Scribe Notes: Adaptation 4 – 10-05-18

Nikhil Shinde, Nipun Ramakrishnan

October 5, 2018

1 Motivation

In the previous lectures we looked at predicting sequences using algorithms like FTL, Hedge and AdaHedge. The goal in all these cases was to produce an algorithm that incurs the least amount of regret, which is to say it does well against some established set of baselines or reference classes. While evaluating these algorithms we found that they seemed to incur higher losses on a particular sequence:

$$Y = (0, 1, 0, 1, \ldots, 0, 1)^n$$

that we considered adversarial. Yet by just viewing this sequence we can see a pattern indicating that it should in fact be easy to predict.

It was this thought that motivated us to expand our baseline or reference classes to explore better and more expressive models. Using higher order models made predicting previously "hard" sequences much easier. In the specific case of $Y = (0, 1, 0, 1, \ldots, 0, 1)^n$, using models of order 1: $\{[0 \to 0, 1 \to 1], [0 \to 1, 1 \to 1], [0 \to 1, 1 \to 0], [0 \to 0, 1 \to 1]\}$. Since this set contained $[0 \to 1, 1 \to 0]$ we could precisely predict our sequence.

Yet this concept of expanding our set of reference classes with model order poses a new question: Which models and model orders should we consider?

2 Model order selection

2.1 Intuition

When considering models of different orders we have an inherent tension between approximation error and estimation error: A more complex model has the ability to fit to any noise in the sequence, decreasing approximation error by lowering the loss seen but increasing estimation error as the noise is not a key part of the pattern in the observed sequence. Meanwhile, a lower order model that is closer to the "true order" of the sequence seen would have a lower estimation error but may have a higher approximation error, due to its inability

to fit to the data.

In the next section we will explore how to address this problem in an online manner by changing the best model order as we view more data in the sequence.

2.2 A race

Let us start by considering follow the leader (FTL) type approaches. To gain an intuitive understanding of this topic we will analyze this as a race between different orders of models. Picture this by imagining different predictive models lined up at some starting line. As they predict the sequence over time they accumulate reward for the terms that they predicted right. In terms of online prediction we want to bet on the model order that is in the lead or has accumulated the highest reward.

Figure 1: Start of the race

As this race continues along a certain sequence, one model order will appear in the lead and continue to stay forward. We can use this idea of a race to evaluate different model orders for FTL.

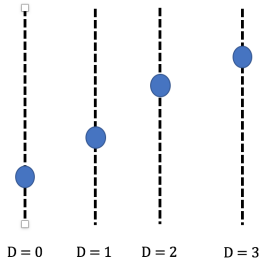

Figure 2: End of the race where D = 3 wins

Let us first consider what it means to have FTL of order or depth D. In classical FTL of order 0 we simply looked at the sequence seen so far and voted with a majority. With depth D FTL we consider all possible D length sequences (of which there are 2^D). Each of these depth D sequence can predict a 1 or a 0, and each permutation of these sequences 2^D predicting either a 1 or a 0 is one of the depth D models. Thus a model order of depth D ends up having 2^{2^D} potential models. When doing online prediction we consider each of these 2^{2^D} models and predict using the model that we have seen the most.

Knowing this we can see that when considering a class of higher order predictors they will clearly have a false advantage. Since there are exponentially more of them there is a higher probability that they will get lucky, essentially **overfitting** to any noise in the sequence, and predict correctly. In other words, these models will end up looking better than they actually are. This gives classes of a higher order an inherent advantage in the model order race. To continue analysis of the race between model orders we must calculate this propensity to overfit and how much regret we will to incur due to this. In order to offset this effect we calculate a penalty as a function of model order, to in effect "event out" the race. Before further analyzing the penalty function and regret let us take an aside to better understand this concept of overfitting.

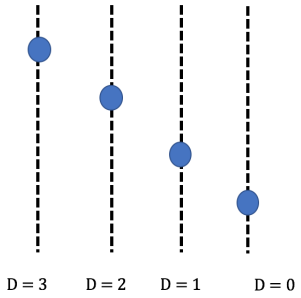

Figure 3: Start of race incorporating penalties

2.3 Aside: Understanding overfitting as a scam

You have come up with a great way to scam people out of money. You start by picking some stock and sending letters to $n = 10000$ people. To the first $\frac{n}{2}$ people you send the following letter:

"You don't know me, but the this particular stock price will go up"

To the remaining $\frac{n}{2}$ you send a similar letter indicating instead that the stock price will go down. Since the stock price will change one way or the other one

of the two messages will be accurate. On the second day, we evenly divide the group of people to whom we sent the correct prediction and tell one half:

"See I was right! The stock will go up again."

We send a similar message the other half the same thing but that the stock went down. In a similar manner to before, our prediction will hold true for half the people. We continue this pattern for several days and as a result we will have constantly given correct predictions to $\frac{n}{2^{days}}$ people. Having established credibility we now ask them for money for future predictions! The more people we send letters to, the more people we get all the predictions correct for by construction. Thus the people that trust in us do so purely out of dumb luck! This is akin to what we see when we increase the model order. We get more predictions correct purely due to the reason that we have more models that can capture random variability in the data.

3 Overfitting in Trees

We've seen that the penalty function for higher order models with model orders of D is on the order of 2^D. Why is this the correct choice? To gain some intuition on this, let's analyze the potential for overfitting in tree-like models that represent our prediction problem. In a general sequence $\{0,1\}^N$, a depth D tree predictor in an online setting will look at the previous D bits to determine the next bit. We will show that such a model has an $O(2^D)$ overfit potential in ther worst case.

One worst case scenario that's particularly nice to analyze is the Bernoulli($\frac{1}{2}$). Notice that in this case, the true model is completely random, so all depth D models will overfit. We will define this capacity of overfit as a worst case count over time of how many predictions these higher order models got correct just by dumb luck of how the sequence realized itself.

Assuming we have a T length sequence where T is much larger than 2^D(implying we have seen many more contexts than possible models of depth D), we can approximate the situation as a balls and bins problem in which each node in the tree represents a bin and each context of the sequence represents a ball. We know from the study of the behavior of balls and bins that as the number of balls becomes large, the distribution of balls in each bin approaches the uniform distribution.

With 2^D bins, each bin gets $\frac{T}{2^D}$ realizations. Each bin has a count of the respective 1's and 0's, one of which is the leader for that context. The leading 0 or 1 will be ahead $O(1+\sqrt{\frac{T}{2^D}})$. This incorporates the fact that each realization has seen the context at least once and takes into the account the standard deviation of the realizations. Consequently, each of the 2^D bins have $O(1+\sqrt{\frac{T}{2^D}})$ of regret, as they think that either 1 or 0 is ahead by $O(1+\sqrt{\frac{T}{2^D}})$ even

though it is completely random. Thus we can bound the total regret by

$$O(2^D(1+\sqrt{\frac{T}{2^D}})) = O(2^D + 2^{\frac{D}{2}}\sqrt{T})$$

We can interpret this bound as the first 2^D term representing the first time a context has been seen and the next term incorporating the variance of the contexts in this randomized sequence. Overall, this analysis shows that we have at least $O(2^D)$ regret in the worst case, so we have to penalize higher order models enough to overcome the possibility of getting contexts correct by dumb luck.

4 Regret Analysis in the Race Context

Let's now revisit the race perspective that we introduced earlier. Assuming the data comes from a true model of order d, we want to understand how much regret we accumulate relative to the best d^{th} order model. We also would like to understand the evolution of how our model orders change as we progress through the algorithm. Recall, that our algorithm for model selection makes a decision according to the following optimization problem:

$$\underset{m}{\text{argmax}}\, Reward(m, (x_1, x_2, \ldots x_T)) - Penalty(m)$$

where m is the model order.

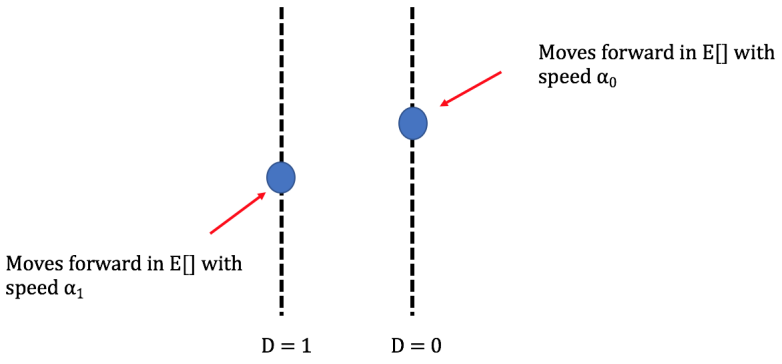

Figure 4: Speed of model orders during race

To understand how to analyze regret in the context of the race, let us first understand how the race will evolve over time: The simplest will win for a while, then more complicated models will overtake it until the model with the true order d comes out ahead and stays in the lead (This can have some shaking

due to random rewards and losses caused by fitting to noise).
In this context regret can be conceptualized as the amount of time it takes for the best model order to overtake the rest. To make this easy to understand let us consider just two model orders:0 and 1 racing against one another. Consider the graphic above for this example where model order 0 is moving at speed α_0 and model order 1 is moving at speed α_1 where $\alpha_1 > \alpha_0$. We define speed to be the reward gained by a model at each time step. With these speeds it takes the best runner: order 1:

$$\frac{penalty(1) - penalty(0)}{\alpha_1 - \alpha_0}$$

time to overtake. Since we know the speed we know that we incur α_i reward for model order i at each time step making our regret in this case: $\alpha_1 - \alpha_0$. Our total regret is:

$$(\alpha_1 - \alpha_0) * \frac{penalty(1) - penalty(0)}{\alpha_1 - \alpha_0} = penalty(1) - penalty(0)$$

. Thus total regret is exactly the difference between the penalties! Using this we can guarantee low regret assuming our sequence is stochastic.

4.1 Model selection in an Adversarial context

We can also consider model order selection under adversarial scenarios. In such a case we combine the perspective taken above and combine it with the entropy regularization. This gives us a new algorithm:

$$\text{softmax}_m[Reward(m, (x_1, x_2, \dots x_T)) - \frac{1}{\eta} * (Penalty(m) - H(\omega))]$$

This serves to protect against adversarial sequences by introducing randomization via $H(\omega)$ or entropy. The $\frac{1}{\eta}$ serves as the learning rate as we have seen before. Here we use the same budget as elegant adahedge to set the learning rate. The more regret we incur we drop the learning rate. However when we drop the learning rate we end up applying a bigger model order penalty making it take more time for the true model to take a lead in the race. Due to this we incur a larger regret though it can show it is only off by a constant multiple making it the same $O(2^D)$

Another consideration when penalizing model orders when using exponential weights is to penalize by using priors. You effectively use priors to downweight model orders that are less likely to be seen.

5 Computational points

When it comes to implementation there is a computational note to be addressed. As it was shown previously, there are 2^{2^D} models of order D to consider when

running this algorithm. To keep track of all these models makes it computationally unfeasible to consider many model orders.
To solve this we implement the algorithm only using the $O(2^D)$ contexts where D is the highest model order considered. Since there are only 2^D unique sequences for model order D, when running online prediction we only have to keep track of how many times we saw a 1 or a 0 after the occurrence of that sequence, then the next time we see the same sequence we can just predict a 1 or 0 based on which of the two has a higher count for the observed context. This drastically reduces the amount of information we have to store as we can simply store information for contexts rather than for every individual model itself.
To implement this we can construct a binary tree where models of order d are represented by the dth layer in the tree. The root node keeps track of the 0s and 1s that have been seen for model order zero and each node has a '0' child and a '1' child that denotes the sequence that is seen before predicting with the contexts stored in that node. When operating in an online fashion, every time we see a new entry in the sequence we we take the latest length d sequence and go down the tree updating the counts for the contexts stored in each node with the next item that has been seen. Similarly to predict we traverse down the latest length d sequence and choose a 1 or a 0, whichever has a higher count in that context. This drastically reduces our runtime per prediction to $O(D)$ and changes our memory complexity to $O(2^D)$

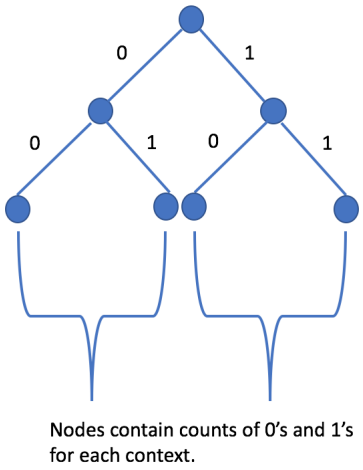

Figure 5: Tree example at depth $d = 2$