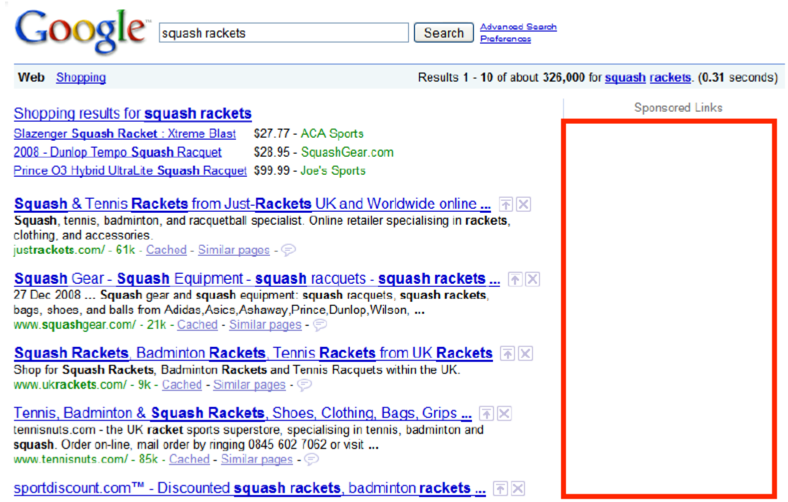
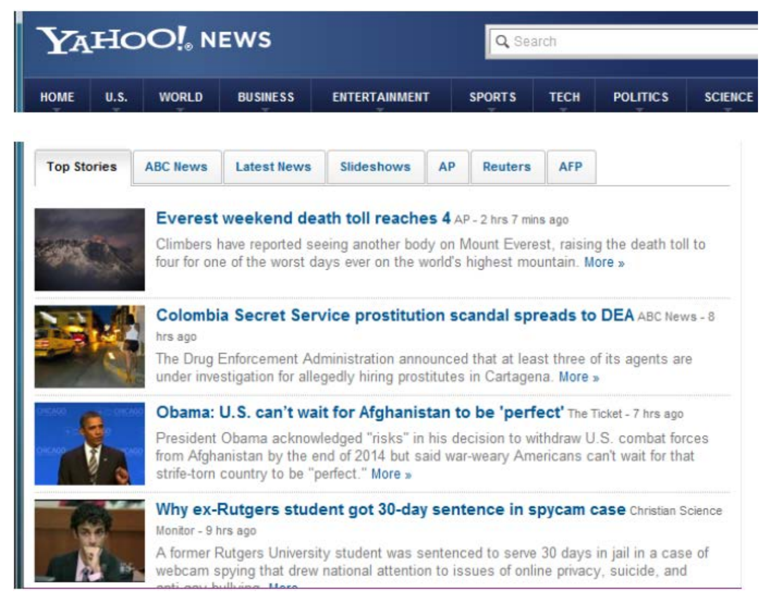
Learning through Experimentation

- **Web advertising**
  - We discussed how to match advertisers to queries in real-time
  - **But we did not discuss how to estimate the CTR (Click-Through Rate)**
- **Recommendation engines**
  - We discussed how to build recommender systems
  - **But we did not discuss the cold start problem**

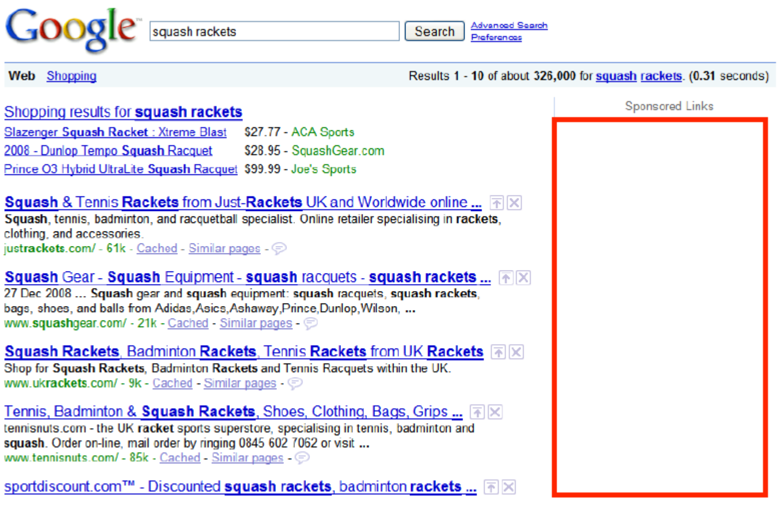


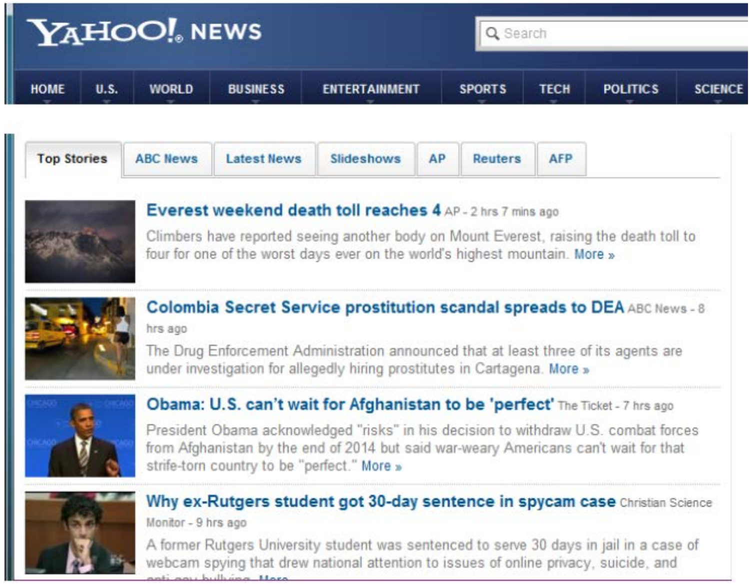


Learning through Experimentation

- What do **CTR** and **cold start** have in common?
- With every **ad we show/ product we recommend** we gather more data about the **ad/product**

- **Theme: Learning through experimentation**





Example: Web Advertising

- **Google's goal: Maximize revenue**
- **The old way: Pay by impression (CPM)**
  - **Best strategy: Go with the highest bidder**
    - But this ignores "effectiveness" of an ad
- **The new way: Pay per click! (CPC)**
  - **Best strategy: Go with expected revenue**
  - What's the expected revenue of ad ***a*** **for query *q***?
  - $\mathbf{E[revenue_{a,q}] = P(click_a \mid q) * amount_{a,q}}$

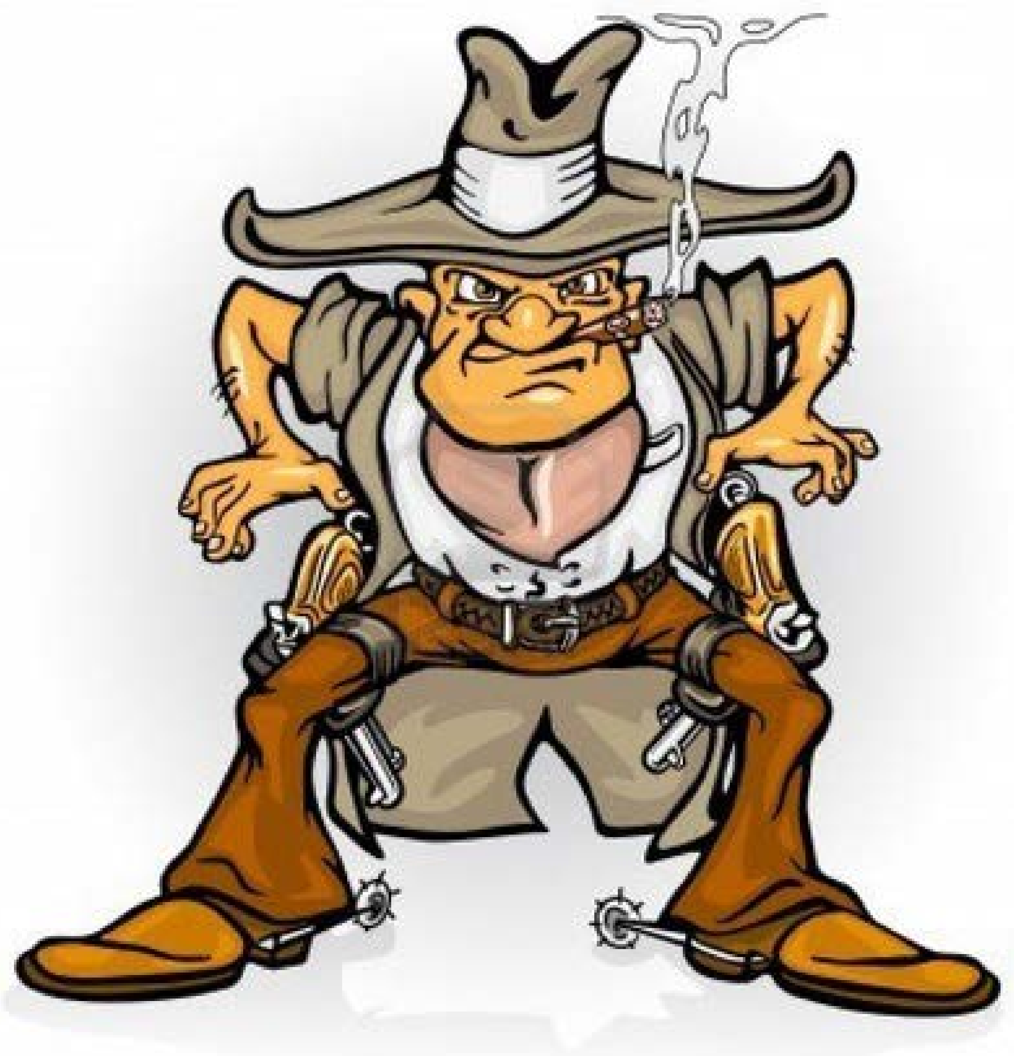
Prob. user will click on ad **a** given that she issues query **q**
(Unknown! Need to gather information)

Bid amount for ad **a** on query **q**
(Known)

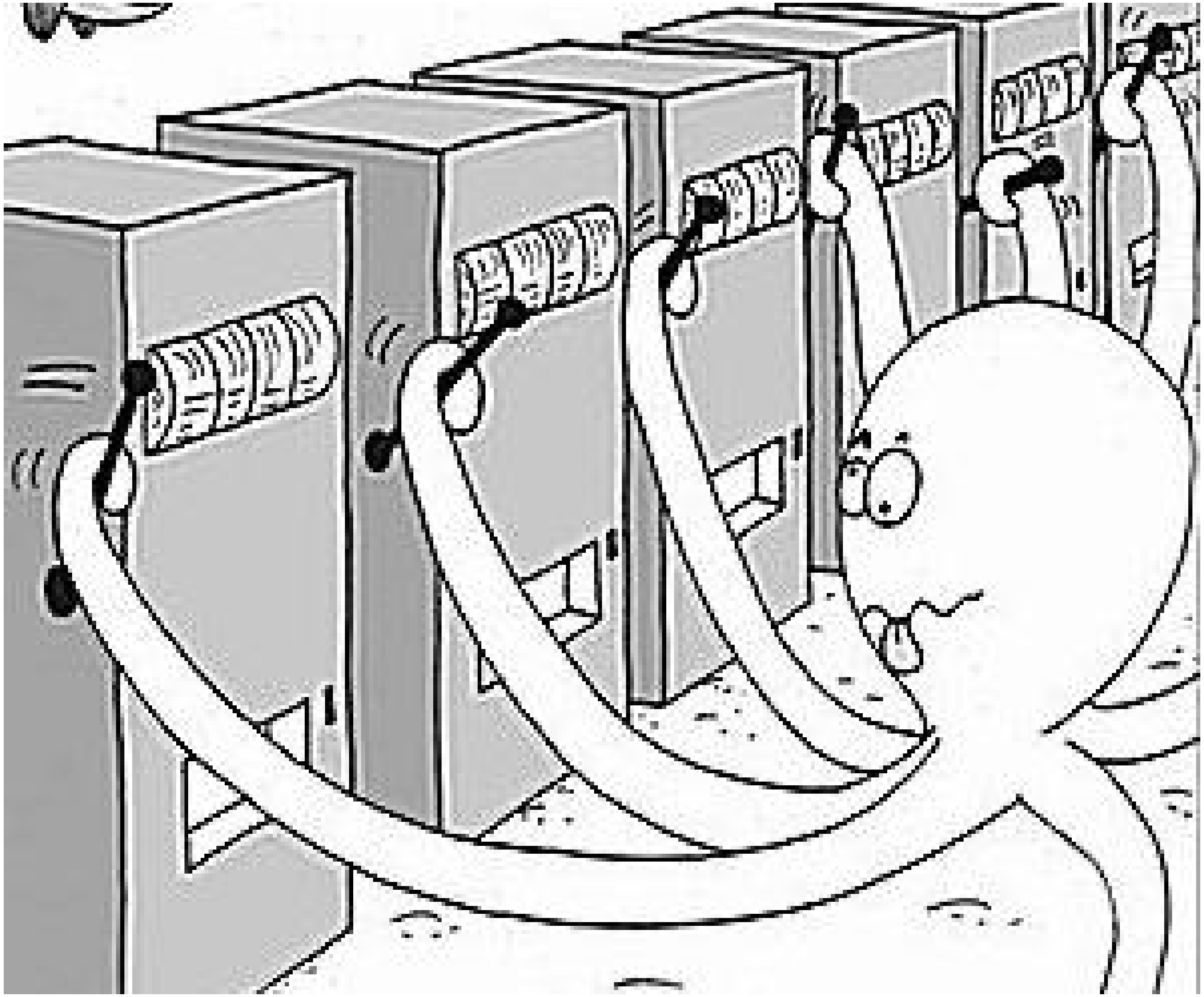
Other Applications

- **Clinical trials:**
  - Investigate effects of different treatments while minimizing patient losses
- **Adaptive routing:**
  - Minimize delay in the network by investigating different routes
- **Asset pricing:**
  - Figure out product prices while trying to make most money

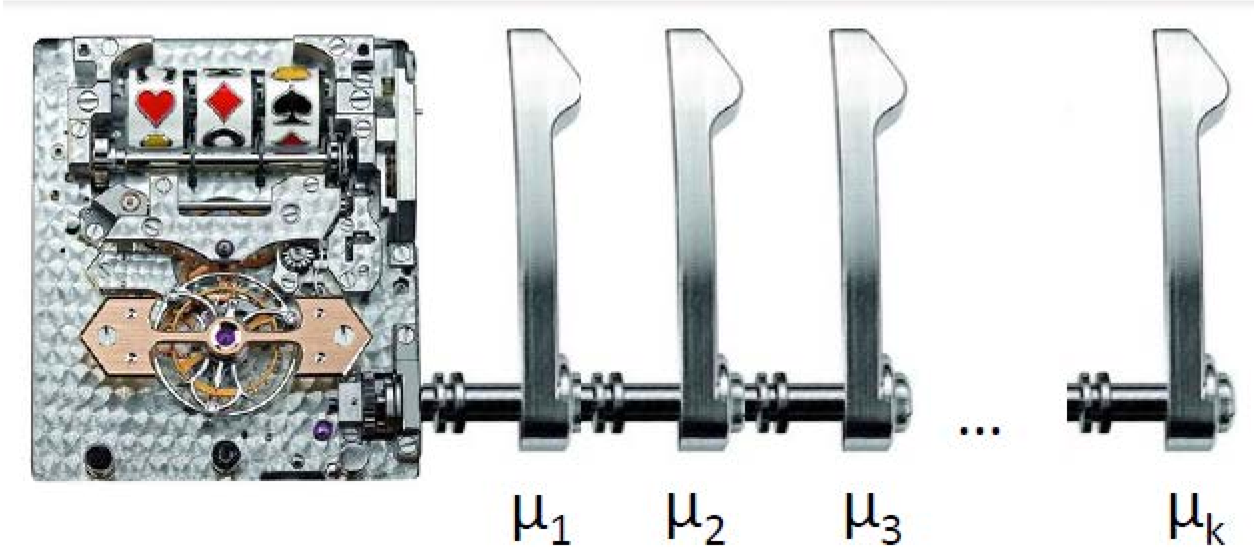
Approach: Bandits



Approach: Multiarmed Bandits

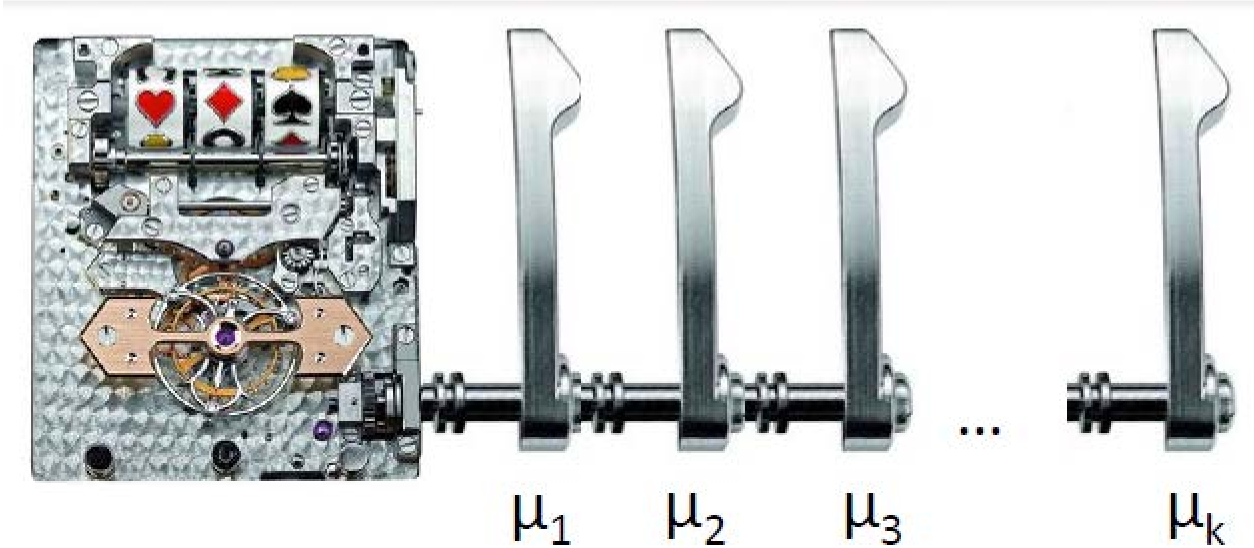


k-Armed Bandit



- **Each arm *a***
  - **Wins** (reward=**1**) with fixed (unknown) prob. μ_a
  - **Loses** (reward=**0**) with fixed (unknown) prob. $1-\mu_a$
- All draws are independent given $\mu_1 \ldots \mu_k$
- **How to pull arms to maximize total reward?**

k-Armed Bandit



- **How does this map to our setting?**
- Each **query** is a **bandit**
- Each **ad** is an **arm**
- **We want to estimate the arm's probability of winning μ_a (i.e., ad's the CTR μ_a)**
- Every time we pull an arm we do an 'experiment'

Stochastic k-Armed Bandit

The setting:

- Set of ***k*** choices (arms)
- Each choice ***a*** is associated with unknown probability distribution P_a supported in **[0,1]**
- We play the game for ***T*** rounds
- In each round ***t***:
  - **(1)** We pick some arm ***j***
  - **(2)** We obtain random sample X_t from P_j
    - Note reward is independent of previous draws
- **Our goal is to maximize** $\sum_{t=1}^{T} X_t$
- **But we don't know** μ_a**!** But every time we pull some arm ***a*** we get to learn a bit about μ_a

Online Optimization

- Online optimization with limited feedback

Choices	X_1	X_2	X_3	X_4	X_5	X_6	…
a_1					1	1	
a_2	0		1	0			
…							
a_k		0					

Time

- Like in online algorithms:
  - Have to make a choice each time
  - But we only receive information about the chosen action

Solving the Bandit Problem

- **Policy**: a strategy/rule that in each iteration tells me which arm to pull
  - Hopefully policy depends on the history of rewards

- **How to quantify performance of the algorithm? Regret!**

Performance Metric: Regret

- Let be μ_a the mean of P_a
- Payoff/reward of **best arm**: $\mu^* = \max_a \mu_a$
- Let $i_1, i_2 \dots i_T$ be the sequence of arms pulled
- Instantaneous **regret** at time t: $r_t = \mu^* - \mu_{a_t}$
- **Total regret:**

$$R_T = \sum_{t=1}^{T} r_t$$

- **Typical goal: Want a policy (arm allocation strategy) that guarantees: $\frac{R_T}{T} \to 0$ as $T \to \infty$**

Allocation Strategies

- **If we knew the payoffs, which arm would we pull?**

$$\text{Pick } \arg\max_a \mu_a$$

- **What if we only care about estimating payoffs μ_a?**
  - Pick each of k arms equally often: $\frac{T}{k}$
  - **Estimate:** $\widehat{\mu_a} = \frac{k}{T}\sum_{j=1}^{T/k} X_{a,j}$
  - **Regret:** $R_T = \frac{T}{k}\sum_a^k(\mu^* - \mu_a)$

Bandit Algorithm: First try

- **Regret is defined in terms of average reward**
- **So, if we can estimate avg. reward we can minimize regret**
- **Consider algorithm: *Greedy***
  Take the action with the highest avg. reward
  - **Example:** Consider 2 actions
    - **A1** reward 1 with prob. 0.3
    - **A2** has reward 1 with prob. 0.7
  - Play **A1**, get reward 1
  - Play **A2**, get reward 0
  - Now avg. reward of **A1** will never drop to 0, and we will never play action **A2**

Exploration vs. Exploitation

- **The example illustrates a classic problem in decision making:**
  - We need to trade off **exploration** (gathering data about arm payoffs) and **exploitation** (making decisions based on data already gathered)
- **The Greedy does not explore sufficiently**
  - **Exploration:** Pull an arm we never pulled before
  - **Exploitation:** Pull an arm $\boldsymbol{a}$ for which we currently have the highest estimate of $\boldsymbol{\mu_a}$

Optimism

- The problem with our **Greedy** algorithm is that it is **too certain** in the estimate of $\boldsymbol{\mu_a}$
  - When we have seen a single reward of 0 we shouldn't conclude the average reward is 0

- **Greedy does not explore sufficiently!**

New Algorithm: Epsilon-Greedy

Algorithm: Epsilon-Greedy

- **For t=1:T**
  - Set $\boldsymbol{\varepsilon_t = O(1/t)}$
  - **With prob. $\boldsymbol{\varepsilon_t}$: Explore** by picking an arm chosen uniformly at random
  - **With prob. $\boldsymbol{1 - \varepsilon_t}$: Exploit** by picking an arm with highest empirical mean payoff
- **Theorem** **[Auer et al. '02]**
  For suitable choice of $\boldsymbol{\varepsilon_t}$ it holds that
  $$R_T = O(k \log T) \Rightarrow \frac{R_T}{T} = O\left(\frac{k \log T}{T}\right) \to 0$$

Issues with Epsilon Greedy

- What are some issues with **Epsilon Greedy**?
  - **"Not elegant"**: Algorithm explicitly distinguishes between exploration and exploitation
  - **More importantly:** Exploration makes **suboptimal choices** (since it picks any arm equally likely)
- **Idea:** When exploring/exploiting we need to **compare** arms

Comparing Arms

- **Suppose we have done experiments:**
  - **Arm 1**: 1 0 0 1 1 0 0 1 0 1
  - **Arm 2**: 1
  - **Arm 3**: 1 1 0 1 1 1 0 1 1 1
- **Mean arm values:**
  - **Arm 1**: 5/10, **Arm 2**: 1, **Arm 3**: 8/10
- **Which arm would you pick next?**
- **Idea: Don't just look at the mean (that is, expected payoff) but also the confidence!**

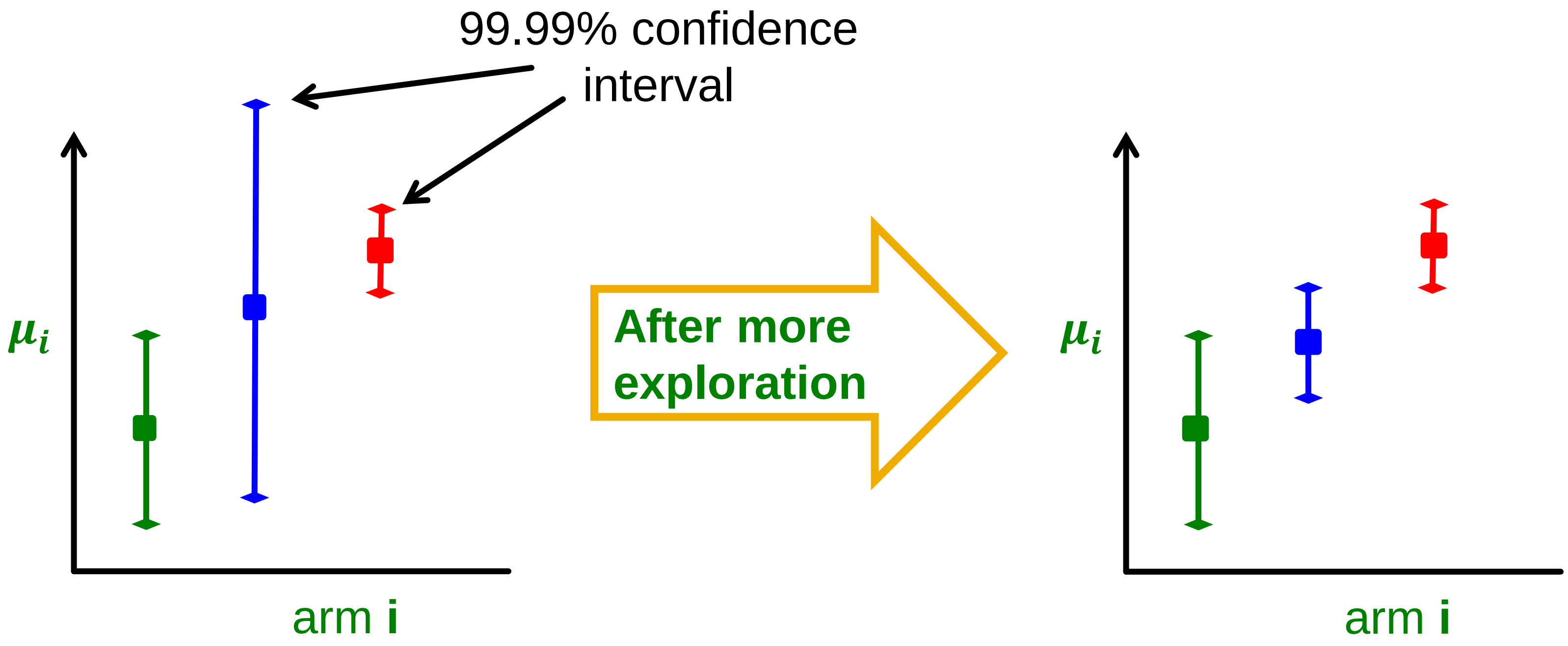
Confidence Intervals (1)

- **A confidence interval is a range of values within which we are sure the mean lies with a certain probability**
  - We could believe $\boldsymbol{\mu_a}$ is within [0.2,0.5] with probability 0.95
  - If we would have tried an action less often, our estimated reward is less accurate so the confidence interval is larger
  - Interval shrinks as we get more information (try the action more often)

Confidence Intervals (2)

- **Assuming we know the confidence intervals**
- **Then, instead of trying the action with the highest mean we can try the action with the highest upper bound on its confidence interval**
- This is called an **optimistic policy**
  - We believe an action is as good as possible given the available evidence

Confidence Based Selection



Calculating Confidence Bounds

Suppose we fix arm *a*:

- Let $Y_{a,1} \ldots Y_{a,m}$ be the payoffs of arm ***a*** in the first ***m*** trials
  - So, $Y_{a,1} \ldots Y_{a,m}$ are **i.i.d.** rnd. vars. taking values in **[0,1]**
- **Mean payoff of arm *a*:** $\mu_a = E[Y_{a,m}]$
- **Our estimate:** $\widehat{\mu_{a,m}} = \frac{1}{m}\sum_{\ell=1}^{m} Y_{a,\ell}$
- Want to find b such that with high probability $\left|\mu_a - \widehat{\mu_{a,m}}\right| \le b$
  - Also want b to be as small as possible (**why?**)
- **Goal: Want to bound** $\mathrm{P}\left(\left|\mu_i - \widehat{\mu_{a,m}}\right| \le b\right)$

Hoeffding's Inequality

- **Hoeffding's inequality:**
  - Let $X_1 \dots X_m$ be **i.i.d.** rnd. vars. taking values in **[0,1]**
  - Let $\mu = E[X]$ and $\widehat{\mu_m} = \frac{1}{m}\sum_{\ell=1}^{m} X_\ell$
  - **Then:** $\mathrm{P}(|\mu - \widehat{\mu_m}| \leq b) \leq 2\, exp(-2b^2 m) = \delta$
- **To find out the confidence interval b (for a given confidence level δ) we solve**
  - $2e^{-2b^2 m} \leq \delta$ **then** $-2b^2 m \leq \ln(\delta/2)$
  - **So:** $b \geq \sqrt{\frac{\ln\left(\frac{2}{\delta}\right)}{2\, m}}$

UCB1 Algorithm

- **UCB1 (Upper confidence sampling) algorithm**
  - **Set:** $\widehat{\mu_1} = \cdots = \widehat{\mu_k} = 0$ and $m_1 = \cdots = m_k = 0$
    - $\widehat{\mu_a}$ **is our estimate of payoff of arm** i
    - m_a **is the number of pulls of arm** i **so far**
  - **For** ***t = 1:T***
    - **For each arm** ***a*** **calculate:** $UCB(a) = \widehat{\mu_a} + \alpha\sqrt{\frac{2\ln t}{m_a}}$

      Upper confidence interval (Hoeffding's inequality)
    - **Pick arm** $j = arg\,max_a\, UCB(a)$
    - **Pull arm** ***j*** **and observe** y_t
    - **Set:** $m_j \leftarrow m_j + 1$ **and** $\widehat{\mu_j} \leftarrow \frac{1}{m_j}(y_t + (m_j - 1)\,\widehat{\mu_j})$

UCB1: Discussion

$$b \geq \sqrt{\frac{\ln\left(\frac{2}{\delta}\right)}{2\,m}}$$

- $UCB(a) = \widehat{\mu_a} + \alpha\sqrt{\frac{2\ln t}{m_a}}$
  - Confidence interval **grows** with the total number of actions t we have taken
  - But **shrinks** with the number of times m_a we have tried arm a
  - This ensures each arm is tried infinitely often but still balances exploration and exploitation
  - α plays the role of δ: $\alpha = f\left(\frac{2}{\delta}\right)$ $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$
- **Optimism in face of uncertainty**
  - The algorithm believes that it can obtain extra rewards by reaching the unexplored parts of the state space

Performance of UCB1

- **Theorem [Auer et al. 2002]**
  - Suppose optimal mean payoff is $\boldsymbol{\mu^* = \max_a \mu_a}$
  - And for each arm let $\boldsymbol{\Delta_a = \mu^* - \mu_a}$
  - **Then it holds that**

$$E[R_T] = \left[8 \sum_{a:\mu_a<\mu^*} \frac{\ln T}{\Delta_a}\right] + \left(1 + \frac{\pi^2}{3}\right)\left(\sum_{i=a}^{k} \Delta_a\right)$$

$O(k \ln T)$

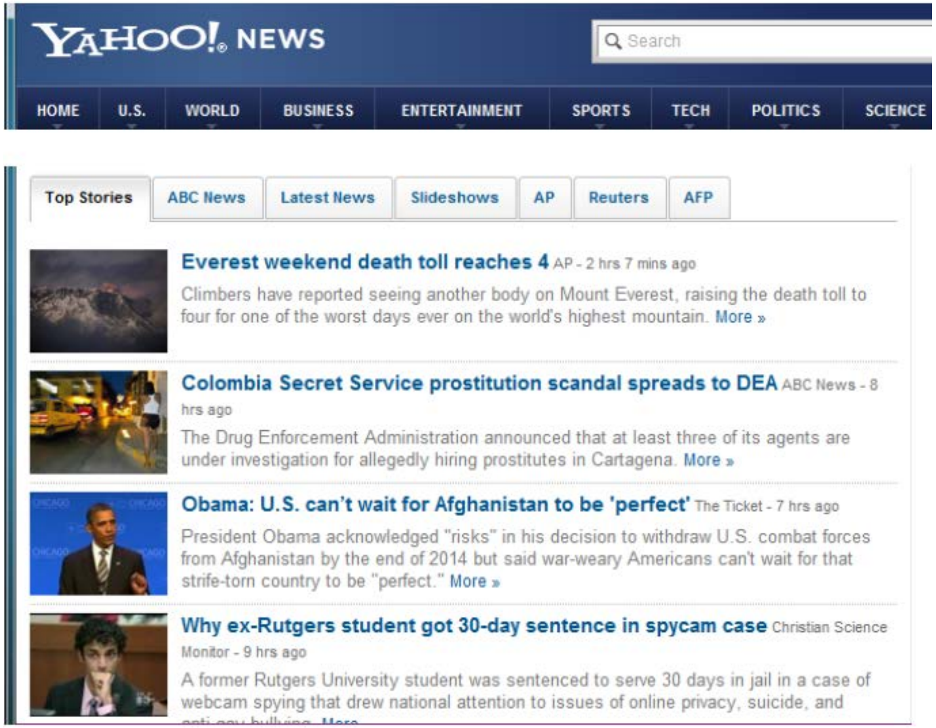
$O(k)$

- **So:** $O\left(\frac{R_T}{T}\right) = k\,\frac{\ln T}{T}$

Summary so far

- ***k***-armed bandit problem as a formalization of the exploration-exploitation tradeoff
- Analog of online optimization (e.g., SGD, BALANCE), but with **limited feedback**
- **Simple algorithms are able to achieve no regret (in the limit)**
  - Epsilon-greedy
  - UCB (Upper Confidence Sampling)

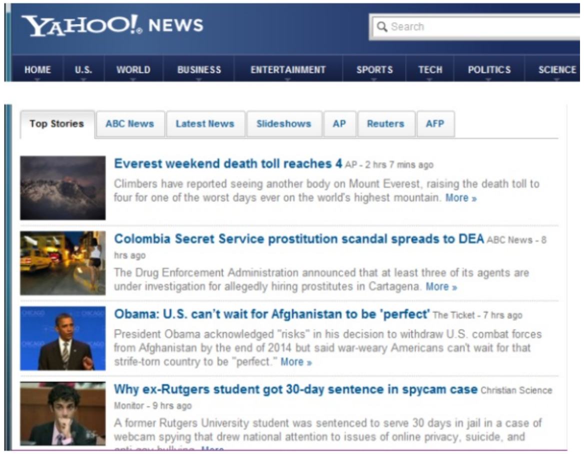
Back to News Recommendation



- Every round receive **context** [Li et al., WWW '10]
  - **Context:** User features, articles view before
- **Model for each article's click-through rate**

News Recommendation

- **Feature-based exploration:**
  - **Select articles to serve users based on contextual information about the user and the articles**
  - **Simultaneously adapt article selection strategy based on user-click feedback to maximize total number of user clicks**



Contextual Bandits

- **Contextual bandit algorithm in round *t***
  - **(1) Algorithm** observes user $\boldsymbol{u_t}$ and a set $\boldsymbol{A}$ of arms together with their features $\boldsymbol{x_{t,a}}$
    - Vector $\boldsymbol{x_{t,a}}$ summarizes both the user $\boldsymbol{u_t}$ and arm $\boldsymbol{a}$
    - We call vector $\boldsymbol{x_{t,a}}$ the **context**
  - **(2)** Based on payoffs from previous trials, algorithm chooses arm $\boldsymbol{a \in A}$ and receives payoff $\boldsymbol{r_{t,a}}$
    - **Note only feedback for the chosen *a* is observed**
  - **(3)** Algorithm improves arm selection strategy with each observation $(\boldsymbol{x_{t,a}}, \boldsymbol{a}, \boldsymbol{r_{t,a}})$

LinUCB Algorithm (1)

- Payoff of arm ***a***: $E\left[r_{t,a} | x_{t,a}\right] = x_{t,a}^{\mathrm{T}} \cdot \theta_a^*$
  - $\boldsymbol{x_{t,a}}$ … ***d***-dimensional feature vector
  - $\boldsymbol{\theta_a^*}$… unknown coefficient vector we aim to learn
    - Note that $\boldsymbol{\theta_a^*}$ are not shared between different arms!
- **What's the difference between LinUCB, UCB1?**
  - UCB2 directly estimates $\boldsymbol{\mu_a}$ through experimentation (without any knowledge about arm $\boldsymbol{a}$)
  - LinUCB estimates $\boldsymbol{\mu_a}$ by regression $\boldsymbol{\mu_a = x_{t,a}^T \cdot \theta_a^*}$
    - The hope is that we will be able to learn faster as we consider the context $\boldsymbol{x_a}$ (user, ad) of arm $\boldsymbol{a}$

LinUCB Algorithm (2)

- Payoff of arm ***a***: $E\left[r_{t,a} | x_{t,a}\right] = x_{t,a}^{\mathrm{T}} \cdot \theta_a^*$
  - $\boldsymbol{x_{t,a}}$ … ***d***-dimensional feature vector
  - $\boldsymbol{\theta_a^*}$… unknown coefficient vector we aim to learn
- **How to estimate $\boldsymbol{\theta_a}$?**
  - $\boldsymbol{D_a}$… $m \times d$ matrix of $\boldsymbol{m}$ training inputs $[\boldsymbol{x_{a,t}}]$
  - $\boldsymbol{b_a}$… $\boldsymbol{m}$-dim. vector of responses to ***a*** (click/no-click)
  - **Linear regression solution to $\boldsymbol{\theta_a}$ is then**
    - $\widehat{\boldsymbol{\theta}}_{\boldsymbol{a}} = \mathbf{arg\,min}_{\boldsymbol{\theta}} \sum_{\boldsymbol{m} \in \boldsymbol{D_a}} \left(x_{t,a}^{\mathrm{T}} \cdot \theta_a - \boldsymbol{b_a^{(m)}} \right)^{\mathbf{2}}$
      - Which is solved by: $\hat{\theta}_a = \left(D_a^{\mathrm{T}} D_a + I_d\right)^{-1} D_a^T b_a$

$\mathbf{I_d}$ is **d** x **d** identity matrix

LinUCB Algorithm (3)

- One can then show (using similar techniques as we used for UCB) that

$$\left|\mathbf{x}_{t,a}^{\top}\hat{\boldsymbol{\theta}}_a - \mathbf{E}[r_{t,a}|\mathbf{x}_{t,a}]\right| \leq \alpha\sqrt{\mathbf{x}_{t,a}^{\top}(\mathbf{D}_a^{\top}\mathbf{D}_a + \mathbf{I}_d)^{-1}\mathbf{x}_{t,a}}$$

$$\alpha = 1 + \sqrt{\ln(2/\delta)/2}$$

- So LinUCB arm selection rule is:

$$a_t \stackrel{\text{def}}{=} \arg\max_{a \in \mathcal{A}_t} \left(\mathbf{x}_{t,a}^{\top}\hat{\boldsymbol{\theta}}_a + \alpha\sqrt{\mathbf{x}_{t,a}^{\top}\mathbf{A}_a^{-1}\mathbf{x}_{t,a}} \right)$$

Estimated $\boldsymbol{\mu_a}$

Confidence interval:
Standard deviation

$$\mathbf{A}_a \stackrel{\text{def}}{=} \mathbf{D}_a^{\top}\mathbf{D}_a + \mathbf{I}_d$$

LinUCB Algorithm (3)

$$\mathbf{A}_a \stackrel{\text{def}}{=} \mathbf{D}_a^\top \mathbf{D}_a + \mathbf{I}_d$$

Initialization:

For each arm $\boldsymbol{a}$:

$\boldsymbol{A_a = I_d}$ identity matrix m x m

$\boldsymbol{b_a = [0]_d}$ vector of zeros

Online algorithm:

For t = 1, 2, 3,… T:

Observe features of all arms $\boldsymbol{a}$: $\boldsymbol{x_{t,a} \in R^d}$

For each arm $\boldsymbol{a}$**:**

$\boldsymbol{\theta_a = A_a^{-1}\, b_a}$ regression coefficients

$\boldsymbol{p_{t,a} = \theta_a^T x_{t,a} + \alpha\sqrt{x_{t,a}^T\, A_a^{-1}\, x_{t,a}}}$ confidence bound

Choose arm $\boldsymbol{a_t = \arg\max_a p_{t,a}}$ choose arm

$\boldsymbol{A_{a_t} = A_{a_t} + x_{t,a_t}\, x_{t,a_t}^T}$ update A for the chosen arm a_t

$\boldsymbol{b_{a_t} = b_{a_t} + r_t\, x_{t,a_t}}$ updated b for the chosen arm a_t

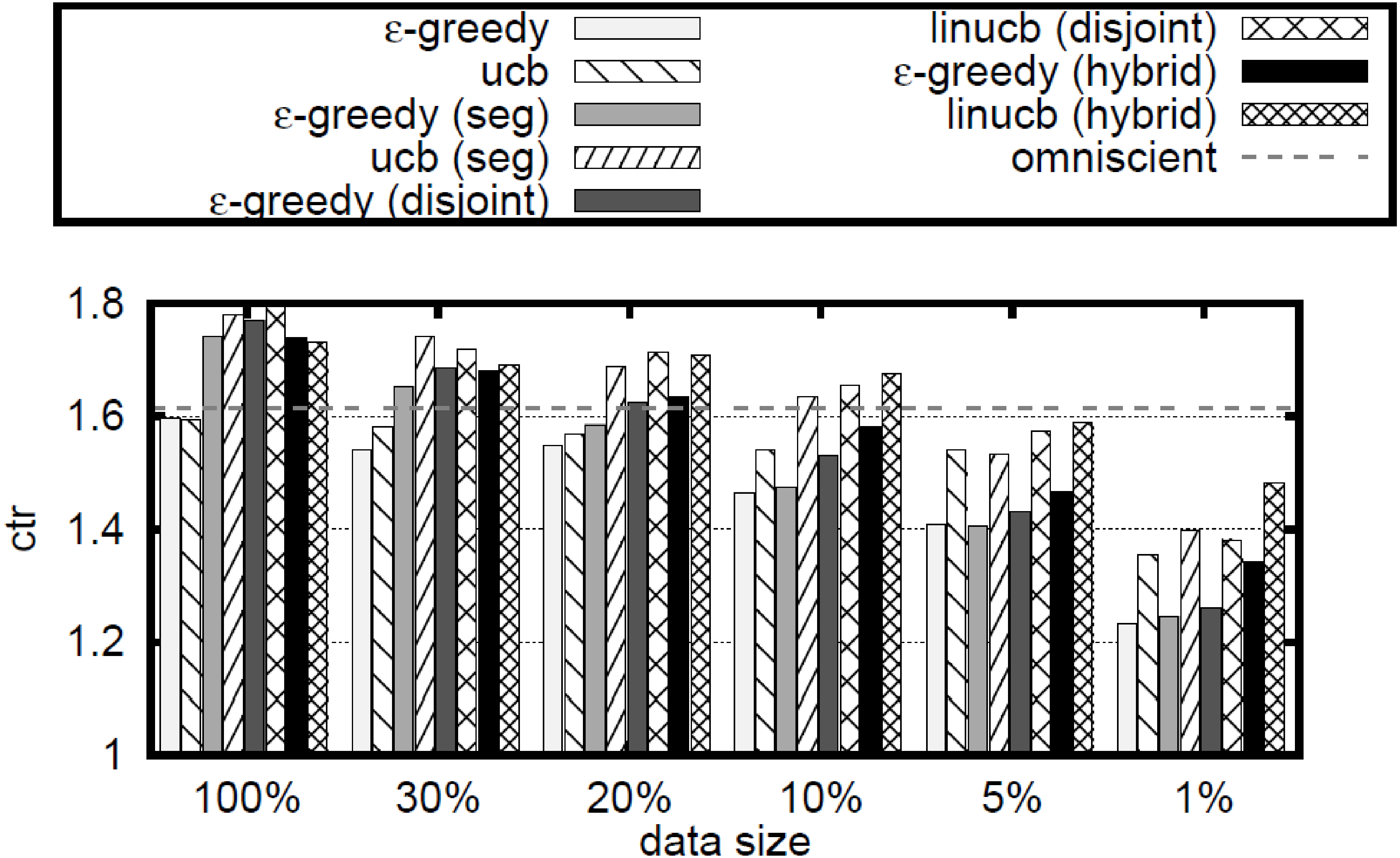
LinUCB: Discussion

- **LinUCB** computational complexity is
  - **Linear** in the **number of arms** and
  - At most **cubic** in the **number of features**
- **LinUCB** works well for a **dynamic arm set** (arms come and go):
  - For example, in news article recommendation, for instance, editors add/remove articles to/from a pool

Yahoo! News Experiment

- **What to put in slots F1, F2, F3, F4 to make the user click?**

Results



ε-greedy
ucb
ε-greedy (seg)
ucb (seg)
ε-greedy (disjoint)
linucb (disjoint)
ε-greedy (hybrid)
linucb (hybrid)
omniscient
1.8
1.6
1.4
1.2
1
ctr
100%
30%
20%
10%
5%
1%
data size

Relevance vs. Diversity

- Want to choose a set that caters to as many users as possible
- Users may have different interests, queries may be ambiguous
- Want to optimize both the relevance and diversity