Homework 2
CS 236r, Spring 2016

Due Wednesday, April 6, 11:59pm

In this homework, we'll look at a slightly more general online learning setting than learning from expert advice, called *online convex optimization.*

We'll define the setting as follows. We are given a hypothesis space $\mathbb{R}^d$ and some space of possible data points Z. We are also given a loss function $\ell : \mathbb{R}^d \times Z \to \mathbb{R}$, where $\ell(h, z)$ is the loss of hypothesis h on data point z. It is assumed that the loss function is convex in its first argument.

At each time $t = 1, \dots, T$, the algorithm must output a hypothesis $h \in \mathbb{R}^d$, then some data point z_t arrives and the algorithm suffers a loss $\ell(h_t, z_t)$. The *regret* of the algorithm is

$$\sum_{t=1}^{T} \ell(h_t, z_t) \; - \; \min_{h \in \mathbb{R}^d} \sum_{t=1}^{T} \ell(h, z_t).$$

The Follow-the-Regularized-Leader (FTRL) algorithm is as follows: for a convex "regularizer" function $G : \mathbb{R}^d \to \mathbb{R}$ and some η, choose

$$h_t = \arg\min_{h \in \mathbb{R}^d} \quad \sum_{s=1}^{t-1} \ell(h, z_s) \; + \; \frac{R(h)}{\eta}.$$

1 Example FTRL algorithms

1.1 Randomized weighted majority

Suppose that each data point is a vector $z_t \in \mathbb{R}^d$, that the hypothesis must be a probability distribution over support size d, and the loss is $\ell(h, z) = h \cdot z$ (the dot product). Some notation: let $h(i)$ be the ith coordinate of the vector h.

(i) Explain why this model of data points and this loss function is equivalent to the setting of "prediction from expert advice" described in lecture. (2pts)

Multiplicative weights, randomized weighted majority, and hedge are three names (and there are probably more) for essentially the same online learning algorithm, which appears broadly in computer science. This algorithm is as follows:

1. Set $h_0 = \left(\frac{1}{d}, \ldots, \frac{1}{d}\right)$.
2. At time t, for $i = 1, \ldots, d$, update the ith coordinate of the hypothesis as follows: set $h_t(i) \propto h_{t-1}(i)e^{-\eta z_{t-1}(i)}$. (That is, take each $h_t(i) = h_{t-1}(i)e^{-\eta z_{t-1}(i)}$, then renormalize.)

(ii) Show that this algorithm is a special case of FTRL with $R(h) = \sum_{i=1}^d h(i) \log h(i)$. (4pts)

1.2 Online gradient descent

Sometimes, running FTRL may be computationally annoying. This motivates the following simplified version of FTRL: replace each $\ell(h_s, z_s)$ by its subgradient at h_s. That is, define $r_s = \ell'(h_s, z_s)$ where ℓ' is the subgradient with respect to h_s. This gives what one might call "linearized" FTRL:

$$h_t = \arg\min_{h \in \mathbb{R}^d} \sum_{s=1}^{t-1} r_s \cdot h \; + \; \frac{R(h)}{\eta}.$$

It can be shown that this linearized FTRL still guarantees regret $O(\sqrt{T})$.

Consider the following algorithm, called online gradient descent:

1. Set $h_0 = 0$.
2. At time t, set $h_t = h_{t-1} - \eta r_{t-1}$ where r_{t-1} is defined as above.

Show that this algorithm is a special case of "linearized" FTRL with $R(h) = \frac{1}{2}\|h\|_2^2 = \sum_{i=1}^d h(i)^2$. (4pts)

2 Lower Bound on Regret for Online Learning

Consider the following problem of predicting from expert advice. There are two experts, Heads and Tails. Each day, one of them will receive loss 1 and the other will receive loss 0.

Show that, for every algorithm, there is some sequence of such losses such that the algorithm's regret is $\Omega(\sqrt{T})$. (10pts)

Here $\Omega(\sqrt{T})$ means "at least $c \cdot \sqrt{T}$ for some constant c". You do not need to find an exact constant.

(Hint: Come up with an adversary who chooses the sequence of losses in a random way, and argue that every algorithm has bad expected regret against this adversary.)

(Hint 2: Recall the Central Limit Theorem.)

3 Coding

We ask you to code up your favorite online learning algorithm (for instance, one of the versions of FTRL above) and run it on some data. We recommend you use a programming language such a python or R (it may be somewhat difficult in Matlab, or Excel...), but any choice of language is ok. We will study a *linear regression* problem:

- Hypotheses: $h \in \mathbb{R}^d$. We encourage you to try different d such as $2, 5, 10, 50, 100$.
- Data points: pairs (x, y) with $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$.
- Loss function: $\ell(h, (x, y)) = (y - h \cdot x)^2$, where $h \cdot x$ is the dot-product.

Generate data. (5pts) To simplify the task, rather than using a real data set, we're asking you to generate the data yourself. One suggested approach for generating interesting data is the following:

1. Pick a "true" hypothesis $h^* \in \mathbb{R}^d$.
2. Pick a "noise level" σ^2.
3. Draw x_i from some distribution on $\mathbb{R}^d$ – for instance, uniform on $[0, 1]^d$ – and set $y_i = h^* \cdot x_i + \mathcal{N}(0, \sigma^2)$ where h^* is the "true" hypothesis and $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2.

We encourage you to generate *lots* of data!

Please submit your code (not your data).

Implement an online learning algorithm and run it on your data. (10pts) You may find it interesting to try multiple learning algorithms (but implementing one is sufficient for full credit). While we encourage you to try an FTRL algorithm, *e.g.* Online Gradient Descent, you may also like to compare to some other learning algorithm you already know.

Please submit your code.

Test your algorithm(s) and visualize the results. (5pts) Ask an interesting question(s) about the performance of your algorithm and use simulations to investigate, plotting the results.

In particular, what is its average regret? Because it may be less tractable to calculate the best hypothesis in hindsight, if you generated data from *e.g.* the process described above, you may pretend that the true hypothesis h^* is the optimal one and consider regret of the algorithm from not using h^*. (If you have a large amount of data, almost certainly h^* will be very close to optimal.)

Now you may want to ask how this average regret varies as different parameters of the problem change:

- The number of data points, T.
- The dimensionality of the space, d.
- The "noisiness" of the data, σ^2.

We do not expect you to thoroughly investigate all three of these questions. You can receive full credit by picking just one and giving a brief story of how performance varies with that parameter, supported by a plot or two of your simulation results. You may wisth to generate multiple or perhaps a large number of data sets and use average performance across data sets to strengthen your argument.

Bonus: do an excellent job and/or go above and beyond on this problem. (2pts)