Multi-Armed Bandits: Exploration vs. Exploitation

Subhash Suri

January 27, 2020

1 Explore vs. Exploit

- Many choices in life come down to this: *try something new* or *go with a familiar option*? For example, shall we go to an old favorite restaurant, or try a new one? Try a new pizza topping or stay with the old favorite? Read a book by an old favorite author, or try a new author?

- What is the right tradeoff between trying new things or sticking with old favorites? After all, our current "old favorite" is probably a result of us trying something new once; every "best restaurant" or "best song" has a humble beginning: before we "discovered" it, we had some other favorite.

- Computer scientists study these phenomenon under the name "explore or exploit". Explore = try a new option, while Exploit = stay with an old favorite.

    1. When we explore we are gather new information, and when we exploit, we are using our current knowledge to make the best decision.
    2. In most interesting cases, a *balanced strategy* is needed: if we never explore, we learn nothing new, and never expand our horizon. But if we endlessly explore, without exploitation, we may be throwing away all the information we have accumulated. Trying a different restaurant every night, and never going back to the really great ones, seems like opportunity lost.

- This is also a classic *reinforcement (machine) learning* problem, exemplifying the exploration-exploitation tradeoff.

- Many practical applications confront this explore vs. exploit dilemma:

1. Clinical trials: efficacy of different treatments while minimizing patient losses.
2. Adaptive routing efforts for minimizing delays in a network.
3. Financial portfolio design.
4. Web page designs for maximizing advertising revenue.

1.1 Multi-armed Bandits

- In CS, the tension between exploitation and exploration is studied under the concrete mathematical framework of *Multi-Armed Bandit*. This odd name comes from casino slot machines called "one armed bandits."
- Imagine walking into a casino full of different slot machines.
  - Each of which has its own odds of payoffs, but we *do not know* those odds.
  - We *learn* these odds only by playing them.
- What strategy should we use to maximize your expected win?
- It is clear that some combination of trying different machines (explore) and favoring the most promising ones encountered so far (exploit) is needed. But how to devise a precise strategy around this general principle?
- To get a sense of the problem's subtleties, imagine the simple case of just two machines.
  - One machine paid off *9 out of 15* times we played.
  - The second machine paid off *1 out of 2* times.
  - Which is more promising?
- Simply dividing the wins by the total number of pulls gives us an estimate of the machine's "expected value." By that method, the first machine with a 9-6 record, and expected value 60 %, is better. (The second one has only 50% exp. value.)
- But, on the other hand, we have played the 2nd machine only twice, so in a sense we just don't know how good it may actually be.
- The MAB problem models situations where an agent simultaneously tries to acquire new knowledge (exploration) and optimize decisions based on existing knowledge (exploitation). The agent attempts to balance these competing tasks in order to maximize their total value over a period of time.

- **Time Horizon in MAB.** An important aspect of explore vs. exploit, which is some times overlooked in other models of human decision making, is the role of "time interval." People often focus on the "next decision"—where to eat today or which song to listen to—with the goal of maximizing the expected value. But these decisions are almost never isolated—we will be making those decisions many many times in the future, and so our algorithm must consider a long horizon. In the casino game, e.g., the strategy depends on *how long you plan to be in the casino!*

- When balancing favorite experiences and new ones, the *interval* over which we plan to play matters a lot. When you are new in town, you tend to explore a lot and try many different restaurants. Once you have become more familiar, your exploration drops and you tend to visit old favorites more and more.

    1. Intuitively, the *value of exploration goes down with time* as the remaining opportunities dwindle. Discovering an amazing cafe on your last night in a new city does not give you any exploitation opportunity to return to it.
    2. On the other hand, the value of exploitation *goes up with time*. The best cafe you know today is at least as good as the best cafe you knew last month.

- The movie producers in Hollywood also seem to be using this strategy: the number of *franchise* sequels (exploit), with a guaranteed fan base, vs. the number of brand-new movies (explore).

1.2 Win Stay Algorithms

- In spite of its seeming simplicity, finding optimal algorithms for multi-armed bandit has proved to be extremely difficult. Some researchers working on it during WWII even joked that "the problem should be dropped over Germany as the ultimate instrument of intellectual sabotage."

- The first important contribution towards a solution was made by Herbert Robbins (Columbia mathematician) who proved that a simple strategy, while not optimal, delivered some nice guarantees.

- Robbins considered the simple setting of *two slot machines* and proposed a solution called the "Win-stay, Lose-shift" algorithm. The algorithm is this:

    > *choose an arm at random, and keep pulling as long as it keeps paying off. If the arm does not pay off after a pull, switch to the other arm.*

- While simple, the strategy is not trivial to analyze. In particular,

1. the Win-Stay move makes intuitive sense: if you were willing to pull an arm and it just paid off, then it should only increase your estimate of its value, making you even more willing to pull it next.
2. but what about "Lose-Shift"? Changing arms every time one fails to payoff seems like a rash move. Imagine going to a restaurant 100 times, and having a wonderful meal each time. Would one disappointing meal be enough to give up on it? Good options should not be penalized too strongly for being imperfect.
3. Even more significantly, the Win-stay, Lose-Shift does not have any notion of the interval over which we are optimizing.

- Nevertheless, Robbins showed that it is guaranteed to be better than one based on pure chance, and his work kicked off substantial research over the next several years.

- If we know in advance exactly how many options and opportunities we will have in total, then as in the full-information secretary problem, we can use Dynamic Programming and work backwards—what machine to choose for the final pull given *all possible earlier outcomes*, and having figured that out, turn to the next-to-last pull and so on. However, except for very simple settings, *the computational cost of this algorithm is simply formidable, and not practical.*

1.3 Gittins Index

- In 1970s, the Unilever corporation asked a young mathematician named John Gittins to help them optimize their drug trials. Unexpectedly, what they got was an answer to the MAB. (This is often the case in mathematics, where a fundamental problem comes up independently in many areas, with unexpected connections.)

- Gittins is now a professor of statistics at Oxford. The basic question Unilever was asking involved some tradeoffs of the form: *given several different chemical compounds, what is the most efficient way to determine which compound is likely to be effective against a disease?*

- Gittins cast the problem as a "stochastic optimization" problem: *given multiple options, a different probability of reward for each option, and a certain amount of effort (money, time, etc) to be allocated among them, find the best allocation*—which is basically the MAB problem.

- Drug companies and medical professionals face the same competing demands of MAB's explore vs. exploit.

1. Doctors want to prescribe best *existing* treatments to their patients get the best care, but they also want to *encourage experimental* studies that may turn up even better ones.
2. Similarly, drug companies want to invest research money into discovery of new drugs but they also want to make sure their current profitable product lines are flourishing.

- In order to deal with the problem of *infinite future*, Gittins approached the MAB problem using the idea of "discounting." The idea behind discounting is that *any gain today has higher value than the same gain in the future.* This idea is not new and is common practice in economics.
- Gittins made the assumption that the value of payoffs decreases *geometrically.* That is, the *present-day* value of a gain v realized t days into the future is $v\gamma^t$, where $\gamma < 1$ is the geometric discounting factor. (Analogy with monetary *inflation.*)
- For instance, if you believe there is a 1% chance you will be hit by a bus on any given day, then a dinner tomorrow is worth only 99% of today's dinner.
- Working with geometric-discounting assumption, Gittins was able to analyze the MAB problem using a "Deal or No Deal" style tradeoff.

  1. In the Deal-or-No-Deal, the contestant chooses one of 26 briefcases, containing prizes ranging from a penny to a million dollars.
  2. During the game, a mysterious Banker will periodically call in and offer the contestant various sums of money (which may go up and down, depending on which briefcases have been opened) to *not open the chosen suitcase.*
  3. The contestant must decide whether to take the *sure* thing offer over the uncertain prize in the suitcase.

- Gittin's reasoning, which predated Deal or No Deal by several decades, was that MAB problem poses the same dilemma. For each slot machine we know little or nothing about, there is some guaranteed payout which, if offered, will make us trade: we will accept the payoff and not pull the arm.
- This number, which became known as *Gittins Index*, is called "dynamic allocation index" suggests the following strategy: *always play the arm with the highest index.*
- Gittins index solves the MAB completely *for geometric discounted payoffs.* However, *calculating the index for a machine, given its track record and discounting rate, is still quite involved.* But once it has been calculated for a particular set of assumptions, it can be used for any problem of that form.

- Homework: read up and explain how Gittins index is calculated. What's the main intuition?

1.4 Regret and Optimism

- Gittins algorithm suffers from two shortcomings: first, computing the index for an MAB instance is quite complicated, and second, the algorithm only works under geometric discounting.
- An alternative approach, based on the idea of *regret*, addresses both of these shortcomings.
  - Regret compares what we actually did with that would have been best in hindsight, and serves as a measure of *wrong choice's* effect.
  - More formally, an *algorithm's regret score* is the difference between the total payoff obtained by following that algorithm and the total payoff that *theoretically* could have been obtained by just pulling the best arm every time *(had we known from the start which arm was the best).*
  - Computer scientists approach the MAB problem with the goal of finding an algorithm that minimizes *expected regret.*
- In 1985, Robbins reanalyzed MAB using the regret framework, and was able to prove some basic results.
- Homework: read up and explain the logarithmic regret lower bound.
- Following the work of Lai and Robbins, researchers have been searching for algorithms that deliver good guarantees on minimum regret. One of the most popular strategies is based on *Upper Confidence Bound,* which is based on the following intuition:
  - Statistical measurements typically are presented with error bars, which show the range of plausible values. This range is called *confidence interval.* (For instance, the 95% confidence range.) As we gain more data, this confidence interval shrinks, reflecting more accurate assessment.
  - For instance, slot machine A that has paid out once out of two tries has a wider confidence interval than one that has paid out 5 out of 10 times, although they both have same expected value.
- In the MAB problem, the Upper Confidence Bound (UCB) algorithm, also called Optimism under Uncertainty, always picks the option for which the top of the confidence bound is highest. (That is, the option with the highest uncertainty.)

- Like Gittins index, UCB assigns a single number to each arm of MAB, and that number is set to the highest value that the arm could *reasonably* have based on the current information.
- The UCB algorithm does not care which arm has performed the best so far. Instead, it chooses the arm that *could* reasonably perform the best. UCB for an option is always greater than its expected value, but less and less as we gain more experience with that option.
- UCB numbers are significantly easier to compute, and do not require the assumption of geometric discounting.

2 Deriving Regret Bounds for MAB

- We now delve into mathematics of how to formally *prove* the upper bounds on the regret of MAB algorithms. We will consider the basic model, called *stochastic bandits*, with i.i.d. (independent, identically distributed) rewards.
- In particular, the algorithm has K possible actions (arms) to choose from, and there are T rounds, where we assume K and T are known, called the *finite horizon version*. In each round, the algorithm chooses an arm and collects a reward for that arm. Thus, the protocol has the following form:

  Given K and T, in each round $t = 1, 2, \ldots, T$, do
  the algorithm selects some arm a_t
  the algorithm receives reward $r_t \in [0, 1]$ for this arm.

- The algorithm's goal is to maximize it total reward over the T rounds. We make three key assumptions:

  1. The algorithm only observes the reward for the arm it selected. It learns nothing about the rewards of the other actions it could have taken but did not. This is called *bandit feedback*.
  2. The reward for each action is i.i.d.: for each action a, there is a probability distribution $\mathcal{D}_a$ over reals, called the *reward distribution*. Every time this action is chosen, the reward is sampled independently from this distribution. *$\mathcal{D}_a$ is unknown to the algorithm at the start; it only learns about $\mathcal{D}_a$ from the sample rewards it observes.*
  3. Per-round rewards are bounded, which we can assume without loss of generality to be in the interval $[0, 1]$.

- One of the simplest reward distribution is Bernoulli, where the reward of each arm is either 1 or 0 (success or failure). This distribution is fully specified by the probability p of success at each trial. Observe that p is also the *mean reward*. More generally, the reward distribution can take values in $[0, 1]$.

2.1 Example Applications

The stochastic bandit is a simple abstraction for many applications, such as:

- **News.** In a simple web news application, a user visits a new site, which presents it with a news header, and the user either clicks on it or not. The goal of the news site is to maximize the number of clicks. Each possible news header is a bandit arm, and clicks are (Bernoulli) rewards. A common modeling assumption is that each user is drawn independently from a fixed distribution over the users, and so in each round the clicks are independent with a prob. that only depends on the chosen header.

- **Ad Selection.** A user visits a webpage, and a learning algorithm selects one of the many possible ads to display. If the ad a is displayed, the website observes whether the user clicks on it, and in case of a click the advertiser pays some amount $v_a \in [0, 1]$. Each ad is an arm and payment is the reward.

- **Medical Trials.** A patient visits a doctor, who can prescribe one of many different treatments. Once a treatment is prescribed, the doctor can observe its effectiveness. Then the next patient arrives, and so on. Let us quantify the treatment effectiveness as a number in $[0, 1]$. Each treatment is an arm, reward is treatment effectiveness, and number of patients is the number of rounds.

2.2 Key Definitions

- **Notation.** Throughout, we use a to denote an arm (out of K arms), t to denote a round (out of T rounds). The set of all arms is $\mathcal{A}$.

- The mean reward of arm a is $\mu(a) = \mathbf{E}[\mathcal{D}_a]$. The best mean reward is denoted as

$$\mu^* := \max_{a \in \mathcal{A}} \mu(a)$$

- The difference $\Delta(a) = \mu^* - \mu(a)$ describes how bad arm a is compared to the (omniscient) optimal arm, which we denote as a^*.

- How do we argue whether an algorithm is doing a good job or not across different problem instances, where some instances inherently allow higher rewards than others? Our approach is to compare the algorithm to the best one could possibly have done on a given instance *if one knew the mean rewards*.

- More formally, consider the first t rounds, and compare the cumulative mean reward of the algorithm against $\mu^* \times t$, which is the expected reward of always playing the optimal arm:

$$R(t) \;=\; \mu^* t \;-\; \sum_{s=1}^{t} \mu(a_s)$$

  This quantity is called *regret* at round t.

- We note that $\mu(a_s)$ is random quantity because it may depend on the randomness in rewards as well as random choices made by the algorithm. So, the regret is also a random variable, and we will typically talk about its expected value $\mathbf{E}[R(t)]$.

- The regret definition sums over all rounds, and so it is the *cumulative* regret. We call $R(T)$ *realized* regret, and $\mathbf{E}[R(T)]$ the *expected* regret.

- We will use big-Oh notation to focus on the asymptotic dependence of our regret bound, and not worry about the constant factors.

2.3 Simple Algorithm: Uniform Exploration

- We begin with a very simple algorithm but its analysis sets the stage for more sophisticated results.

- Our simple algorithm works as follows: explore arms uniformly, at the same rate, regardless of what we have observed previously, and then select the empirically best arm to exploit for all the remaining rounds. Specifically, the algorithm has 3 phases:

  1. Exploration: try each arm N times (N to be determined by analysis).
  2. Select the arm a' with the highest average reward (break ties arbitrarily).
  3. Exploitation: play arm a' in all remaining rounds.

- The parameter N is chosen as a function of the horizon T and the number of arms K, and it is fixed at the onset.

- Let the average reward of each action a after exploration phase be denoted as $\mu'(a)$. How good an estimate is this of the true expected value $\mu(a)$?

- Specifically, what can we say about the difference $|\mu'(a) - \mu(a)|$? We will use Hoeffding's inequality to quantify this gap.

2.4 Review of Hoeffding's Inequality

- Let $X_1, X_2 \ldots, X_n$ be independent random variables bounded in the range $[0, 1]$, namely, $0 \leq X_i \leq 1$. (More general statements of the inequality deal with random variables bounded in arbitrary ranges $X_i \in [a_1, b_i]$.)
- Define the *empirical mean* of these variables by

$$\bar{X} \quad = \quad \frac{X_1 + X_2 + \cdots + X_n}{n}$$

- Then, Hoeffding's inequality says that, for all $z \geq 0$,

$$Pr(\bar{X} \; - \; \mathbf{E}[\bar{X}] \; \geq \; z) \quad \leq \quad e^{-2nz^2}$$

2.5 Analysis of the Uniform Exploration Algorithm

- If we use the confidence radius value $r(a) = \sqrt{(2 \log T)/N}$, then Hoeffding's inequality tells us that

$$Pr(|\mu'(a) - \mu(a)| \; \leq \; r(a)) \quad \geq \quad 1 - \frac{2}{T^4}$$

  - Proof. If μ' is the observed mean over N trials, then by Hoeffding's (multiplying the RHS by 2 to account for the absolute value in LHS):

$$Pr(|\mu' - \mu| \; \geq \; r) \quad \leq \quad 2e^{-2Nr^2}$$

  We want $2e^{-2Nr^2} \; \leq \; 2/T^4$, or $T^4 \leq e^{2Nr^2}$, which is guaranteed for $r \geq \sqrt{(2 \log T/N)}$.

- Thus, the prob. that the empirical mean deviates much from the true mean is very small.
- For simplicity of proof, let us first consider the case of two arms $K = 2$. Let us call the *clean event* to be the event that the previous Hoeffding's inequality is true for *both* the arms simultaneously.
- We break our proof into two parts: analysis of clean event and analysis of the remaining (bad) event.
- First consider the Clean Event. We will show that even if we accidentally choose the worse arm, the regret cannot be that bad because the expected rewards of both the arms must be very close.

- Let the best arm be a^*, and suppose the algorithm (wrongly) chose the other arm $a \neq a^*$.
- That must be because $\mu'(a) > \mu'(a^*)$. Since we are in the clean event, we have

$$\mu(a) + r(a) \;\geq\; \mu'(a) \;>\; \mu'(a^*) \;\geq\; \mu(a^*) - r(a^*)$$

- Rearranging the terms, we get

$$\mu(a^*) - \mu(a) \;\leq\; r(a) + r(a^*) \;=\; O(\sqrt{\frac{\log T}{N}})$$

- Thus, each round in the exploitation phase contributes at most $O(\sqrt{\frac{\log T}{N}})$ to regret. Each round in Exploration trivially contributes at most 1. Thus the upper bound on regret is the sum over the N exploration rounds and $T - 2N$ exploitation rounds:

$$R(T) \quad\leq\quad N + O(\sqrt{\frac{\log T}{N}}) \times (T - 2N)$$

  which is at most $N + O(\sqrt{\frac{\log T}{N}} \times T)$.
- We choose the value of N to minimize this regret: set N so that the two terms are equal, which occurs for $N = T^{2/3}(\log T)^{1/3}$.
- Then, the upper bound on regret is

$$R(T) \quad\leq\quad O(T^{2/3}(\log T)^{1/3})$$

- To complete the proof, we also have to analyze the bad event. But that is easy since regret can be at most T, because each round can contribute at most 1, and the bad event occurs with very small probability $1/T^4$, the expected value of regret from this event is negligible: $T \times O(1/T^4) = O(1/T^3)$.
- We now use the standard definition of expectation:

$$\mathbf{E}[R(T)] \;=\; \mathbf{E}[R(T) \mid \text{clean event}] \times Pr(\text{clean event}) + \mathbf{E}[R(T) \mid \text{bad event}) \times Pr(\text{bad event})$$

- Thus, the expected regret for our algorithm is $O(T^{2/3}(\log T)^{1/3})$.

- This completes the proof for $K = 2$ arms.
- For $K > 2$ arms, we have the apply the Hoeffding's inequality over the union of K arms, and then follow the same argument. We clearly must have $T \geq K$ since we need to explore each arm at least once.
- In this case, the optimal choice of N is $(T/K)^{2/3} \times (\log T)^{2/3}$, which gives the final regret bound of

$$\mathbf{E}[R(T)] \quad \leq \quad T^{2/3} \times O(K \log T)^{1/3}$$

2.6 Adaptive Exploration and UCB Algorithms

- Can the regret bound of the simple uniform exploration be improved?
- One major flaw of the simple algorithm is that it completely ignores the history of the observed rewards. For instance, it continues to try an arm in round-robin fashion even if it yields very poor results.
- It would seem natural to *adapt* the explorations to observed rewards. For instance, if we had $K = 2$ arms, we could alternate between them until we find that one arm is "much better" than the other, at which point we just abandon the inferior one.
- The difficulty is quantifying the question: "when is one arm much better"?
- Let us do some probability calculations using Hoeffding's inequality.
- Suppose at the end of round t, the number of times arm a has been tried is $n_t(a)$, and let $\mu'_t(a)$ be the average reward of this arm so far. We want to bound the prob. that the true mean $\mu(a)$ lies within the confidence radius $r_t(a) = \sqrt{(2 \log T)/n_t(a)}$ of the empirical mean.
- We are now using a random variable $n_t(a)$, instead of a fixed N, in the denominator of the confidence radius, but the Hoeffding's inequality can still be applied, and gives us the error prob. for varying values of $n_t(a)$.
- More specifically, for each integer j, where $0 \leq j \leq T$, let $v'_j(a)$ be the average reward from the first j pulls of the arm a. Then, Hoeffding's inequality shows that for any arm a and any j, the following holds:

$$\Pr(|v'_j(a) - \mu(a)| \ \geq \ r_t(a)) \quad \geq \quad \frac{2}{T^4}$$

- By the union bound (over all a's and j's), where we use that $K \leq T$, we get

$$\Pr\left(\forall a \forall j \;\; |v'_j(a) - \mu(a)| \;\geq\; r_t(a)\right) \quad \geq \quad \frac{2}{T^2}$$

- The *clean event* we are interested in is the following:

$$\mathcal{E} \quad = \quad \{\forall a \forall j \quad |\mu'_t(a) - \mu(a)| \;\leq\; r_t(a)\}$$

  that is, all empirical values are close to their true expectations.

- The prob. of this event is exactly what we just established above, so we have

$$Pr(\mathcal{E}) \quad \geq \quad 1 - \frac{2}{T^2}$$

- Define the *upper and lower confidence bounds* for arm a in round t:

$$UCB_t(A) \quad = \quad \mu'_t(a) + r_t(a)$$

$$LCB_t(A) \quad = \quad \mu'_t(a) - r_t(a)$$

- Call the interval $\quad [LCB_t(a),\; UCB_t(a)] \quad$ *confidence interval.*

2.7 Analysis of $K = 2$ Arms

- Let the two arms be denotes a and a'.
- The adaptive algorithm for $K = 2$ arms works as follows:

  1. Alternately play the two arms until $UCB_t(a) \;<\; LCB_t(a')$ after some round t.
  2. Abandon arm a, and use arm a' forever.

- To analyze the algorithm, we can once again focus on the clean event—the prob. of bad event has been made so small that its contribution to the expected regret will be negligible.
- Under the clean event, the disqualified arm cannot be the best arm. Why?
- We need to bound how much regret have we accumulated *before disqualifying one arm?*

- Suppose t is the last round where we still did not invoke the stopping rule. Then, in round t, the two arms' confidence intervals still overlap. Therefore, we must have

$$\Delta \;=\; |\mu(a) - \mu(a')| \;\leq\; 2(r_t(a) + r_t(a'))$$

- Since we have been alternating between the two arms upto round t, we have $n_t(a) = t/2$ (ignoring floor or ceiling). Thus,

$$\Delta \;\leq\; 2(r_t(a) + r_t(a')) \quad\leq\quad 4\sqrt{\frac{2\log T}{t/2}} \quad=\quad O(\sqrt{\frac{\log T}{t}})$$

- The total regret accumulated til round t is

$$R(t) \quad\leq\quad \Delta \times t \quad\leq\quad O(\sqrt{t \log T})$$

- The total contribution of bad events is at most 1 per round. There are t rounds, and the bad event occurs with prob. $O(1/T^2)$, so in total it contributes at most $O(t/T^2)$ to regret, which is much smaller than the regret from clean events.

- **Theorem:** For $K = 2$ arms, the Confidence Bound algorithm achieves expected regret $O(\sqrt{t \log T})$ at each round $t \leq T$.

- The $\sqrt{t}$ regret ratio is much better than the $T^{2/3}$ bound for non-adaptive algorithm.

2.8 Adaptive Exploration for K Arms

- The algorithm generalizes to K arms as follows:

    1. Initially all arms are set "active."
    2. Each Phase:
    3. try all active arms (thus each phase may contain multiple rounds)
    4. deactivate all arms a s.t. $\exists$ arm a' with $UCB_t(a) < LCB_t(a')$.
    5. Repeat until end of rounds.

- We can then carry out a similar analysis to show that this algorithm has expected regret of $O(\sqrt{Kt \log T})$, for all rounds $t \leq T$.

3 Optimism Under Uncertainty Algorithm

- We now consider another approach for adaptive exploration, called *optimism under uncertainty*. It works on the following natural principle: assume each arm is as good as it can possibly be given observations so far (optimism) and choose the best arm using these optimistic estimates.

  1. Try each arm once.
  2. In each round t, for $t \geq 2$, pick

  $$\text{argmax}_{a \in \mathcal{A}} UCB_t(a), \quad \text{where } UCB_t(a) = \mu'_t(a) + r_t(a)$$

- Intuition. We choose an arm in round t because it has a large UCB, which can happen for two reasons:

  1. the average reward $\mu'_t(a)$ is large, in which case the arm is likely to have high reward, and/or
  2. the confidence radius $r_t(a)$ is large, in which case the arm has not been explore enough.

- Either reason makes this arm worth exploring.

- To analyze this algorithm, let us again focus on the clean event. Recall that a^* is the optimal arm, and a_t the arm chosen by the algorithm in round t.

- According to the algorithm's selection rule, we must have $UCB_t(a) \geq UCB_t(a^*)$.

- Since we are in the clean event, we have $\mu(a_t) + r_t(a_t) \;\geq\; \mu'_t(a_t)$, and of course $UCB_t(a^*) \geq \mu(a^*)$. Therefore, we have

$$\mu(a_t) + 2r_t(a_t) \;\geq\; \mu'_t(a_t) + r_t(a_t) \;=\; UCB_t(a_t) \;\geq\; UCB_t(a^*) \;\geq\; \mu(a^*)$$

- It follows that

$$\Delta(a_t) \;=\; \mu(a^*) - \mu(a_t) \;\leq\; 2r_t(a_t) \;=\; 2\sqrt{\frac{2\log T}{n_t(a_t)}}$$

- One can then show that this algorithm also achieves the same regret bound as the previous one, namely, $O(\sqrt{Kt\log T})$.

4 Online Prediction as the Experts Problem

- Another popular approach for analyzing regret in online prediction, called the *experts problem*, uses the following model for decision making under uncertainty:

  > *We want to predict a sequence of events, where our only information about the future comes from recommendations of a set of "experts." The difficulty is that most, but not all, of these experts might be unreliable, even adversarial. In this "noisy" setting, our hope is to match the performance of the "best expert" with the benefit of hindsight.*

- The ideas developed for the experts problem have widespread applications throughout computer science, most notably, in Machine Learning, Optimization, and Game Theory.

- The general setting is very similar to the MAB problem.

  1. An algorithm needs to make a prediction at each of the T time steps.
  2. The algorithm has access to n "experts" and makes its prediction based on theirs (wisdom of the crowd).
  3. The algorithm can only use the *past* performance of the experts, and has no knowledge about their future performance.
  4. At the end of each time step, the algorithm incurs a loss depending on how wrong its prediction was. *Important: The algorithm learns the losses of all experts at the end of the time step.*
  5. The goal is to minimize the *total loss* over a horizon of T steps.

- **Differences from MAB.** While the two frameworks are motivated by the same stochastic prediction problem, there are some important modeling differences. In MAB, the loss function is assumed to be $i.i.d$ over all actions. By contrast, in the Experts problem, we assume that the loss functions of the experts can be arbitrarily set by an adversary.

- We will consider the case of *binary* (yes/no, 0/1) prediction for simplicity, although the result hold for the general loss functions.

- The experts' problem closely mimics several real-life decision making:

  1. **Weather Forecasting:** Every day, the algorithm needs to make a prediction whether it will rain on that day or not. At the end of each day, we will know which of the n experts were right and which were wrong. We want to build a prediction algorithm that does almost as well as the *best forecaster in hindsight.*

2. **Stock Market Prediction:** For simplicity, we consider a single evolving index X, e.g. the value of the Dow Jones. The index starts at some value $X(0) = 0$ and each day (round) t it can either go or down up by one, namely, either $X(t+1) = X(t) - 1$ or $X(t+1) = X(t) + 1$. The algorithm needs to predict at the beginning of each day which one of these two possibilities will occur. We want to minimize our total number of prediction mistakes over a fixed (but very large) horizon of T day.

4.1 Perfect Expert and Halving Algorithm

- In order to get some insight, let us first consider a simpler situation, where we assume that one of the experts is perfect, *although we don't know which one*; the perfect expert makes correct predictions for all T days.

- We note that the algorithm needs to make predictions on T days, where $T \gg n$ is essentially unbounded, so a priori there is no reason to believe that under our model of adversarial experts we can hope to do very well. In particular, can we avoid making mistakes on *most of the days*? Can the number of mistakes be *independent* of T?

- Fortunately, the following simple algorithm can guarantee that our algorithm makes at most $\log n$ mistakes, where n is the number of experts. (So, independent of T.)

The Halving Algorithm

1. Initialize the set S with all n agents, and day $i = 1$
2. On day i, the algorithm makes the prediction that the *majority* of the experts make.
3. If the algorithm's prediction is correct, do nothing.
4. Otherwise, remove from S all the experts whose prediction was incorrect.
5. increment i, and go to 2.

- To analyze this algorithm, let's consider the size of the expert pool that includes the perfect expert.

1. Initially, the size of the expert pool is n.
2. Each time the algorithm makes a mistake, at least half the experts in the pool are also wrong, and therefore removed. Thus, the remaining pool (which still contains the perfect expert) shrinks by at least half.
3. After $\log n$ mistakes, the expert pool shrinks to size 1, at which point the algorithm always follows the perfect expert, and therefore makes no more mistakes.

- **Homework Exercise.** Returning to the case of a perfect expert, namely, $m^* = 0$, suppose we use the following randomized strategy:

  1. Initially, all experts are in the pool S.
  2. In each round, choose an expert from S uniformly at random, and follow his advice; remove from S all experts whose prediction is found to be wrong at the end of the day.

  Show that the expected number of mistakes of this algorithm is at most $\ln n$.

4.2 The General Case

- We now return to the general setting where there is no perfect expert. Instead we have many different predictors that work well under different circumstances, and often disagree with each other. This is clearly the more realistic setting.

- For instance, there can be many economic indicators for predicting the stock market, which may often contradict each other. Or, in document classification, one rule may say "if article mentions football, classify as sports" and another might say "if it mentions dollar figures, classify as business." What should we do when article contains both football and dollars?

- In our analysis, we assume no prior knowledge about the accuracy of the experts, *except that one or more are believed to perform well.*

- Our goal is to somehow combine the predictions of many *imperfect* experts with the guarantee that it approaches the performance of the best with the benefit of the hindsight.

- **Adopting the Halving Algorithm:** First, it seems tempting to continue with the halving algorithm, with an appropriate modification. In particular, suppose that the best-in-the-hindsight expert makes m^* mistakes. Then, we could consider the following *Iterated Halving Algorithm.*

  1. Initially, all experts are in the pool S.
  2. In each round
     - Use the majority recommendation among the experts in S (breaking ties arbitrarily)
     - After seeing the true outcome, remove from S all experts whose prediction was incorrect
     - *If the pool becomes empty, we reset it by putting all experts back in it.*

- **Theorem.** The number of mistakes made by Iterated Halving Algorithm is at most

$$(m^* + 1)(1 + \log n) - 1$$

- **Proof.**

  1. Whenever S becomes empty, each expert has made at least one mistake since the last reset. Thus, the total number of resets is at most m^*.
  2. Between two consecutive resets, the earlier analysis of Halving Algorithm applies, and the algorithm makes at most $(1 + \log n)$ mistakes until S becomes empty.
  3. On the other hand, after the last reset, there can be at most $\log n$ resets—since otherwise S will become empty. This completes the proof.

4.3 The Weighted Majority Algorithm

- The number of mistakes by IHA are $\log n$ *multiplicative factor* larger than the optimal m^*. Can this be improved?

- One shortcoming of IHA is that each reset discards all the knowledge gained in the previous rounds. In particular, it does not differentiate between an expert that made only a few mistakes and one that was frequently wrong.

- This suggests the idea that instead of dividing the experts into *trustworthy* and (temporarily) *not trustworthy*, we should assign a weight to each expert that expresses our confidence in its prediction based on its performance so far.

- Specifically, the algorithm maintains a *weight* for each expert, initially set to 1, and updates it every time the expert makes a wrong prediction.

Weighted Majority Algorithm (WMA).

1. Initialize all weights to $w_i = 1$.
2. Given a set of predictions $x_1, x_2, \ldots, x_n$ by the experts, where each $x_i \in \{0, 1\}$, output the prediction with the highest total weight.
3. That is, suppose I is the set of experts predicting 1, and Z the set predicting 0. Then predict 1 if $\sum_{i \in I} w_i \ \geq \ \sum_{i \in Z} w_i$, and 0 otherwise.
4. Penalize each wrong expert by *halving* his weight.
5. Go to 2.

- **Theorem.** The number of mistakes M made by the WMA is no more than

$$2.41(m^* + \log n)$$

where m^* is the number of mistakes made by the *best expert*.

- **Proof.**

1. Let W denote the total weight of all the experts. Initially, $W = n$.
2. Every time the algorithm's prediction is incorrect, at least half the total experts' weight is reduced by $1/2$, since they were in the majority and wrong.
3. Thus, each mistake causes the total weight to decrease by factor $\frac{3}{4}$.
4. Suppose the algorithm makes M mistakes. Then, the total weight of all experts is

$$W \quad \leq \quad n(3/4)^M$$

5. On the other hand, the best expert makes at most m^* mistakes. Therefore, its weight is *at least* $1/2^{m^*}$.
6. Thus, we have $1/2^{m^*} \ \leq \ W \ \leq \ n(3/4)^M$, which gives us

$$M \leq (m^* + \log n)\frac{1}{\log(4/3)} \ \leq \ 2.41(m^* + \log n)$$

4.4 Improvements to WMA.

- One can improve the WMA in two ways:

1. Instead of making the prediction with the total weight, we can use the weights as probabilities and choose the outcome probabilistically.
2. Instead of penalizing the wrong experts by factor $1/2$, we can penalize each by factor beta.
3. With these modifications, we can prove the following mistake bound:

$$M \leq \{m^* \ln(1/\beta) + \ln n\}/(1 - \beta)$$

E.g. when $\beta = 3/4$, we can $M \approx 1.15m^* + 4 \ln n$.

4. By adjusting beta further dynamically, one can also get $M \leq m^* + \ln n + O(\sqrt{(m^* \ln n)})$.

4.5 Experts Problem Against Adversarial Bandits

- Let us now briefly discuss the challenges in generalizing the experts problem from the binary prediction to loss functions, which was the setting used in MAB problem.

- Specifically, we assume that each expert i makes a prediction at each time step t and incurs a loss $\ell_i^t \in [0,1]$; in the simpler version we analyzed earlier the loss is either 0 (correct prediction) or 1 (incorrect prediction). The general loss function has considerably broad applicability, where simple yes or no answers are insufficient. (Even modeling of weather prediction and stock market needs non-binary loss functions.)

- Based on this information, the algorithm chooses an expert $\sigma(t)$ at time t, incurring the loss of $\ell_{\sigma(t)}^t$. We want to design a policy (algorithm) that minimizes the total loss

$$\sum_{t=1}^{T} \ell_{\sigma(t)}^t$$

  As in the earlier online prediction problem, we compare this loss to the best loss achieved by an optimal algorithm with full knowledge of the sequence—equivalently, the "best in hindsight."

- If the loss function is under the control of an adversary (instead of dictated by a stochatic process as in MAB), then it is easy to construct an example showing that no algorithm can do well.

  1. The adversary chooses one "good" expert at each step, with loss of 0, and makes all others experts "bad" with loss of 1.
  2. In addition, the good expert is determined randomly at each step.
  3. Then, any algorithm incurs a loss of at least $(1-1/n)T$ regardless of its strategy, while a complete knowledge of the loss sequence leads to a loss of 0, by always picking the good expert in each step.

- This analysis however is unfair—it not only gives adversary full power to set the loss function, but also full knowledge of the sequence so that it can always choose the best expert. This is like comparing ourselves to a hypothetical stock trader who can always use tomorrow's prices to pick stock today.

- A more reasonable benchmark is to do what we did in analyzing MAB: require the optimal strategy to pick *one expert for the entire duration*, even with full knowledge of the future, while still allowing the adversary to set the loss function arbitrarily.

- More specifically, we will define the regret as

$$\sum_{t=1}^{T} \ell^t_{\sigma(t)} \quad - \quad \min_{i=1}^{n} \sum_{t=1}^{T} \ell^t_i$$

  The first term is the algorithm's loss at each step t, while the second term is the minimum total regret caused by any single expert.

- Even with this adjustment, one can show that *no deterministic* algorithm can hope to do well.

  1. In particular, given any algorithm A, at each step the adversary sets the loss of the expert chosen by the algorithm to 1, and loss of all other agents to 0.
  2. The algorithm incurs a loss of 1 in each step, for a total loss of T.
  3. On the other hand, there is only one non-zero loss value in each step, so the sum of all losses of all the experts over all times is only T. Thus, there must exist some agent for which $\sum_{t=1}^{T} \ell^t_i \leq T/n$.
  4. Thus, once again, the algorithm A's regret is $\geq T(1 - 1/n)$.

- In particular, this shows that the natural strategy of "follow the leader" (FTL) does not have good worst-case guarantee. (In FTL, the algorithm chooses at each step t the expert i whose total past loss is minimum, namely, the expert i that minimizes $\sum_{k=1}^{t-1} \ell^k_i$.)

- A way out of this negative-news mire is *randomization*. By allowing the algorithm to choose its expert at each step randomly precludes the adversary to cause the worst-case behavior discussed above. One can show that with this adaptation, a randomized version of WMA achieves expected regret of $O(\sqrt{T \ln n})$.

- The main idea is to use a randomized version of the FTL (follow the leader), where we choose an expert with prob. that depends exponentially on the total loss.