CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

April 14, 2021

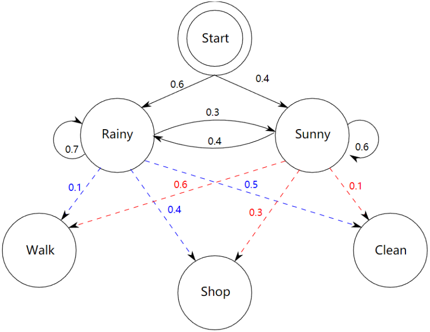
Outline

Outline

Hidden Markov Models

Model parameters:

- **initial distribution**
  $P(Z_1 = s) = \pi_s$
- **transition distribution**
  $P(Z_{t+1} = s' \mid Z_t = s) = a_{s,s'}$
- **emission distribution**
  $P(X_t = o \mid Z_t = s) = b_{s,o}$



Baum–Welch algorithm

Step 0 Initialize the parameters $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$

Step 1 (E-Step) Fixing the parameters, compute forward and backward messages for all sample sequences, then use these to compute $\gamma_s^{(n)}(t)$ and $\xi_{s,s'}^{(n)}(t)$ for each n, t, s, s'.

Step 2 (M-Step) Update parameters:

$$\pi_s \propto \sum_n \gamma_s^{(n)}(1), \quad a_{s,s'} \propto \sum_n \sum_{t=1}^{T-1} \xi_{s,s'}^{(n)}(t), \quad b_{s,o} \propto \sum_n \sum_{t:x_t=o} \gamma_s^{(n)}(t)$$

Step 3 Return to Step 1 if not converged

Viterbi Algorithm

Viterbi Algorithm

For each $s \in [S]$, compute $\delta_s(1) = \pi_s b_{s,x_1}$.

For each $t = 2, \ldots, T$,

- for each $s \in [S]$, compute

$$\delta_s(t) = b_{s,x_t} \max_{s'} a_{s',s} \delta_{s'}(t-1)$$

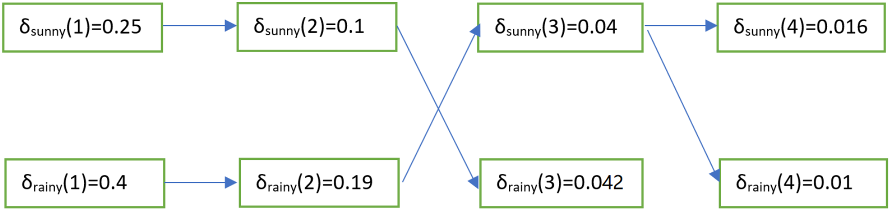
$$\Delta_s(t) = \operatorname*{argmax}_{s'} a_{s',s} \delta_{s'}(t-1)$$

Backtracking: let $z_T^* = \operatorname{argmax}_s \delta_s(T)$.
For each $t = T, \ldots, 2$: set $z_{t-1}^* = \Delta_{z_t^*}(t)$.

Output the most likely path $z_1^*, \ldots, z_T^*$.

Example

Arrows represent the "argmax", i.e. $\Delta_s(t)$.



The most likely path is **"rainy, rainy, sunny, sunny"**.

Viterbi Algorithm with missing data

Viterbi Algorithm with partial data $x_{1:T_0}$

For each $s \in [S]$, compute $\delta_s(1) = \pi_s b_{s,x_1}$.

For each $t = 2, \ldots, T$,

- for each $s \in [S]$, compute

$$\delta_s(t) = \begin{cases} b_{s,x_t} \max_{s'} a_{s',s}\delta_{s'}(t-1) & \text{if } t \le T_0 \\ \max_{s'} a_{s',s}\delta_{s'}(t-1) & \text{else} \end{cases}$$

$$\Delta_s(t) = \operatorname*{argmax}_{s'} a_{s',s}\delta_{s'}(t-1).$$

Backtracking: let $z_T^* = \operatorname{argmax}_s \delta_s(T)$.
For each $t = T, \ldots, 2$: set $z_{t-1}^* = \Delta_{z_t^*}(t)$.

Output the most likely path $z_1^*, \ldots, z_T^*$.

Outline

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

But many real-life problems are about **learning continuously**:

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

But many real-life problems are about **learning continuously**:

- make a prediction/decision
- receive some feedback
- repeat

Decision making

Problems we have discussed so far:

- start with a training dataset
- learn a predictor or discover some patterns

But many real-life problems are about **learning continuously**:

- make a prediction/decision
- receive some feedback
- repeat

Broadly, these are called **online decision making problems**.

Examples

Amazon/Netflix/MSN **recommendation systems**:

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move
- receive some reward (e.g. score a point) or loss (e.g. fall down)

Examples

Amazon/Netflix/MSN **recommendation systems**:

- a user visits the website
- the system recommends some products/movies/news stories
- the system observes whether the user clicks on the recommendation

Playing games (Go/Atari/StarCraft/...) or **controlling robots**:

- make a move
- receive some reward (e.g. score a point) or loss (e.g. fall down)
- make another move...

Two formal setups

We discuss two such problems:

- **multi-armed bandit** (this lecture)
- **reinforcement learning** (next lecture)

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- invariably it takes your money like a "bandit".

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- invariably it takes your money like a “bandit”.

Of course there are many slot machines in the casino

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- invariably it takes your money like a "bandit".

Of course there are many slot machines in the casino

- like a bandit with multiple arms (hence the name)

Mulit-armed bandits: motivation

Imagine going to a casino to play a slot machine

- invariably it takes your money like a "bandit".

Of course there are many slot machines in the casino

- like a bandit with multiple arms (hence the name)
- **if I can play for 10 times, which machines should I play?**

Applications

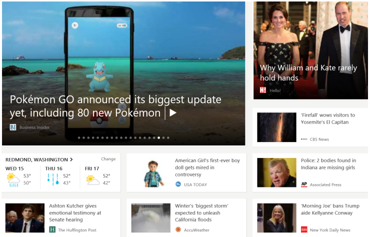
This simple model and its variants capture many real-life applications

- recommendation systems, each product/movie/news story is an arm

Applications

This simple model and its variants capture many real-life applications

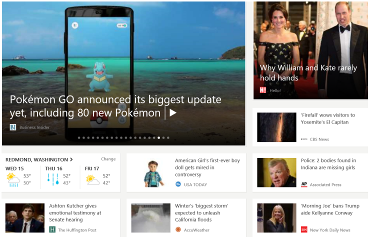
- recommendation systems, each product/movie/news story is an arm (Microsoft MSN indeed employs a variant of bandit algorithm)



Applications

This simple model and its variants capture many real-life applications

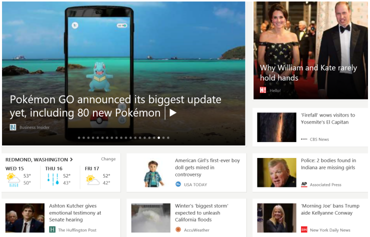
- recommendation systems, each product/movie/news story is an arm (Microsoft MSN indeed employs a variant of bandit algorithm)
- game playing, each possible move is an arm

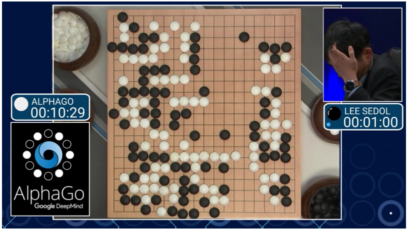


Applications

This simple model and its variants capture many real-life applications

- recommendation systems, each product/movie/news story is an arm (Microsoft MSN indeed employs a variant of bandit algorithm)
- game playing, each possible move is an arm (AlphaGo indeed has a bandit algorithm as one of the components)





Formal setup

There are K **arms** (actions/choices/...)

Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner:

Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner:
for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$

Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$

Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

Importantly, *learner does not observe rewards for arms not selected!*

Formal setup

There are K **arms** (actions/choices/...)

The problem proceeds in rounds between the environment and a learner: for each time $t = 1, \ldots, T$

- the environment decides the reward for each arm $r_{t,1}, \ldots, r_{t,K}$
- the learner picks an arm $a_t \in [K]$
- the learner observes the reward for arm a_t, i.e., r_{t,a_t}

Importantly, *learner does not observe rewards for arms not selected!*

This kind of limited feedback is now usually referred to as bandit feedback

Objective

What is the goal of this problem?

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful,

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*.

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*. A classic benchmark is

$$\max_{a \in [K]} \sum_{t=1}^{T} r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*. A classic benchmark is

$$\max_{a\in[K]} \sum_{t=1}^{T} r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm

So we want to minimize

$$\max_{a\in[K]} \sum_{t=1}^{T} r_{t,a} - \sum_{t=1}^{T} r_{t,a_t}$$

Objective

What is the goal of this problem?

Maximizing total rewards $\sum_{t=1}^{T} r_{t,a_t}$ seems natural

But the absolute value of rewards is not meaningful, instead we should compare it to some *benchmark*. A classic benchmark is

$$\max_{a\in[K]} \sum_{t=1}^{T} r_{t,a}$$

i.e. the largest reward one can achieve by always playing a fixed arm

So we want to minimize

$$\max_{a\in[K]} \sum_{t=1}^{T} r_{t,a} - \sum_{t=1}^{T} r_{t,a_t}$$

This is called the **regret**: *how much I regret for not sticking with the best fixed arm in hindsight?*

Environments

How are the rewards generated by the environments?

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution
- they could be generated even completely arbitrarily/adversarially

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution
- they could be generated even completely arbitrarily/adversarially

We focus on a simple setting:

- rewards of arm a are i.i.d. samples of $\text{Ber}(\mu_a)$, that is, $r_{t,a}$ is 1 with prob. μ_a, and 0 with prob. $1 - \mu_a$, independent of anything else.

Environments

How are the rewards generated by the environments?

- they could be generated via some fixed distribution
- they could be generated via some changing distribution
- they could be generated even completely arbitrarily/adversarially

We focus on a simple setting:

- rewards of arm a are i.i.d. samples of $\text{Ber}(\mu_a)$, that is, $r_{t,a}$ is 1 with prob. μ_a, and 0 with prob. $1-\mu_a$, independent of anything else.
- each arm has a different mean $(\mu_1, \ldots, \mu_K)$; the problem is essentially about finding the best arm $\mathrm{argmax}_a\, \mu_a$ as quickly as possible

Empirical means

Let $\hat{\mu}_{t,a}$ be the **empirical mean** of arm a up to time t:

$$\hat{\mu}_{t,a} = \frac{1}{n_{t,a}} \sum_{\tau \leq t : a_\tau = a} r_{\tau,a}$$

where

$$n_{t,a} = \sum_{\tau \leq t} \mathbb{I}[a_\tau == a]$$

is the **number of times** we have picked arm a.

Empirical means

Let $\hat{\mu}_{t,a}$ be the **empirical mean** of arm a up to time t:

$$\hat{\mu}_{t,a} = \frac{1}{n_{t,a}} \sum_{\tau \leq t : a_\tau = a} r_{\tau,a}$$

where

$$n_{t,a} = \sum_{\tau \leq t} \mathbb{I}[a_\tau == a]$$

is the **number of times** we have picked arm a.

Concentration: $\hat{\mu}_{t,a}$ should be close to μ_a if $n_{t,a}$ is large

Exploitation only

Greedy

Pick each arm once for the first K rounds.

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \hat{\mu}_{t-1,a}$

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)
- suppose the alg. first pick arm 1 and see reward 0, then pick arm 2 and see reward 1 (this happens with decent probability)

Exploitation only

Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$, pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

What's wrong with this greedy algorithm?

Consider the following example:

- $K = 2, \mu_1 = 0.6, \mu_2 = 0.5$ (so arm 1 is the best)
- suppose the alg. first pick arm 1 and see reward 0, then pick arm 2 and see reward 1 (this happens with decent probability)
- the algorithm will never pick arm 1 again!

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good
- on the other hand we need to explore all arms often enough in order to figure out which one is better

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good
- on the other hand we need to explore all arms often enough in order to figure out which one is better
- so each time we need to ask: *do I explore or exploit? and how?*

The key challenge

All bandit problems face the same **dilemma**:

Exploitation vs. Exploration trade-off

- on one hand we want to exploit the arms that we think are good
- on the other hand we need to explore all arms often enough in order to figure out which one is better
- so each time we need to ask: *do I explore or exploit? and how?*

We next discuss **three ways** to trade off exploration and exploitation for our simple multi-armed bandit setting.

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

Exploitation phase: for the remaining $T - T_0$ rounds, stick with the empirically best arm $\mathrm{argmax}_a\ \hat{\mu}_{T_0,a}$

A natural first attempt

Explore–then–Exploit

Input: a parameter $T_0 \in [T]$

Exploration phase: for the first T_0 rounds, pick each arm for T_0/K times

Exploitation phase: for the remaining $T - T_0$ rounds, stick with the empirically best arm $\text{argmax}_a \ \hat{\mu}_{T_0,a}$

Parameter T_0 clearly controls the exploration/exploitation trade-off

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

- not clear how to tune the hyperparameter T_0

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

- not clear how to tune the hyperparameter T_0
- in the exploration phase, even if an arm is clearly worse than others based on a few pulls, it's still pulled for T_0/K times

Issues of Explore–then–Exploit

It's pretty reasonable, but the disadvantages are also clear:

- not clear how to tune the hyperparameter T_0
- in the exploration phase, even if an arm is clearly worse than others based on a few pulls, it's still pulled for T_0/K times
- clearly it won't work if the environment is changing

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \; \hat{\mu}_{t-1,a}$

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \text{argmax}_a \; \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \operatorname{argmax}_a \ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \operatorname{argmax}_a \; \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a\ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ
- same uniform exploration

A slightly better algorithm

ϵ-Greedy

Pick each arm once for the first K rounds.

For $t = K + 1, \ldots, T$,

- with probability ϵ, explore: pick an arm uniformly at random
- with probability $1 - \epsilon$, exploit: pick $a_t = \mathrm{argmax}_a \ \hat{\mu}_{t-1,a}$

Pros

- always exploring and exploiting
- applicable to many other problems
- first thing to try usually

Cons

- need to tune ϵ
- same uniform exploration

Is there a *more adaptive* way to explore?

More adaptive exploration

A simple modification of “Greedy” leads to the well-known:

Upper Confidence Bound (UCB) algorithm

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \mathrm{argmax}_a \ \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \mathrm{argmax}_a\ \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathsf{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathrm{UCB}_{t,a}$ where

$$\mathrm{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathrm{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathsf{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)
- a parameter-free algorithm,

More adaptive exploration

A simple modification of "Greedy" leads to the well-known:

Upper Confidence Bound (UCB) algorithm

For $t = 1, \ldots, T$, pick $a_t = \operatorname{argmax}_a \ \mathsf{UCB}_{t,a}$ where

$$\mathsf{UCB}_{t,a} \triangleq \hat{\mu}_{t-1,a} + 2\sqrt{\frac{\ln t}{n_{t-1,a}}}$$

- the first term in $\mathsf{UCB}_{t,a}$ represents exploitation, while the second (bonus) term represents exploration
- the bonus term is large if the arm is not pulled often enough, which encourages exploration (adaptive due to the first term)
- a parameter-free algorithm, and *it enjoys optimal regret!*

Upper confidence bound

Why is it called upper confidence bound?

Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$\mu_a \leq \mathsf{UCB}_{t,a}$$

so $\mathsf{UCB}_{t,a}$ is indeed an upper bound on the true mean.

Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$\mu_a \leq \mathsf{UCB}_{t,a}$$

so $\mathsf{UCB}_{t,a}$ is indeed an upper bound on the true mean.

Another way to interpret UCB, "**optimism in face of uncertainty**":

Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$\mu_a \leq \mathsf{UCB}_{t,a}$$

so $\mathsf{UCB}_{t,a}$ is indeed an upper bound on the true mean.

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment is unknown due to randomness (**uncertainty**)

Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$\mu_a \leq \mathrm{UCB}_{t,a}$$

so $\mathrm{UCB}_{t,a}$ is indeed an upper bound on the true mean.

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment is unknown due to randomness (**uncertainty**)
- just pretend it's the most preferable one among all plausible environments (**optimism**)

Upper confidence bound

Why is it called upper confidence bound?

One can prove that with high probability,

$$\mu_a \leq \mathrm{UCB}_{t,a}$$

so $\mathrm{UCB}_{t,a}$ is indeed an upper bound on the true mean.

Another way to interpret UCB, "**optimism in face of uncertainty**":

- true environment is unknown due to randomness (**uncertainty**)
- just pretend it's the most preferable one among all plausible environments (**optimism**)

This principle is useful for many other bandit problems.