EE194/290S: ML for Sequential Decision under Uncertainty **Fall 2018**

Lecture 11: Adaption I

Lecturer: Anant Sahai/Vidya Muthukumar *Scribes: Qiaowei Zhang, Xingyu Lu*

11.1 Introduction – The Skier's Problem

Recall the following scenario:

> *It is your first time skiing, and you need a pair of skis. In the store, you find two options: renting or purchasing the skis. The cost of renting skies is 50 dollars, while the cost of purchasing a new pair is 500 dollars. Which option should you choose?*

Surely, you can just purchase the skis and never worry about them again - but since you are new to skiing, every time you use your skis you have a chance of breaking your legs, after which you will never want to ski again. Not knowing when nature will break your legs, how should you minimize your regret?

To answer this question, we must be more careful about what "minimizing regret" means. Instead of minimizing additive regret (i.e. the actual cost of using the skies for some days before you break your legs minus the optimal cost had you known when you would break your leg), we are more interested in the regret ratio, which is the ratio of the actual cost to the optimal cost. We use the regret ratio, because it allows us to generically bound the regret without having to specify the rental and purchase prices.

The solution to this problem is the **Skier's Rule**: you rent the skis until your total rental cost reaches the cost of purchasing a new pair, and only then will you purchase a new pair. This way, your worst-case regret is never more than half of your cost. If you break your legs before you purchase the skis, you incur 0 regret; after you purchase the skis, your worst case regret would be 500 dollars out of 1000 dollars total, and that happens when you break your legs right after your purchase.

Now, let's think about a completely different scenario: online prediction of a binary stream. Remember that using Multiplicative Weights, our regret is bounded by the sum of two terms: the mix loss regret *(the approximation term)* and the mixability gap *(the estimation term)*. From our analysis, both terms relate to T, which is the total length of the stream. Now, what if we do not know what T is? How should we appropriately balance the two terms in this case?

It turns out that the skier's rule can help us here: if we think of the "approximation error" as the cost of buying the skis, and the "estimation error" as renting the skis, then a natural approach is to increase the approximation error only when we have accumulated enough estimation error. This leads us to the **Doubling Trick**.

11.2 Doubling Trick

To achieve the regret bound $O(\sqrt{T})$ in the multiplicative weights algorithm, we must know the time horizon T in advance, and specify η accordingly. We can avoid having to know T while still achieving the regret bound, at the expense of a constant factor, with the *doubling trick*.

11.2.1 Overview & Algorithm

The central idea of the doubling trick is to use an initial $T = T_0$, and double this value when appropriate. Specifically, the algorithm assumes that the stream should end at time T_0, and runs multiplicative weights (or other online prediction algorithm) with such assumption; if it turns out that the steam does not end at T_0, then we reset $T = 2 * T_0 = T_1$, and runs with the new assumption. Optionally, each time we reset T (and hence η in multiplicative weights), we may also reset the parameters and pretend it is a fresh start.

Data: An online data stream, an online prediction algorithm A
Result: Sequence of predictions
$T = T_0$;
while *true* **do**
  while *have not reached time T and not at the end of the data stream* **do**
    run A with T;
  end
  $T = 2 * T$;
  (optional) Reset parameters of A;
end

Algorithm 1: Doubling Trick

A natural question that may arise at this point is: why may we choose to reset the parameters of the algorithm every time we update T? The idea of throwing away the past experience is not intuitive, but in the following section we will show that both versions give the same regret bound. In fact, thanks to the "crudeness" in our analysis, resetting the parameters actually results in a better constant factor!

11.2.2 Analysis & Proofs

Theorem 11.1 *The regret of using the doubling trick with resets on multiplicative weights (or other zero-regret prediction algorithm) is $O(\sqrt{T})$, where T is the total length of the data stream.*

Proof:
First, we assume that $T_0 = n$, $T \in (\sum_{i=0}^{m-1} 2^i n, \sum_{i=0}^{m} 2^i n]$.

This means that we will double T exactly m times before we reach the end of the stream. During each interval where T is fixed, the regret is bounded by $O(\sqrt{T})$, so the total regret is:

$$\begin{aligned}
\sum_{i=0}^{m} \{\alpha \sqrt{2^i T_0}\} &= \alpha \sqrt{T_0} \sum_{i=0}^{m} (\sqrt{2})^i \\
&= \alpha \sqrt{T_0} \frac{1 - \sqrt{2}^{m+1}}{1 - \sqrt{2}} \\
&= \alpha' (\sqrt{2^{m+1} T_0} - \sqrt{T_0}) \\
&\leq \alpha' (\sqrt{2^{m+1} T_0}) \\
&\leq \alpha' \sqrt{2T} \in O(\sqrt{T})
\end{aligned}$$

■

Theorem 11.2 *The regret of using the doubling trick without resets on multiplicative weights (or other zero-regret prediction algorithm) is $O(\sqrt{T})$, where T is the total length of the data stream.*

Proof:
we assume that $T_0 = n$, $T \in (2^{m-1}n, 2^m n]$. Again, we will double T exactly m times before we reach the end of the steam. Note that total regret of each interval is no greater than the total regret of running from start,which is bounded by $O(\sqrt{T})$. Denoting the total regret of interval i by R_i, we have:

$$\begin{aligned}
\sum_{i=0}^{m} R_i &\leq \sum_{i=0}^{m} \{\alpha\sqrt{2^i T_0}\} \\
&= \alpha\sqrt{T_0}\sum_{i=0}^{m}(\sqrt{2})^i \\
&= \alpha\sqrt{T_0}\frac{1-\sqrt{2}^{m+1}}{1-\sqrt{2}} \\
&= \alpha'(\sqrt{2^{m+1}T_0} - \sqrt{T_0}) \\
&\leq \alpha'(\sqrt{2^{m+1}T_0}) \\
&\leq \alpha'\sqrt{4T} \in O(\sqrt{T})
\end{aligned}$$

■

Remark: While the analysis above may suggest that resetting the parameter is better, empirically not resetting the parameters usually yields better performance. Ultimately, the analysis above is in nature an approximation, and the upper bounds are not tight.

11.2.3 Budget-based Doubling Trick

Recall that the vanilla doubling trick is essentially "pessimistic": we always assume that we get the worst case estimation error. This is not always the case, especially when the environment is not adversarial in nature.

With budget-based doubling trick, we give the current learning rate a budget (the upper bound on estimation error), and only decrease the learning rate if our actual estimation error exceeds the budget.

Data: An online data stream, an online prediction algorithm A, a budget function f
Result: Sequence of predictions
$\eta = \eta_0$;
budget = $f(\eta)$;
error = 0;
while *not at end the data steam* **do**
  if *error* $\geq$ *budget* **then**
    $\eta = \eta/2$;
    budget = $f(\eta)$;
    error = 0;
    (optional) Reset parameters of A;
  end
  Run A;
  error += estimation error
end

Algorithm 2: Budget-Based Doubling Trick

11.3 Performance Comparison among different strategies

In this section, we will compare different flavors of the doubling trick under the simple task of **binary prediction**, which has been thoroughly discussed in lecture:

We have a stream of data of length $T : X_1, X_2, \ldots, X_T$, where each data point $X_i \in \{0, 1\}$. At each time stamp t, having observed $X_1, X_2, \ldots, X_{t-1}$, the task is to predict X_t before it arrives. Our objective is to minimize the cumulative regret incurred by our prediction across T. For each prediction $\hat{X}_t$, we incur the Hamming loss:

$$\begin{aligned} l_t(\hat{X}_t, X_t) &= HAMMING(\hat{X}_t, X_t) \\ &= \mathbf{1}(\hat{X}_t \neq X_t) \end{aligned}$$

We will use all one or all zero as our hindsight candidates, and calculate the hindsight loss at every time step t. We will use three types of sequences for the experiments:

1. $X_i \sim Bernoulli(0.7)$ Best Hindsight: All 1
2. $X_i \sim Bernoulli(0.5)$ Best Hindsight: All 0
3. $X_{1:T} = [0, 1, 0, 1, \ldots]$ Best Hindsight: All 0

11.3.1 Scenario 1: Assuming that T is known

To begin, we assume that T is known, and so we can directly use multiplicative weights on the sequences. We compare the performance of two algorithms: **Follow-The-Leader** and **Multiplicative Weights**.

The graphs below show the cumulative regret of each algorithm:

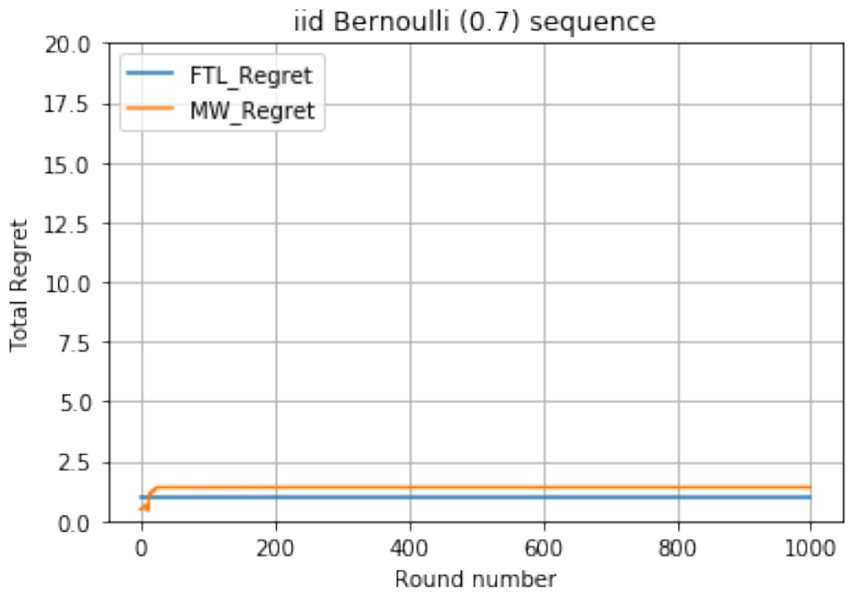

Figure 11.1: T is known and $X_i \sim Bernoulli(0.7)$

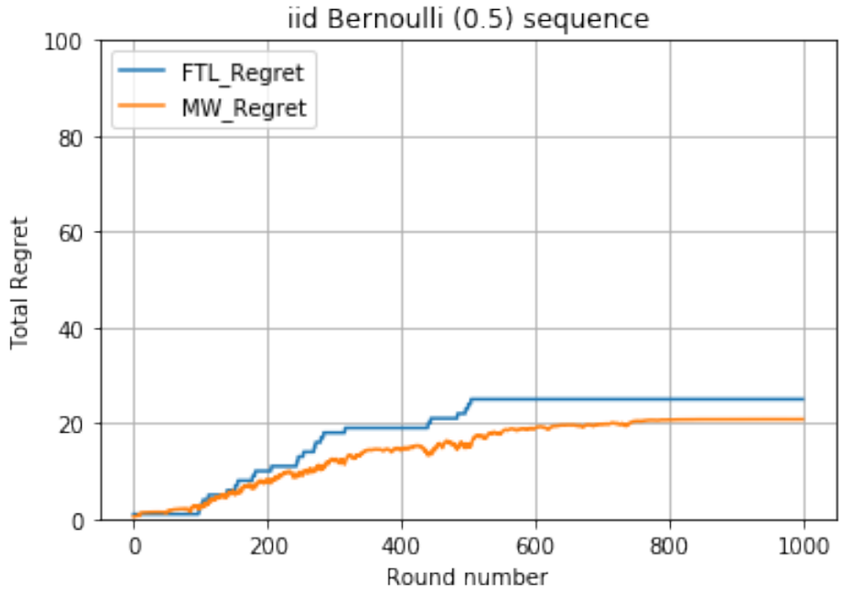

Figure 11.2: T is known and $X_i \sim Bernoulli(0.5)$

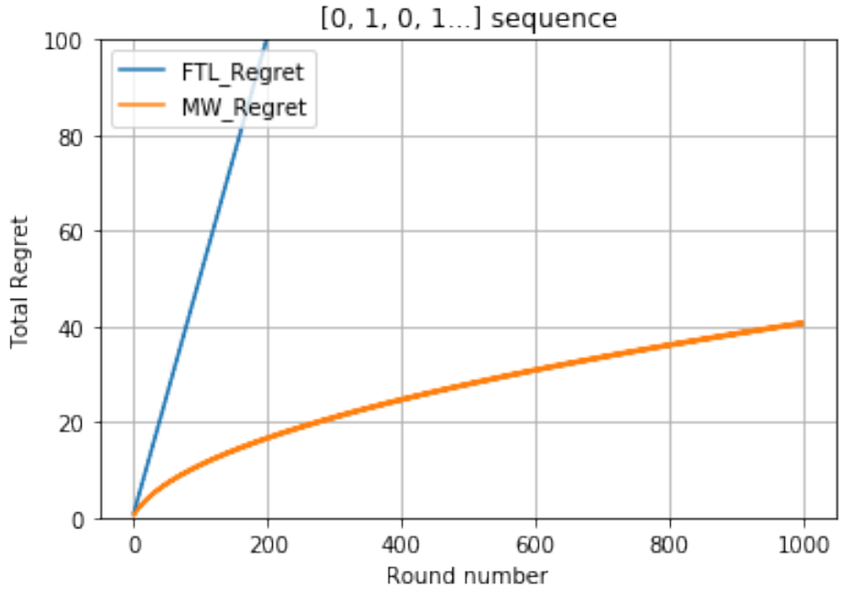

Figure 11.3: T is known and $X_{1:T} = [0, 1, 0, 1, \ldots]$

These experiments should be similar to the ones we did several lectures ago.

11.3.2 Scenario 2: Assuming that T is unknown

We then consider the case where T is unknown. Now, we cannot just use multiplicative weights, because we cannot directly set η. Instead, we use the **doubling trick**, and compare the performance with **Follow-The-Leader**.

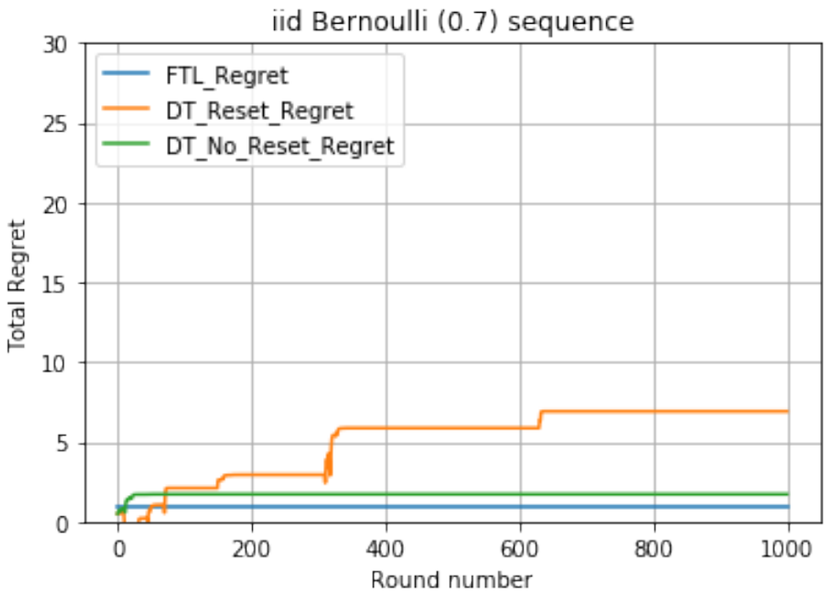

Figure 11.1: T is unknown and $X_i \sim Bernoulli(0.7)$

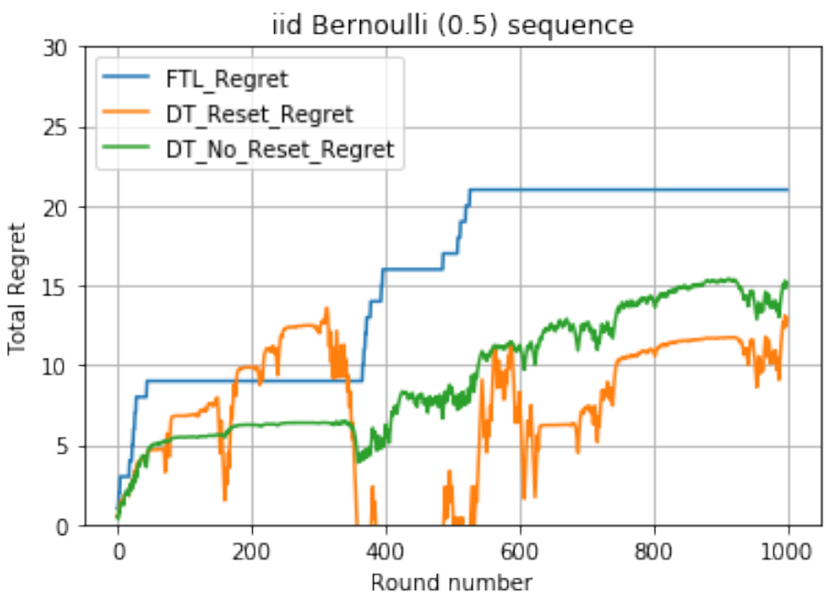

Figure 11.2: T is unknown and $X_i \sim Bernoulli(0.5)$

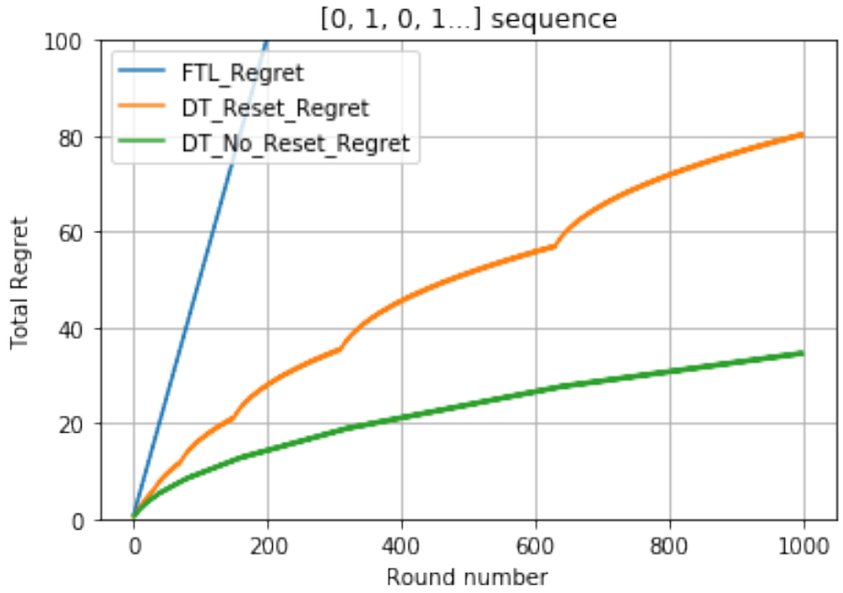

Figure 11.3: T is unknown and $X_{1:T} = [0, 1, 0, 1, \ldots]$

We see that both flavors of the doubling trick achieve sub-linear growth in regret, even though they do not have access to the total length of the sequence T. Interestingly, using doubling trick with resets results in bumps in the curve, due to the parameter resets at the end of each interval.

11.3.3 Scenario 3: Budget-based Doubling Trick

We now include the budget-based doubling trick (AdaHedge). While the specific details of the implementation will be explored in more depth in the next lecture, these graphs should readily show you that budget-based doubling trick has "the best of both worlds": when the environment is nice, it quickly converges to Follow-The-Leader; when there is a lot of uncertainty, it behaves like the normal doubling trick.

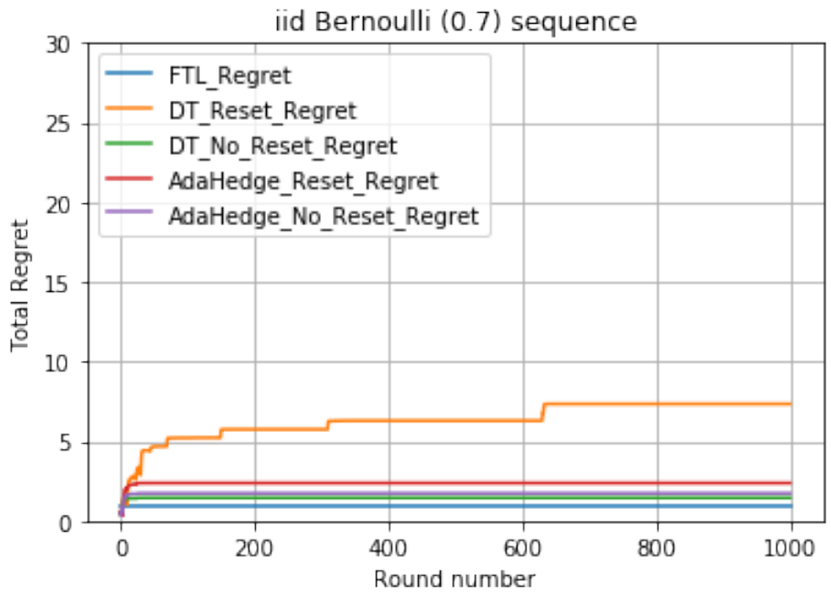

Figure 11.1: T is unknown and $X_i \sim Bernoulli(0.7)$

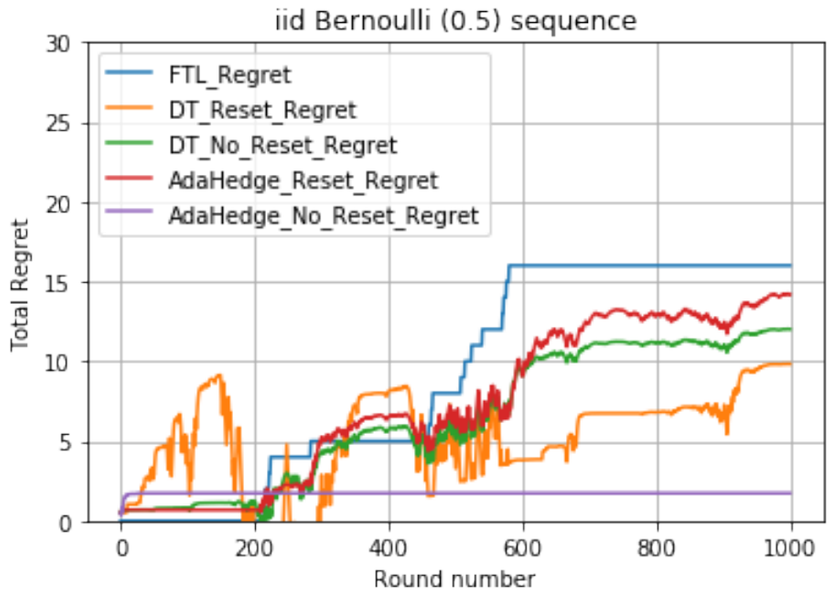

Figure 11.2: T is unknown and $X_i \sim Bernoulli(0.5)$

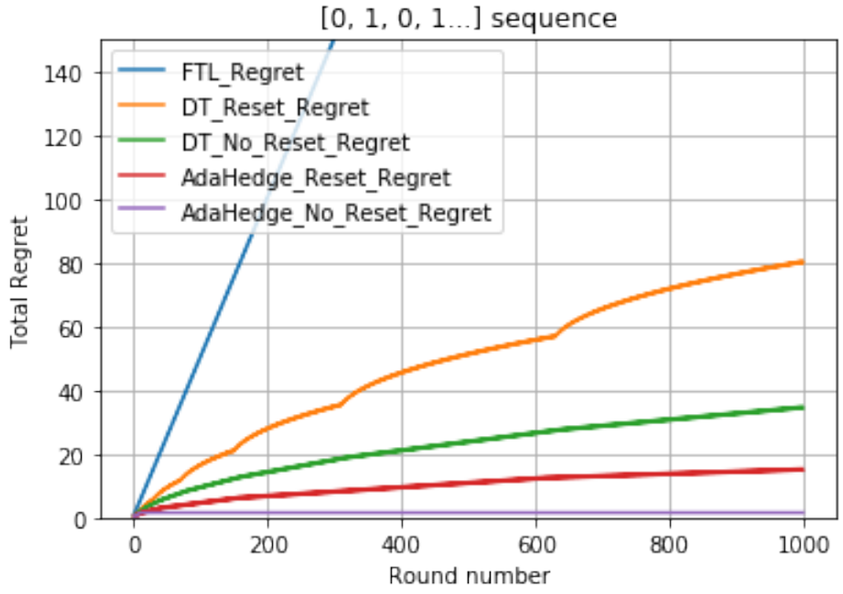

Figure 11.3: T is unknown and $X_{1:T} = [0, 1, 0, 1, \ldots]$