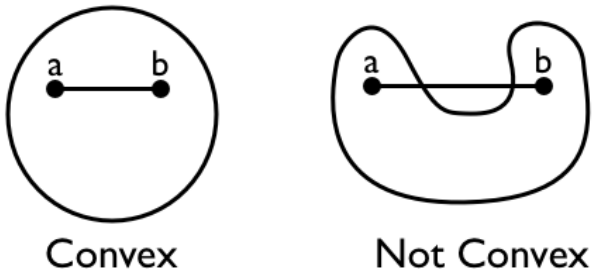

Figure 3: Example of a convex and non-convex set. If we take any two points inside our set, and draw a line in between them, convexity requires that every point along that line is also in that set.

So how many examples will we need before we do a good job predicting at long range? We should expect to need $O(d)$ samples for a linear classifier (Drew says about $10d$)

We see that our regret is linear in d. However, we need to keep track of weights for all $(\frac{2}{a}+1)^d$ experts! Thus the algorithm is actually exponential in d. We'll fix this for certain types of problems - specifically, if our problem is convex.

3 Online Learning with Convexity

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, even if we have an infinite number of experts.

3.1 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, as in Figure 3. Mathematically, this is equivalent to saying that if $A \in C, B \in C$, then

$$\theta A + (1-\theta)B \in C \tag{10}$$

For example, the perimeter of a circle is not convex, because a linear combination of two points is a chord, which passes through the interior (which is not in the set). Examples of convex sets include:

- Box in $\mathbb{R}^n$. $S = \{||x||_\infty \leq R\}$
- General unit ball. $S = \{||x||_i \leq R \qquad i \geq 1\}$
- Linear subspace

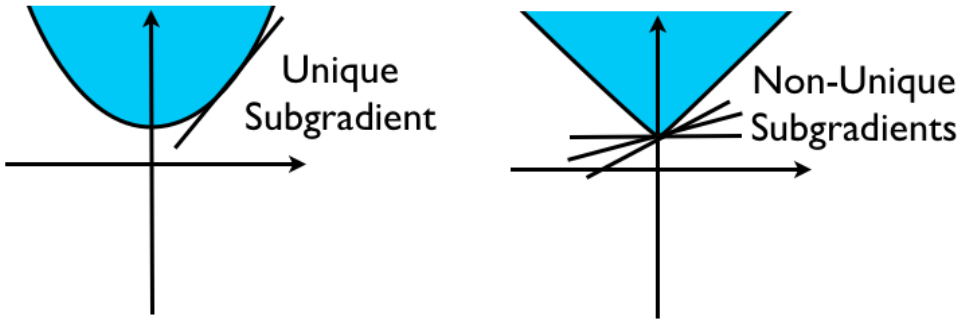

Figure 4: Examples of convex functions and their subgradients. For differentiable points, we have only one subgradient, equivalent to the gradient. For non-differentiable points, we see here infinitely many subgradients.

- Half space. $S = \{w_t x \leq b\}$
- Intersection of any convex sets
- 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 rigorously we have:

$$\theta f(A) + (1-\theta) f(B) \geq f(\theta A + (1-\theta) B) \tag{11}$$

Several key properties of convex functions follow:

- Any local minima is also a global minimum (not necessarily the unique global minimum).
- Local optimization never gets stuck (we can always follow a subgradient down, unless already at a global min)

3.2 Subgradients

We can consider a generalization of gradients for convex functions. We define a subgradient $\nabla f(x)$ at x as any vector that is normal to $f(x)$, and is below the rest of f. We write this as:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \qquad \forall y \tag{12}$$

Convex functions have subgradients at every point in their domain. If a function is differentiable at a point, then it has a unique subgradient at that point equivalent to the gradient. At other points, we may have infinitely many subgradients. See Figure 4.

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