# Multi-armed bandits

## Part 1

# The basic problem

We have a collection of *K* advertisements we can visualize inside our web site page.

Each time  a new user enters the page, we select an ad and show it. Each visualization is named an *impression*.

We hope the user gives us a *click* on the ad. Indeed, advertisers pay us €1 per click.

We can have *T* impressions (the letter T suggests time).

Our goal is to maximize clicks.

If we could know which ad has the greatest probability to be clicked, we could select it at each round.

But we do not know these probabilities. Indeed, finding the max probability is the core of this problem.

We want to define a *policy*, i.e. a criterion to select an ad at each round.

We will implement the policy as a program, which will manage the task without any help while running.

So, we have to anticipate different scenarios which can happen during the game.

# First naïve attempt

Let N = 100 rounds and K = 2 ads, be them A and B.

We define this policy:

*Try A and B alternatively until one gets a click.*

*Afterwards, select forever the same ad.*

(Forever means until the end of the game)

As soon as an ad surpasses the other, we declare it as winner.

We assess it as the best, so it is rational to visualize it instead of the other.

Why is this policy too raw?

Example:

| A | | B | |
|---|---|---|---|
| Clicks | Impressions | Clicks | Impressions |
| 1 | 3 | 0 | 3 |

Why is it not reasonable to assign all remaining 94 impressions to ad A?

It is a premature decision. We are not confident enough to decide that A is the best one. Maybe it was simply luckier than B over the first 3 rounds.

We need more evidence.

How much evidence?

We define a refined policy:

*Try A and B for N times each one.*

*The winner is the ad which get more clicks.*

*Afterwards, select forever the winner.*

It sounds better than the first attempt.

We have a first phase of *exploration*, then a second phase of *exploitation*.

This algorithm has a *parameter* i.e. *N*. The policy designer has to choose a number. This is really a difficult task.

A small *N* (little exploration, large exploitation) risks making a premature decision.

A large *N* gives us larger chance to choose the "true" winner, i.e. the ad which has really the maximum success rate.

Yet, we do not only want to find the best ad, we want to find it as soon as possible.

If we choose 49 as the value for the parameter *N*, in practice we use this algorithm:

*Try A and B for the same number of times.*

Not very smart, indeed.

We have to balance exploration and exploitation.

This **exp-exp dilemma** is well-know since many decades and is extremely challenging in its most complex forms.

*Exploration implies sustaining cost because of choosing an ad which is not the best one. This has to be intended as a risky but necessary investment in knowledge acquisition.*

A critical point is that the right value for *N* depends of the nature of ads *A* and *B*. If one is much better than the other, a short exploration is enough to understand it is and we can quickly move to exploitation mode, gaining more clicks.

Otherwise, we need more exploration … but the cost of exploration is not very high, because the ad have similar performance.

# GREEDY policy

The problem is so messy that we simply give up with the attempt of choosing the right exploration rate in advance.

Instead, we design an *adaptive policy*: at round $t$ the ad to show will be selected depending of the history of the whole game up to round $t - 1$.

Let us try with the simplest adaptive strategy:

GREEDY Policy

*Try each ad once.*

*Afterwards, try the ad best performing up to now.*

Ties are solved random.

From the third round on, we compare performances of *A* and *B* and select the best until now.

| A | | B | |
|---|---|---|---|
| Clicks | Impressions | Clicks | Impressions |
| 2 | 10 | 1 | 4 |

In this state, at round 15 we choose *B* because 1/4 > 2/10.

If the user does not click, the comparison becomes 1/5 vs 2/10. It is a tie, at round 16 we select random. If we select *B* and it fails again, the balance is 1/6 vs 2/10, *A* wins and is selected at round 17.

Reasonable, but still not satisfactory.

In this state

| A | | B | |
|---|---|---|---|
| Clicks | Impressions | Clicks | Impressions |
| 1 | 1 | 0 | 1 |

we select A and from now on we will select it forever.

Even if *A* fails at each round, its success rate will decrease but never reaching zero. The rate of *B* equals zero, so

Rate(A) > rate(B) forever

Not very fair! Clearly, *B* is worth a new chance.

We could extend the initial exploration phase

GREEDY Policy with initial exploration:

*Try each ad N times.*

*Afterwards, try the ad best performing up to now.*

Yet, we again face the problem of choosing the right investment in exploration. To be fair, the problem is lighter than with the first naïve algorithm, because now the second phase allows a certain degree of exploration. Though, we feel something better must be possible.

Intuition: mixing exploration and exploitation phases using data from the process itself.

# EPSILON-GREEDY policy

This idea let us introduce the classic

EPSILON-GREEDY Policy:

Choose a value for parameter ε (epsilon).

*Try each ad once.*

*Afterwards, try the ad best performing up to now with probability 1 - ε (epsilon) or a random chosen ad with probability ε.*

This policy is really a random mix of two policies, i.e. GREEDY and

RANDOM Policy:

*At each round select an ad random.*

EPSILON-GREEDY minimizes the initial exploration, then alternates exploitation and exploration.

| A | | B | |
|---|---|---|---|
| Clicks | Impressions | Clicks | Impressions |
| 2 | 10 | 1 | 4 |

Let $\varepsilon = 0.1$, i.e. the exploration rate we chose is 10%.

The leader *B* is selected with probability 95% = 90% + 5%, *A* with probability 5%.

With $K = 4$ ads, probabilities would be 92.5% for the leader, 2.5% for each follower.

This policy is simple, too simple one ould say.

Indeed, it is difficult to demonstrate useful properties of this algorithm. Theory of EPSILON-GREEDY does not give very interesting guidelines.

The great problem is tuning the parameter $\varepsilon$.

Why did we choose 10% exploration rate instead of 1% or 5%? In practice, the designer tries a value like this and waits. After hours, days or months, he/she can decide to try another way.

Despite its evident limits, for many years this policy was implemented in a lot of real systems.

It is very simple to implement and often performs more or less as well as more complex algorithms.

Only recently policy designers have acquired tools clearly better then EPSILON-GREEDY. Get familiar with this policy if you aim at developing algorithms for optimizing online advertising campaigns: it could turn out as a useful quick-and-dirty solution for some problems.

# EPSILON-GREEDY implementation

EPSILON-GREEDY Policy:

*Read a value for parameter $\varepsilon$.*

*// For each ad, initialize the history*

*for i = 1, ..., K*

*clicks($ad_i$) = imps($ad_i$) = 0*

*// Show each ad once, record what happened*

*for t = 1, ..., K*

*show $ad_t$*

*imps($ad_t$) += 1*

*if the user clicked then clicks($ad_t$) += 1*

*ctr($ad_t$) = clicks($ad_t$) / imps($ad_t$)*

*// For each round after,*

*// decide whether to apply Greedy or Random policy*

*For t = K + 1 ... T*

*If (r > $\varepsilon$)*

*x = i that maximizes ctr($ad_i$) (solve ties random)*

*else*

*x = a random number in the range 1 .. K*

*// Show the selected ad, record what happened*

*show $ad_x$*

*imps($ad_x$) += 1*

*if the user clicked then clicks($ad_x$) += 1*

*ctr($ad_x$) = clicks($ad_x$) / imps($ad_x$)*

*// ctr stays for click through rate, the usual name for the ratio*

*// clicks / impressions*

# Adaptive EPSILON-GREEDY

This policy can perform well if well-tuned, i.e. if ε has assigned a "good" value. Yet, it can as well perform very badly.

Tuning the parameter is difficult and depends of what happens during the game. The parameter ε adaptively tunes the exploration rate, but requires adaptive tuning for itself. The problem raises again, even though in a better version.

The tuning of ε can be achieved in a wide range of ways.

The most common is to define a *scheduling*, i.e. a policy which assigns ε a value depending of time (round number).

Usually, those values decrease with time.

Examples of decreasing scheduling can be:

$\varepsilon(t) = 1 / t$

$\varepsilon(t) = 1 /$ square root of t

$\varepsilon(t) = 1 /$ logarithm of t

The idea is that theexploration rate must be reduce for two reasons:

1. With more time (rounds) past, we know more.
2. With less time in the future, we have less opportunity to exploit the accumulated knowledge.

A possible improvement could be linking the value of $\varepsilon$ not only to time but also to uncertainty:

$\varepsilon(t, u)$ decreasing in t, increasing in u

something like $\varepsilon(t, u) = u / t$

could work, if we can give a suitable definition of uncertainty.

Intuitively, uncertainty is greater if

- The leader ad has only a small edge on the second, third ... competitors. We are not much confident teh leader is really the best.
- Ads are maybe changing their “true” success rates (do not confuse them with observed ctr!)

# Meta-level EPSILON-GREEDY

A very interesting approach is to have EPSILON-GREEDY to tune the parameter of EPSILON-GREEDY.

We select a set of possible values for $\varepsilon$, e.g. {0.05, 0.10, 0.15}

A meta-$\varepsilon$ parameter is used to rule the competition among those values.

At each step, we select a value for $\varepsilon$ and apply EPSILON-GREEDY with that parameter. Not only do we remember the impression and the (possible) click for selected ad but also for the selected $\varepsilon$ value.

So doing, the meta parameter tune the exploration rate for finding the best parameter, while parameters do the same for ads. Of course, we still have a parameter to tune, but the overall performance is less sensitive to our choice.

# Multiple winner EPSILON-GREEDY

Usually, we have more than one slot in your web page where you can show an ad.

If we want to show $k < K$ ads, we can simply apply Epsilon-Greedy $k$ times.

We select the first winner as above.

Then we execute Epsilon-Greedy on $K - 1$ ads, excluding the previous winner. So we get a second winner.

We repeat until we have $k$ ads selected.

# Batch EPSILON-GREEDY

Often, we cannot select one winner ad each time it required. Simply, user arrivals on our site are so many (thousands per second) that we have not enough time to make the selection and show the selected ads on the page.

So, a round is intended as an interval of time during which many decisions will be to be made *using the same policy and data*.

During a round, we cannot update data, so the 1,000$^{th}$ ad will be selected without learning from the 999 previous decisions. Only after the round ends we can update data and make the next round more "informed".

Probably, the system delivering ads allows you to specify a frequency policy.

We mean that you can assign (via program):

ad 1 has weight 60%

ad 2 has weight 30%

ad 3 has weight 10%

The delivering system (let us call it the *ad server*) interprets these instructions and delivers the ads with the specified frequencies.

Imagine that to select each ad it generates a random integer number x in the range 1..100 and selects ad 1 if $1 \leq x \leq 60$, or ad 2 if $61 \leq x \leq 90$, or ad 3 if $91 \leq x \leq 100$.

The effect is similar but not equal to what would happen if you can control each single decision using up-to-date data, instead that a batch od decisions using invariant data.

Each ad server has its own technique for specifying ad selection during a round. If you cannot specify relative frequencies, you have to adapt Epsilon-Greedy to its specific technique. Possibly, this adaptation can turn out to be very difficult or even impossible in practice.

# Softmax policy

The Epsilon-Greedy policy makes each round a choice between two policies: Greedy and Random. So, exploration and exploitation are mixed yet remaining visibly distinct, this or that.

We can say Epsilon-Greedy uses a binary decisional criterion at the single-round level: 100% exploration or 0%. At the global level, the mix is continuous, i.e. exploration will be say 5.2% of the total.

Now we examine a policy which makes the exp-exp mixing continuous for each round.

The idea is to make ad A more likely to be selected than ad B, if A is performing better than B. More precisely, if A is slightly better than B, its probability to be selected should be slightly greater. If much better, much greater probability.

This idea is implemented by the Softmax policy. The name Softmax derives from the idea "choose the max performance, but in a soft way: not each time, only often".

A first, not well working, implementation could be:

Softmax Policy (naïve version)

*For each round t in 1..N*

*compute* $q_i$ *= ctr of ad* $a_i$ *// Measure quality*

*compute* $p_i = \frac{q_i}{\sum_j q_j}$

*select an ad according to probabilities* $p_i$

*show it and update its history (imps, clicks, ctr)*

Given probabilities, doing selection according to it means using a *roulette mechanism*, as previously seen speaking about ad servers.

This version of the algorithm is too naïve.

Imagine we have two ads. Until now, A is 3 times better than B. With this method, we select A 3 times more likely.

This is like using Epsilon-Greedy with exploration rate

$\varepsilon = 25\%$

This can be acceptable initially, but advancing in time appears to be excessive, a waste of rounds.

We refine the policy computing

$$p_i = \frac{e^{q_i}}{\sum_j e^{q_j}}$$

The fundamental meaning of the exponential function is expressing a growth process in which the growth rate is proportional to the current size.

So, if A has quality 3 times the quality of B, then the probability of selecting A will be not 3 times greater, but $e^3$times greater, i.e. 20.09 times greater.

The exponential transformation exalts the differences, prizing the best ads much more than proportionally.

Yet, not absolutely: a worse ad keeps some chance to be selected. In this sense the method chooses the (soft)max candidate.

Now let us see the true Softmax method which includes an adaptive mechanism which makes the exploration rate to change over time

Softmax Policy (naïve version)

*For each round t in 1..N*

*compute $q_i$ = ctr of ad $a_i$ // Measure quality*

*compute* $p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$

*select an ad according to probabilities $p_i$*

*show it and update its history (imps, clicks, ctr)*

Let us interpret

$$p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$$

The ad number *i* is assigned a probability which is exponential (exalted) in its quality, as before.

Now, the quality of each ad is divided by a parameter *T* (chosen by the algorithm designer, i.e. by us). The parameter is called *temperature*. It tunes exploration rate. The bigger the temperature, the bigger the exploration rate.

$$p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$$

To interpret formulas, it is often useful to check first what happens in extremum conditions.

If the temperature is very high, much greater than the qualities, then the exponents are very small. So, the exponents (the arguments of the exponential functions, $q_j/T$) are close to 0. Then, the exponentials themselves are close to 1 *and very similar each other*. High temperature means little attention to differences. Respective qualities are not really important.

*Each ad is assigned a probability close to 1 / K, uniformly*.

$$p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$$

Vice versa, if the temperature is very low, much lower than the qualities, then the exponents are very large. So, the exponents (the arguments of the exponential functions, $q_j/T$) are very large. Then, the exponentials themselves are very large, *but larger qualities are exalted much more than smaller qualities.*

Low temperature means great attention to differences. Respective qualities are extremely important.

*The best ad is assigned a probability close to 1, each other a probability close to 0.*

$$p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$$

At this point an extremely important idea emerges.

We start at round 1 with a very large *T* parameter. Differences are flattened and in practice the Softmax approximates the Random policy. Exploration rate is close to 1.

Then, we gradually reduce the temperature.

Differences in quality get progressively more important and exploration rate decreases.

We continue decreasing temperature and get very close to 0.

Progressively the Softmax approximates the Greedy policy.

$$p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$$

This method of gradually reducing temperature is named *simulated annealing* (for reasons which are interesting but not relevant now).

While time flows, Softmax moves from quasi-Random to quasi-Greedy. This fits very well with intuition: start exploring a lot, then decrease exploration rate, get closer and closer to pure exploitation towards the end of the game.

$$p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$$

Very intriguing.

Unfortunately, tuning parameter *T* is really an hard task. Anyway, if you manage to fine-tuning it, the method can be very effective.

Moreover, we will meet similar concepts later.

Implementation of Softmax is simple, but requires care to avoid arithmetical overflow when computing exponentials, which can be extremely large or extremely small numbers.