Algoritmos e Incerteza **(PUC-Rio INF2978, 2018.2)**

Lecture 06: Other metrics and Online Convex Optimization

Sep, 18^{th}, 2018

Lecturer: Marco Molinaro *Scribe: Leonardo de Almeida Matos Moraes*

There are two main topics in this class: other metrics for online algorithms and online convex optimization.

The following metrics for online algorithms are discussed: (i) **tracking regret** and (ii) **(strongly) adaptive regret**. The FixedShare algorithm, that represents an adaptation of the MWU algorithm, is also discussed.

Finally, the basic concepts of the Online Convex Optimization (OCO) field are presented – we end the lecture presenting two (simple) OCO applications: scalar prediction and portfolio optimization.

1 Recap

In the Abstract Experts Game, at each time step $t = 1, \cdots, T$, a certain algorithm Algo,

1. Generates a distribution $p \in \Delta^m$, where m represents available actions, or experts; and

2. Receives the loss vector $\ell^t \in [0; 1]^m$.

The loss, at each iteration t, is given by $\langle \ell^t, p^t \rangle$ or $\sum_i p_i^t \ell_i^t$.

The goal, in a minimization version of the problem, is to minimize the **total loss**, or

$$\text{(OBJ)} \quad \sum_t \langle p^t, \ell^t \rangle \quad \leq \quad \min_i \sum_{t=1}^{T} \ell_i^t. \tag{1}$$

It was also proved in the last lecture that MWU algorithm has the following bound:

$$\sum_{t=1}^{T} \langle p^t, \ell^t \rangle \quad \tilde{\leq} \quad \min_i \sum_{t=1}^{T} \ell_i^t + \sqrt{T \cdot \log m}. \tag{2}$$

2 Other performance metrics

So far, we have seen metrics that compare the solution obtained by an online algorithm to the ”best action”, i.e., fixed solutions, valid to every time step t. In this section one will touch on different performance metrics.

2.1 Tracking regret

An initial idea could be to compare the solution obtained by the algorithm to the best solution found at each time step t. However, this idea will be proved to be too ambitious. The following example serves an illustration of this.

Example: Let's consider $m = 2$ and the loss function $\ell^t = (0,1)$ or $(1,0)$. Let's build a bad instance for this problem. This can be done by using the following loss function:

$$\ell^t \quad = \quad \begin{cases} (1,0) & \text{, if } p_1^t \geq p_2^t, \\ (0,1) & \text{, if } p_1^t < p_2^t. \end{cases} \tag{3}$$

In this case, at each time step t, the loss will be, at least, equal to 0.5. In mathematical terms, $\langle p^t, \ell^t \rangle \geq \frac{1}{2}, \forall t$. So, total loss will be $\geq T/2$, while the dynamic OPT has total loss 0 (zero).

This example shows how ambitious it is to compare an algorithm solution with the dynamic optimal solution. So, if we are not going to compare the algorithm solution with the dynamic OPT, it will be compared with an OPT obtained if solution is allowed to change k times.

Example: Let the following table represent the loss function for t time steps for each available action (0,1,2).

Actions	1	...	T/4	T/4+1	...	T/2	T/2 + 1	...	3T/4	3T/4 + 1	...	T
1	0	...	0	0	...	0	1	...	1	1	...	1
2	0	...	0	1	...	1	1	...	1	0	...	0
3	1	...	1	1	...	1	0	...	0	1	...	1

If $k = 0$, i.e., one can't change the option during the game, $0-$OPT$=T/2$ (action 1, for instance). Moreover,

$$\begin{array}{lll} k = 0, & 0-\text{OPT} = \frac{T}{2} & (\text{action 1 for } t = 1, \cdots, T) \\ k = 1, & 1-\text{OPT} = \frac{T}{4} & (\text{action 1 for } t = 1, \cdots, \frac{T}{2}, \\ & & \text{action 2 for } t = \frac{T}{2} + 1, \cdots, T) \\ k = 2, & 2-\text{OPT} = 0 & (\text{action 1 for } t = 1, \cdots, \frac{T}{2}, \\ & & \text{action 3 for } t = \frac{T}{2} + 1, \cdots, \frac{3T}{4}, \\ & & \text{action 2 for } t = \frac{3T}{4} + 1, \cdots, T). \end{array}$$

This brings us to the definition of the Tracking Regret metric.

Definition 2.1 (Tracking Regret)**.** The k-tracking regret of an algorithm Algo is defined as:

$$\sum_t \langle p^t, \ell^t \rangle - \left[\begin{array}{c} \text{loss of the} \\ \text{best sequence} \\ \text{of actions with} \\ \text{up to } k \text{ changes} \end{array} \right]$$

Question: is it possible to use the MWU algorithm to obtain a good algorithm with respect to the tracking regret?

To build such an algorithm, one can

1. build meta-actions for each possible sequence of actions; and
2. run MWU in meta-actions.

A meta-action represents a "path" of actions. An example of meta-actions for $m = 2$ and $k = 1$ is

$$\begin{array}{cccccc} a_1 & \cdots & \cdots & \cdots & \cdots & a_1 \\ a_1 & \cdots & \cdots & \cdots & a_1 & a_2 \\ a_1 & \cdots & a_1 & a_2 & \cdots & a_2 \\ \vdots & & \vdots & \vdots & & \vdots \\ a_2 & \cdots & a_2 & a_1 & \cdots & a_1 \\ \vdots & & \vdots & \vdots & & \vdots \end{array},$$

i.e., the first meta-action plays action a_1 at all times, the second plays a_1 until the last time, when plays a_2, etc.

In this case, MWU algorithm run over the meta-actions has the following bound:

$$\text{loss} \quad \tilde{\leq} \quad k - \text{OPT} + \sqrt{T \cdot \log(ma)}, \tag{4}$$

where ma represents the number of meta-actions (paths). So, in order to evaluate this bound, one needs to find, for a given k, how many meta-actions exist.

One can change the solution up to k times. So, there are $\binom{T}{0} + \binom{T}{1} + \cdots + \binom{T}{k}$ ways to build the path. So,

$$\begin{aligned} ma &\leq \left[\binom{T}{0} + \binom{T}{1} + \cdots + \binom{T}{k}\right] \cdot m^k \\ &\leq k\binom{T}{k} m^k \\ &\tilde{\leq} (Tm)^k. \end{aligned}$$

Using this result, $\sqrt{T \cdot \log(ma)} = \sqrt{Tk \cdot \log(Tm)}$.

Lemma 2.2 (MWU bound)**.** *One can use MWU algorithm to obtain a solution* p *such that*

$$\sum_t \langle p^t, \ell^t \rangle \quad \tilde{\leq} \quad k - \textit{OPT} + \sqrt{Tk \cdot \log(Tm)}.$$

When a "big" k is used, the $k-$OPT term tends to 0, while the term $\sqrt{Tk \cdot \log(Tm)}$ grows wrt k. So, it should be desirable to find an "optimal" value for k (a parameter of the algorithm), in order to minimize the loss, though it is not clear how to do this, since $k-$OPT is unknown.

Another issue related to the use of MWU with meta-actions is the computational cost of creating and maintaining in memory all the $(Tm)^k$ meta-experts. So, it is important to find a more efficient algorithm.

Question: Is there any way one can use the "vanilla" MWU algorithm? The following example ($k = 1$) presents its performance analysis. Let the loss function be represented by the following table:

Actions	1	...	T/2	T/2+1	...	T
1	0	...	0	1	...	1
2	1	...	1	0	...	0

Let's use MWU with $\varepsilon = 1/2$. In this case, MWU update rule is

$$w_i^{t+1} = w_i^t e^{-\frac{1}{2}\ell_i^t}, \quad p_i^{t+1} = \frac{w_i^{t+1}}{\sum_i w_i^{t+1}}.$$

Thus, for $t = \frac{T}{2}$,

$$\begin{cases} w_1^{T/2} = w_1^1 = 1, \\ w_2^{T/2} = w_2^1 \cdot e^{-T/4} = e^{-T/4} \approx 0. \end{cases}$$

So, $p^{T/2} \approx (1 - e^{-T/4}, e^{-T/4})$.

For time steps $t > T/2$ one wants MWU algorithm to result in $p_2 \to 1$. However, it would take an excessive amount of time to achieve this goal, because, at time step $T/2$, current weight of p_2 is close to zero. After T iterations:

$$\begin{cases} w_1^T \approx e^{-T/4} \\ w_2^T \approx e^{-T/4}, \end{cases}$$

i.e., at the end of the game, both actions are equal weighted.

At every time step t, $p_1 \geq 1/2$. So, this algorithm has a total loss $\geq T/2$. However, $1-$OPT is equal to 0 (zero). So, the "vanilla" MWU algorithm shouldn't be used (bad performance, when compared to dynamic optimal).

This previous example showed how important the historical information is to the MWU algorithm (which gives too much inertia). So, one can suggest to reinitialize the MWU algorithm after a given number of iterations (e.g. 5 time steps), in order to reduce the effect of the memory into the process.

In this case,

$$\begin{array}{lrcl} & \text{window } j \text{ loss } j & \tilde{\leq} & \text{OPT}_j + \sqrt{\log m} \\ (\sum) & \text{total loss} & \tilde{\leq} & \text{OPT} + \frac{T}{5}\sqrt{\cdots}, \end{array}$$

so the loss continues to have a linear relation wrt T. Then, using this windows-approach (e.g. 5 time-steps windows) isn't a good idea.

[FixedShare Algorithm]: This algorithm is like MWU, but with a different update rule ([3]):

$$p_i^{t+1} = \frac{\alpha}{m} + (1-\alpha)\frac{p_i^t e^{-\varepsilon \ell_i^t}}{\sum_j p_j^t e^{-\varepsilon \ell_j^t}}, \tag{5}$$

that represents a linear combination among uniform discrete distribution and the traditional MWU update rule.

One should pay attention to a few comments about the FixedShare algorithm:

- when $\alpha = 1$, algorithm maintains the uniform discrete distribution at all iterations;
- when $\alpha = 0$, one has the MWU, with full memory;
- the term α/m represents a lower bound to the weights of every expert, at all the iterations.

The third point of the above list is important because, if every expert has, at each iteration, a lower bound for its weight, the algorithm has enough time to change weights allocation, if necessary.

Theorem 2.3 (FixedShare $k-$Regret)**.** *FixedShare algorithm with $\alpha = k/T$ has $k-$tracking regret limited to:*

$$k - \textit{Tracking Regret} \tilde{\lesssim} \sqrt{Tk \cdot \log(Tm)}. \tag{6}$$

2.2 (Strongly) adaptive regret

The idea behind this performance metric is to identify (measure) algorithms with small regrets at any time interval through $1, \cdots, T$. Let $\mathcal{I}$ be an interval (window), such that $\mathcal{I} \subseteq [1, \cdots, T]$. Then, the adaptive regret $\mathcal{R}$ is such that

$$\sum_{t \in \mathcal{I}} \langle p^t, \ell^t \rangle \leq \sum_{t \in \mathcal{I}} \ell^t_{i^*(\mathcal{I})} + \mathcal{R}, \quad \forall \mathcal{I}, \tag{7}$$

where $i^*(\mathcal{I})$ represents the "best" action for interval $\mathcal{I}$.

It is important to note that a good 2-Tracking Regret algorithm doesn't have, necessarily, a good performance with respect to the (strongly) Adaptive Regret measure. However, a good Adaptive Regret algorithm must have a good k-Tracking Regret: in a 2015 paper, Daniely et. al. ([2]) proved that strongly adaptive regret is stronger than tracking regret, i.e., a Strongly Adaptive Regret algorithm satisfies k-Tracking Regret, $\forall k$.

Adamaskiy et. al. have proved ([1]) that FixedShare algorithm satisfies the following:

$$\text{loss interval } \mathcal{I} \quad \tilde{\lesssim} \quad \text{OPT}(\mathcal{I}) + \sqrt{|\mathcal{I}| \cdot \log(mT)}, \quad \forall \mathcal{I}. \tag{8}$$

3 Online Convex Optimization (OCO)

3.1 Basic definitions

Definition 3.1 (Convex Functions)**.** A function $f : \mathbb{R} \to \mathbb{R}$ is said to be convex if

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle, \quad \forall x, y \in \mathbb{R}.$$

Examples: the following functions are convex:

$$\begin{aligned}
&f(x) = x^2, \\
&f(x) = |x|, \\
&f(x_1, \cdots, x_n) = x_1^2 + \cdots + x_n^2, \\
&f(x_1, \cdots, x_n) = \textstyle\sum_i a_i x_i = \langle a, x \rangle, \\
&f(x) = \max\{\langle a^1, x \rangle + b_1, \langle a^2, x \rangle + b_2\}.
\end{aligned}$$

The most important characteristic of convex functions, when one is studying optimization of such functions, is that any local minimum is a global minimum. Moreover, this minimum point can be found (approximately) in polynomial time.

3.2 Online Convex Optimization algorithms

The goal of an OCO algorithm is to find, for each time step $t = 1, \cdots, T$, $x^t \in \mathcal{P}$ (where $\mathcal{P} \subseteq \mathbb{R}^n$ is called *playing set*), in order to minimize total loss (see Algorithm 1). The total loss function $F(x)$ is given by

$$F(x) = \sum_{t=1}^{T} f_t(x).$$

One must note that $F(\cdot)$ is a convex function, because it is defined as the sum of convex functions $f_t(\cdot)$.

Algorithm 1 Online Convex Optimization

```
procedure OCO ALGORITHM
    TotalLoss ← 0
    for t = 1, ··· , T do
        Algo generates x^t, (x^t ∈ P)
        Convex loss function f_t(·) available
        TotalLoss ← TotalLoss + f_t(x^t)
    end for
    return TotalLoss
end procedure
```

The performance of an algorithm like Algorithm 1 above is measured comparing $\sum_{t=1}^{T} f_t(x^t)$ versus $F(x^*)$, where x^* is the minimizer of the function $F(\cdot)$.

The abstract experts problem studied in the previous lectures can be seen as a particular case of OCO. To ”prove”this statement, let

- $\mathcal{P}$ = distribution of actions = Δ^m
- $f_t(x^t) = \langle x^t, \ell^t \rangle$

3.3 Application – scalar prediction

The goal of this problem is to forecast a scalar[1] at each iteration $t = 1, \cdots, T$. Instance definition:

- $\mathcal{P} = [-1; 1]$
- $f_t(x) = |x - a_t|$

Example:

t	Algo	$f_t(\cdot)$	Loss
1	-1	$\|x - 0.8\|$	1.8
2	0	$\|x - 0.7\|$	0.7
3	0.6	$\|x - 0.9\|$	0.3

In this example, TotalLoss $= 2.8$ $(= 1.8 + 0.7 + 0.3)$, versus OPT $= 0.2$, for $x^* = 0.8$.

3.4 Application - (simplified) portfolio optimization

The goal of the problem is to **maximize wealth** at the end of the game.

Setup:

- m stocks
- w_0 initial wealth
- at time t, money is allocated among the stocks: $\mathcal{P} \in \Delta^m$

For instance, at time $t = 1$, component $w_0 \cdot p_i^1$ represents the amount of money allocated to stock i. Only after allocation is done one knows the return of each stock $r_i^t \in [0, \infty)$.

Question: how much money w_t one has at the end of period t?

At the end of period 1, $w_0 \cdot p_i^1 \to w_0 \cdot p_i^1 \cdot r_i^1$. For all the periods,

[1] a scalar x is any number $x \in \mathbb{R}$

$$\begin{array}{rcl} w_1 & = & w_0 \cdot \sum_i p_i^1 \cdot r_i^1 \\ & = & w_0 \langle p^1, r^1 \rangle. \\ \vdots & & \vdots \\ w_{t+1} & = & w_t \langle p^t, r^t \rangle \\ & = & w_0 \langle p^1, r^1 \rangle \cdots \langle p^t, r^t \rangle. \end{array}$$

Since $\log(a \cdot b) = \log(a) + \log(b)$,

$$log(w_{T+1}) = \log(w_0) + \log\langle p^1, r^1 \rangle + \cdots + \log\langle p^1, r^1 \rangle.$$

It is important to note that function $\log(\cdot)$ is an increasing function[2]. So, to maximize w_{T+1} is equivalent to maximize $\log(w_{T+1}) = \sum_{t=1}^{T} \log\langle p^t, r^t \rangle$.

So, writing the problem as an OCO problem,

- $\mathcal{P} = \Delta^m$
- $f_t(p)$ = revenue function = $\log\langle p, r^t \rangle$
- objective: $\max \sum_t f_t(p^t)$.

References

[1] D. Adamskiy, W. M. Koolen, A. Chernov, and V. Vovk. A closer look at adaptive regret. In N. H. Bshouty, G. Stoltz, N. Vayatis, and T. Zeugmann, editors, *Algorithmic Learning Theory*, pages 290–304, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[2] A. Daniely, A. Gonen, and S. Shalev-Shwartz. Strongly adaptive online learning. 02 2015.

[3] M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine Learning*, 32(2):151–178, Aug 1998.

[2]if $y > x$, then $f(y) > f(x)$