18-661 Introduction to Machine Learning

Online Learning and Multi-Armed Bandits

Spring 2025

ECE – Carnegie Mellon University

Announcements

- April 14 (today): Online learning and multi-arm bandits.
- April 16 (Wednesday): Mini-exam 3 and guest lectures.
  - Neural combinatorial bandits (Baran Atalar).
  - Large language model agents (Hanqing Yang).
- Next week: Reinforcement learning, last week of classes.
- Homework 5: Released on Wednesday, due April 29 (Tuesday after the last week of classes).

Outline

1. Review: PCA and ICA
2. Online Learning
3. Multi-Armed Bandits

Review: PCA and ICA

Dimensionality Reduction

Issues

1. Measure redundant signals
2. Represent data via the method by which it was gathered

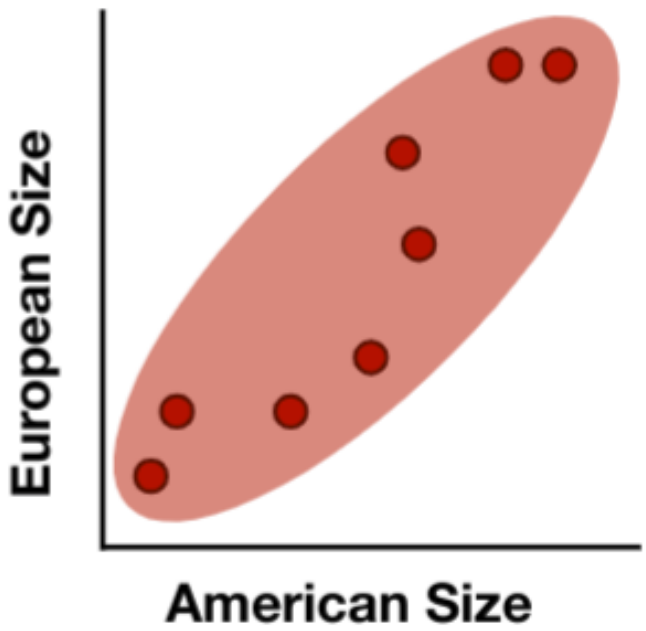
Goal:
Find a "better" representation for data

1. To visualize and discover hidden patterns
2. Preprocessing for supervised task

How do we define "better"?

Goal: Maximize Variance

- For PCA, to identify the right direction to project to, we study the variation of the data in different directions.
- Is the direction on the right a good direction? Does it capture the variation in the data?
- PCA solution finds directions of maximal variance.



PCA Formulation

- $\mathbf{X}$ is $n \times d$ (raw data, each row is a data point)
- $\mathbf{P}$ is $d \times k$ (columns are k principal components)
- $\mathbf{Z} = \mathbf{XP}$ is $n \times k$ (reduced representation)

$$\begin{bmatrix} & \mathbf{Z} & \end{bmatrix} = \begin{bmatrix} & \mathbf{X} & \end{bmatrix} \begin{bmatrix} & \mathbf{P} & \end{bmatrix}$$

  - Intuitively, each column of P is a direction we project to.
  - Each row of Z is the coordinates of the reduced representation under the frame defined by the column vectors in P.
- How do we design the matrix $\mathbf{P}$? The k directions should be the "high variance" directions!

Zero-Center All the Features

For the convenience of calculating variance, it is useful to zero-center all the features.

Given n training points with d features:

- $\mathbf{X} \in \mathbb{R}^{n \times d}$: matrix storing points
- $\mathbf{x}_j^{(i)}$: j^{th} feature value for i^{th} point
- μ_j: mean of j^{th} feature
- Deduct μ_j from all features

$$\begin{bmatrix} \\ \mathbf{Z} \\ \\ \end{bmatrix} = \underbrace{\begin{bmatrix} | & | & | \\ \mathbf{x}_1 & \dots & \mathbf{x}_d \\ | & | & | \end{bmatrix}}_{\mathbf{X}} \begin{bmatrix} \\ \mathbf{P} \\ \\ \end{bmatrix}$$

PCA Solution

- Use eigenvalue decomposition on $\mathbf{C_X}$ to write the covariance matrix of $\mathbf{Z}$ as

$$\mathbf{C_Z} = \mathbf{P}^T \mathbf{C_X} \mathbf{P} = \mathbf{P}^T \mathbf{Q} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_d \end{bmatrix} \mathbf{Q}^T \mathbf{P}$$

- How to find the k directions s.t. the variance in $\mathbf{Z}$ is the largest? Since $\mathbf{C_Z}$ is a matrix, the "largest" is measured using the matrix trace, i.e. sum of the diagonal entries of $\mathbf{C_Z}$.
- Solution: Choose $\mathbf{P}$ as the first k columns of $\mathbf{Q}$,

$$\mathbf{P} = \begin{bmatrix} | & | & | \\ \mathbf{v}_1 & \dots & \mathbf{v}_k \\ | & | & | \end{bmatrix}$$

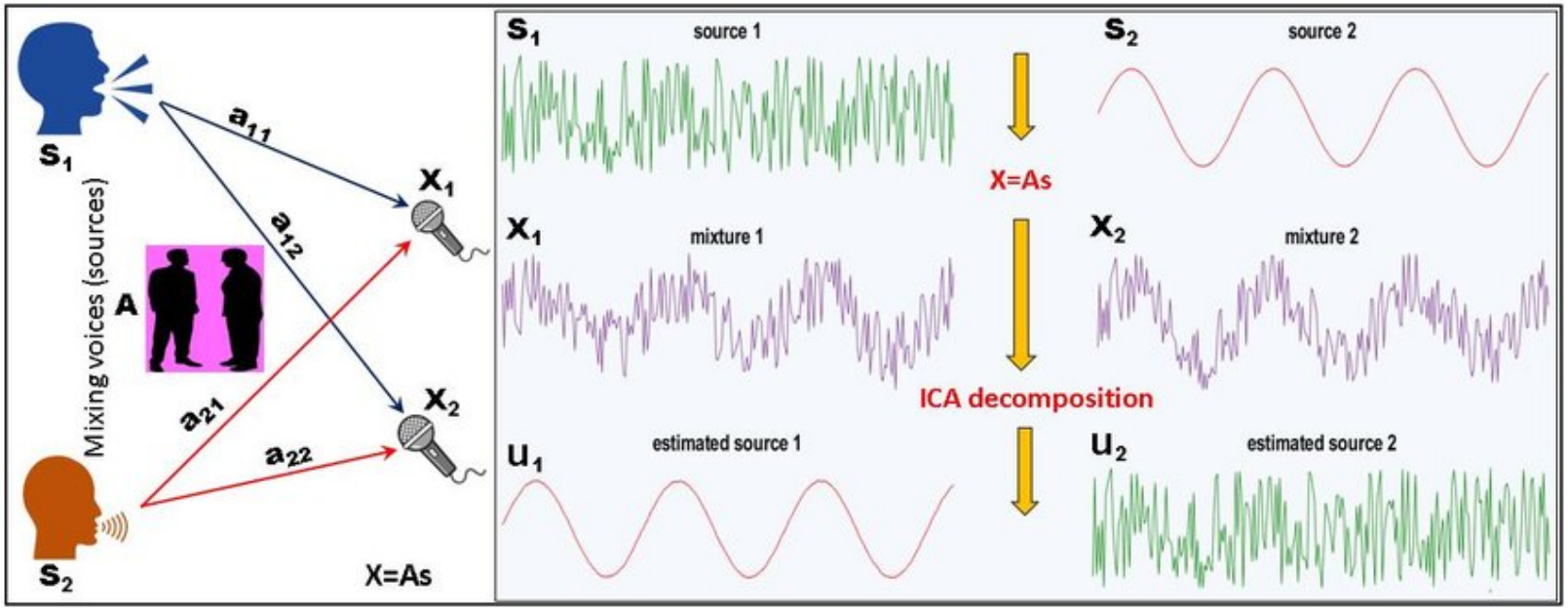
  i.e. the eigenvectors of the top k eigenvalues.
- This captures the k directions of maximum variance. Also, these k directions are orthogonal to each other.

Other Practical Tips

PCA assumptions (linearity, orthogonality) not always appropriate

- Various extensions to PCA with different underlying assumptions, e.g., manifold learning, Kernel PCA, ICA
- Centering is crucial, i.e., we must preprocess data so that all features have zero mean before applying PCA
- PCA results dependent on scaling of data
- Data is sometimes rescaled in practice before applying PCA

The Cocktail Party Problem



- PCA tries to find an orthogonal representation of the mixed data.
- ICA tries to disentangle the data sources.

Comparing PCA and ICA

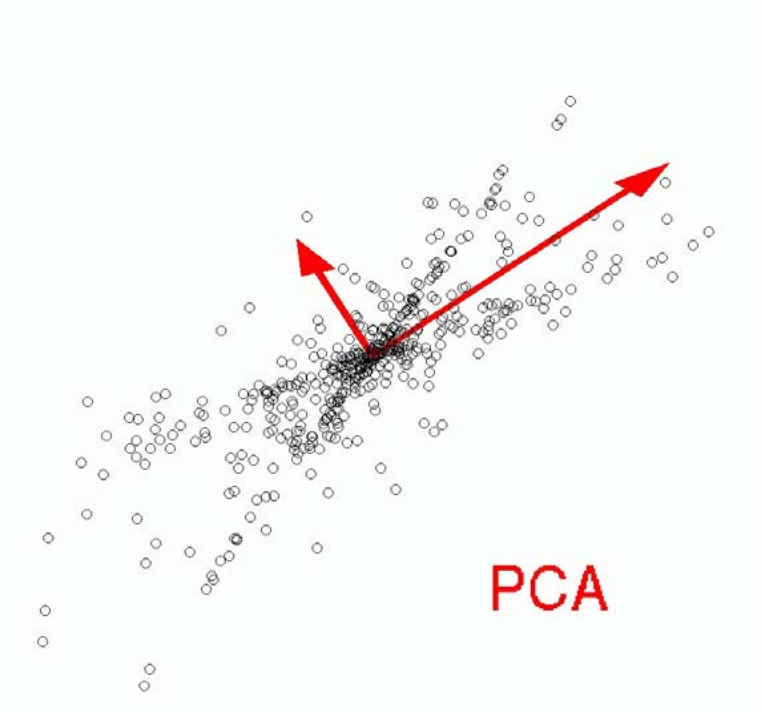
Let $\mathbf{S}$ denote our original data. It can be transformed to a new set of features $\mathbf{X}$:

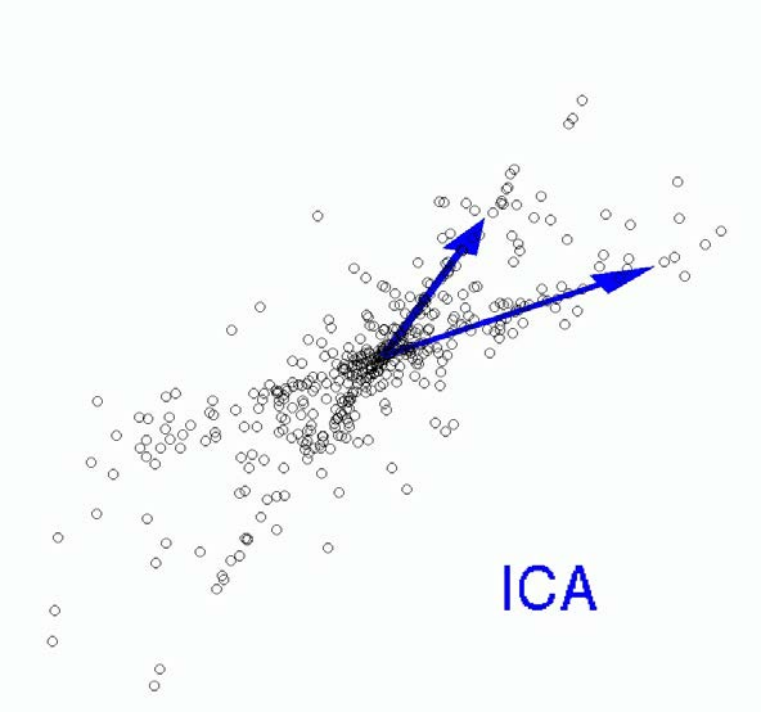
$$\text{PCA} : \mathbf{X} \approx \mathbf{US},\ \mathbf{U}^T\mathbf{U} = \mathbf{I}$$

$$\text{ICA} : \mathbf{X} \approx \mathbf{AS},\ \mathbf{A} \text{ invertible}$$

- PCA reduces the number of features (compression). ICA does not.
- PCA removes correlations but not higher-order dependencies. ICA removes these dependencies.
- PCA allows you to rank the importance of the new features. ICA does not.

ICA vs. PCA





Online Learning

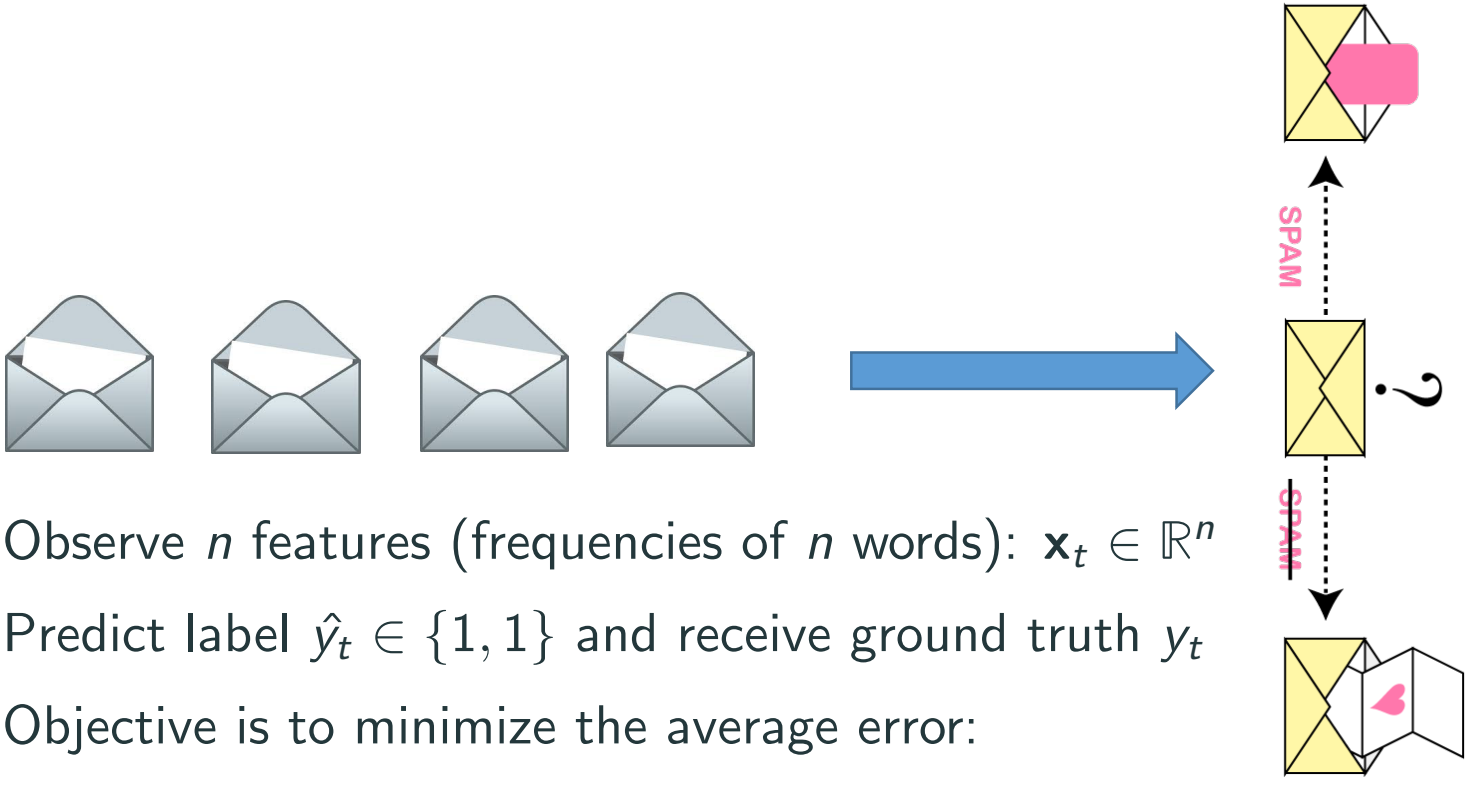
What Is Online Learning?

Online learning occurs when we do not have access to our entire training dataset when we start training. We consider a sequence of data and update the predictor based on the latest sample(s) in the sequence.

- Stochastic gradient descent (and variants)
- etc...

Example: Sequential Spam Classification

Build a "bag of words" classifier for whether an email is spam or ham.



- Observe n features (frequencies of n words): $\mathbf{x}_t \in \mathbb{R}^n$
- Predict label $\hat{y}_t \in \{1, 1\}$ and receive ground truth y_t
- Objective is to minimize the average error:

$$\min_{\mathbf{w} \in \mathbb{R}^n} -\sum_t y_t \log\left(\sigma\left(\mathbf{w}^T \mathbf{x}_t\right)\right) + (1 - y_t) \log\left[1 - \sigma\left(\mathbf{w}^T \mathbf{x}_t\right)\right]$$

Why Is This Useful?

Online learning helps us handle large datasets.

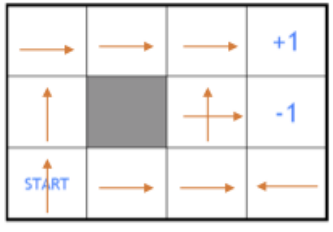
- Allows us to only consider part of the data at a time.
- Can start learning and taking actions before we have all of the data.
- Less data storage and processing requirements.

It also automatically adapts models to changes in the underlying process over time (e.g., house prices' relationship to square footage).

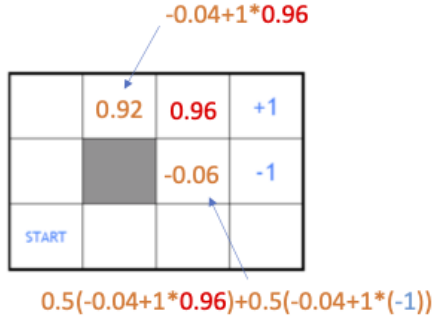
- Prioritizes more recent data samples (sometimes).
- Eventually converges to a good model.

Task 5: Reinforcement Learning

How to take the actions that maximize reward?



input:
state-action-reward trajectories



find: Value function or Q function

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

return: Optimal Policy π

Course Covers: Online learning and bandits, Bellman equation, Policy Evaluation, Q-learning

Online Search

Which advertisement will maximize our revenue?

Note our objective has changed! We no longer (just) want to minimize the loss...instead, we are using a model (of user reactions to ads) to optimize our real objective.

A Simple Approach...

A/B Testing

- Give each alternative to a random sample of users for some time.
- Pick the option that performs the best at the end.

Learn a Better Approach

We want to learn which ads appeal to users: maximize users' click rate on the ads shown to them.

- Offer some ads to users.
- Observe which ads users click on.
- Update estimates of ads' appeal to users.
- Offer more ads according to the updated estimates.

Inputs: user characteristics; Prediction: ads' appeal to users; Feedback: click (or not) on the ad shown

Feedback in Online Learning Problems

What if we don't get full feedback on our actions?

- Spam classification: we know whether our prediction was correct. *We also know whether another choice of model parameters would have given the correct answer.*
- Online advertising: we only know the appeal of the ad shown. We have *no idea* if we were right about the appeal of other ads.
- Partial information often occurs because we observe feedback from an action taken on the prediction, not the prediction itself.
  - Analogy to linear regression: instead of learning the ground truth y, we only observe the value of the loss function, $l(y)$.
  - Makes it very hard to optimize the parameters!
- Often considered via multi-armed bandit problems.

Evaluating Online Learning Models

How do we evaluate online learning models?

- Previously, we measured the loss of our model on test data–but the model changes over time! For model parameters β: $\beta_1 \to \beta_2 \to \ldots \beta_t \to \ldots$
- Define regret as the cumulative difference in loss compared to the best model in hindsight. For a sequence of data (x_t, y_t):

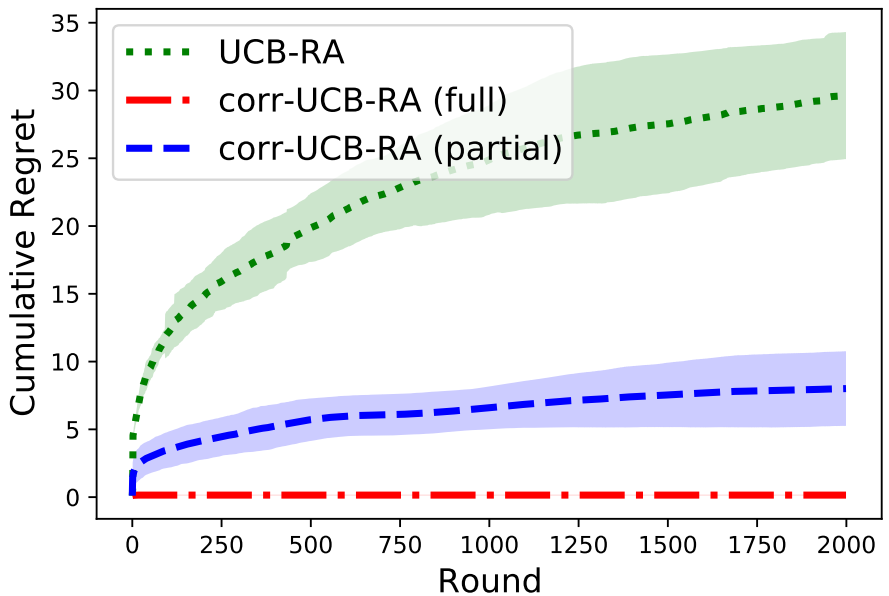
$$R(T) = \sum_{t=1}^{T} [l(x_t, y_t, \beta_t) - l(x_t, y_t, \beta^*)]$$

  Evaluate the loss of each sample using the model parameters at that time, compared to the best parameters in hindsight.
  - Usually, we want the regret to decay sub-linearly over time.
  - Implies that the regret per iteration decays to zero: eventually, you will recover the optimal-in-hindsight model.

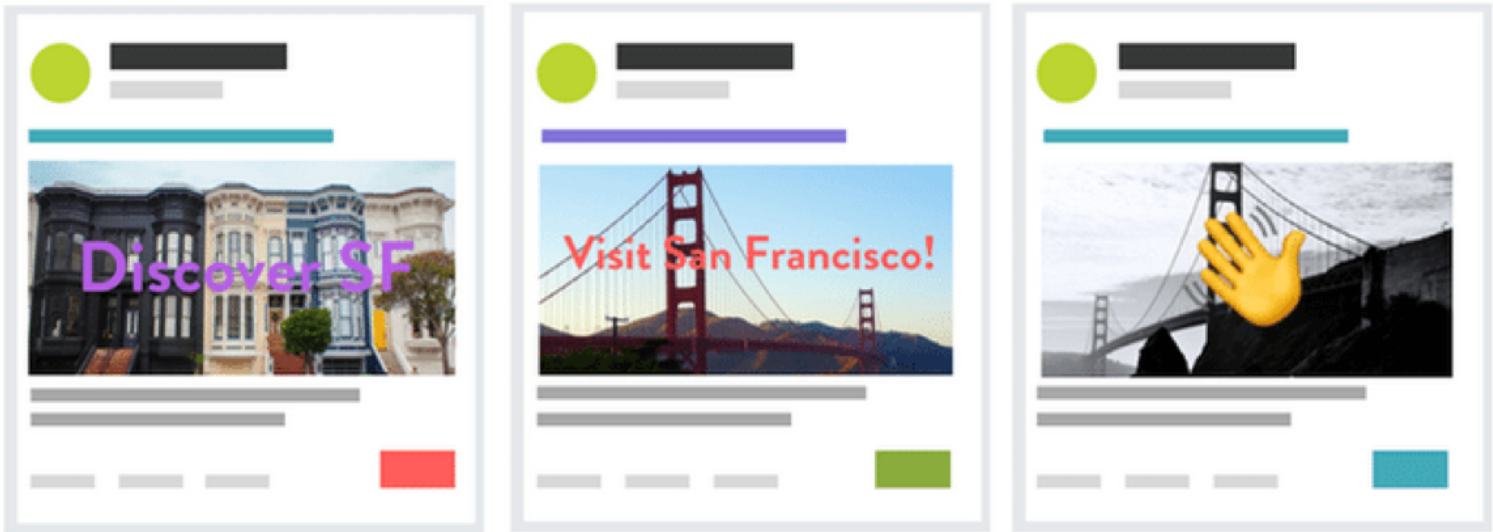
Regret over Time

$$R(T) = \sum_{t=1}^{T} [l(x_t, y_t, \beta_t) - l(x_t, y_t, \beta^*)] = O(\log T)$$

The difference in loss $l(x_t, y_t, \beta_t) - l(x_t, y_t, \beta_t^*) \sim 0$ (i.e., $\beta_t = \beta^*$) for large enough t.



Regret for Online Ads



Suppose that the best ad is the 2nd one, and it gives us a click-through probability (reward) of $r_2 = 0.3$. Then our regret is

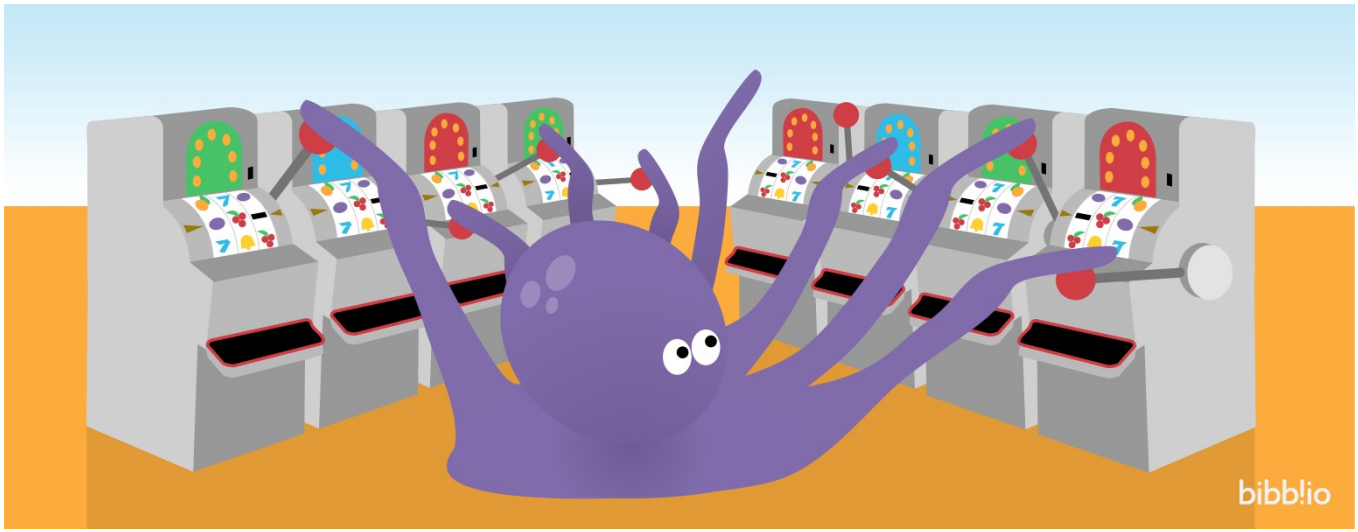
$$R(T) = (r_1 - 0.3) + (r_2 - 0.3) + (r_1 - 0.3) + (r_2 - 0.3) + (r_3 - 0.3) \ldots$$

if we choose arms 1, 2, 1, 2, 3, ...

Multi-Armed Bandits

Multi-armed Bandits

Which slot machine will give me the most money?



Using Online Learning...

Can we learn which slot machine gives the most money?

$1
$0
$0

$1
$4
$0
$2
$1
$3
$5

$1
$0
$1
$2

Bandit Formulation

We can play multiple rounds $t = 1, 2, \ldots, T$. In each round, we select an arm i_t from a fixed set $i = 1, 2, \ldots, n$; and observe the reward $r(i_t)$ that the arm gives.

Arm 1

Arm 2

Arm 3

Objective: Maximize the total reward over time, or equivalently minimize the regret compared to the best arm in hindsight.

Bandit Formulation

Arm 1

Arm 2

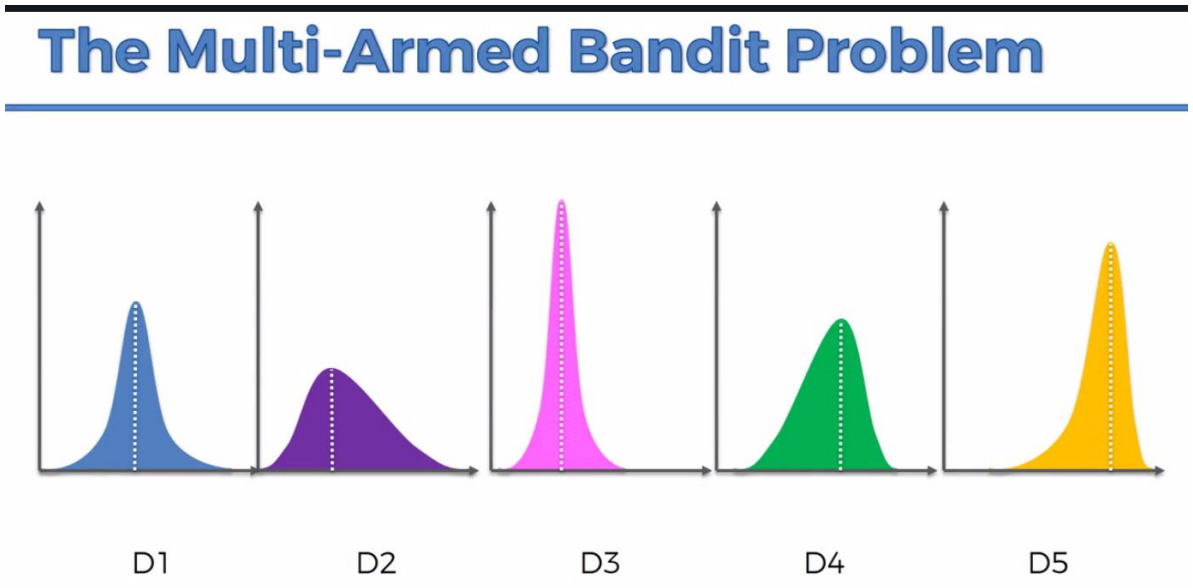
Arm 3

- The reward at each arm is stochastic (e.g., 0 with probability p_i and otherwise 1). We do not know its distribution.
- Usually, the rewards are i.i.d. (independent and identically distributed) over time. The best arm is then the arm with highest *expected* reward.
- We only observe the reward of the arm we played.

Online ads example: arm = ad, reward = 1 if the user clicks on the ad

Finding the Best Arm

Which arm should I choose, if each arm's reward is drawn from the distribution below?



We choose D5, as it has the highest expected reward. In practice, the distributions must be estimated by observing arm rewards.

Learning the Best Arm

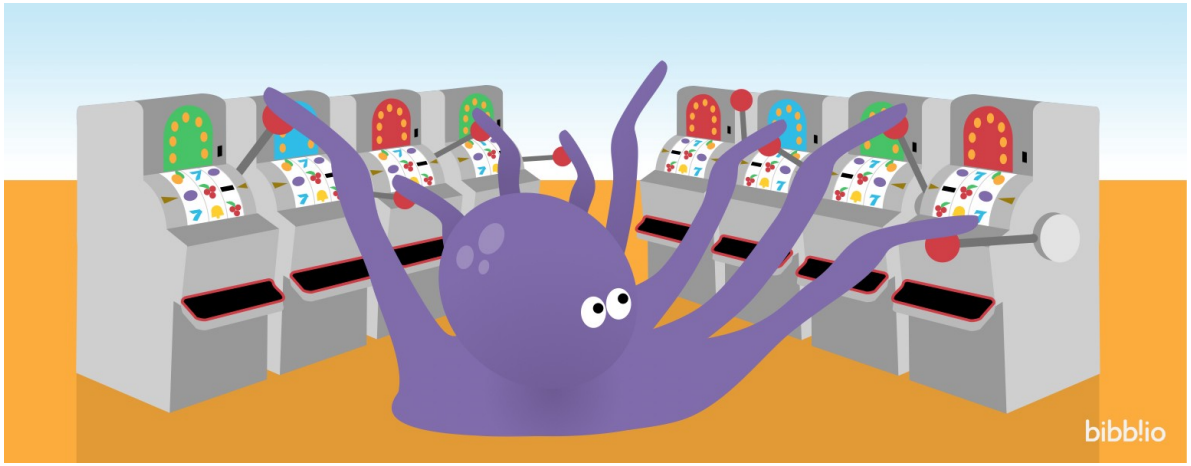
Which arm do I pick next, so that I maximize my reward over time?

$1
$0
$0

$1
$4
$0
$2
$1
$3
$5

$1
$0
$1
$2
$12
$11

Exploration vs. Exploitation Tradeoff



Which arm should I play?

- Best arm observed so far? (exploitation)
- Or should I look around to try and find a better arm? (exploration)

We need both in order to maximize the total reward.

Regret

Suppose we choose the arm i_t based on past observations, and that the reward $r(i_t)$ is independent from past rewards.

We define the expected regret over T rounds as

$$R_T = T\rho^* - \mathbb{E}\left(\sum_{t=1}^{T} r(i_t)\right),$$

where ρ^* is the highest expected reward over all arms.

- The first term captures the highest cumulative reward in hindsight.
- The second term captures the accumulated reward using our strategy.

Implications of the Regret

- The minimum possible regret is $O(\log T)$ [Lai & Robbins, 1985]. So the regret will be infinite as $T \to \infty$.
- But, the minimum average regret is

$$\frac{R_T}{T} = O\left(\frac{\log T}{T}\right) \to 0$$

  as $T \to \infty$. This means (roughly speaking) that we can find the arm with highest expected reward, given enough tries.
- Most MAB algorithms aim to achieve sublinear regret, so that the average regret goes to 0 as $T \to \infty$.

Can we find an algorithm that achieves the regret $O(\log T)$?

The ϵ-Greedy Algorithm

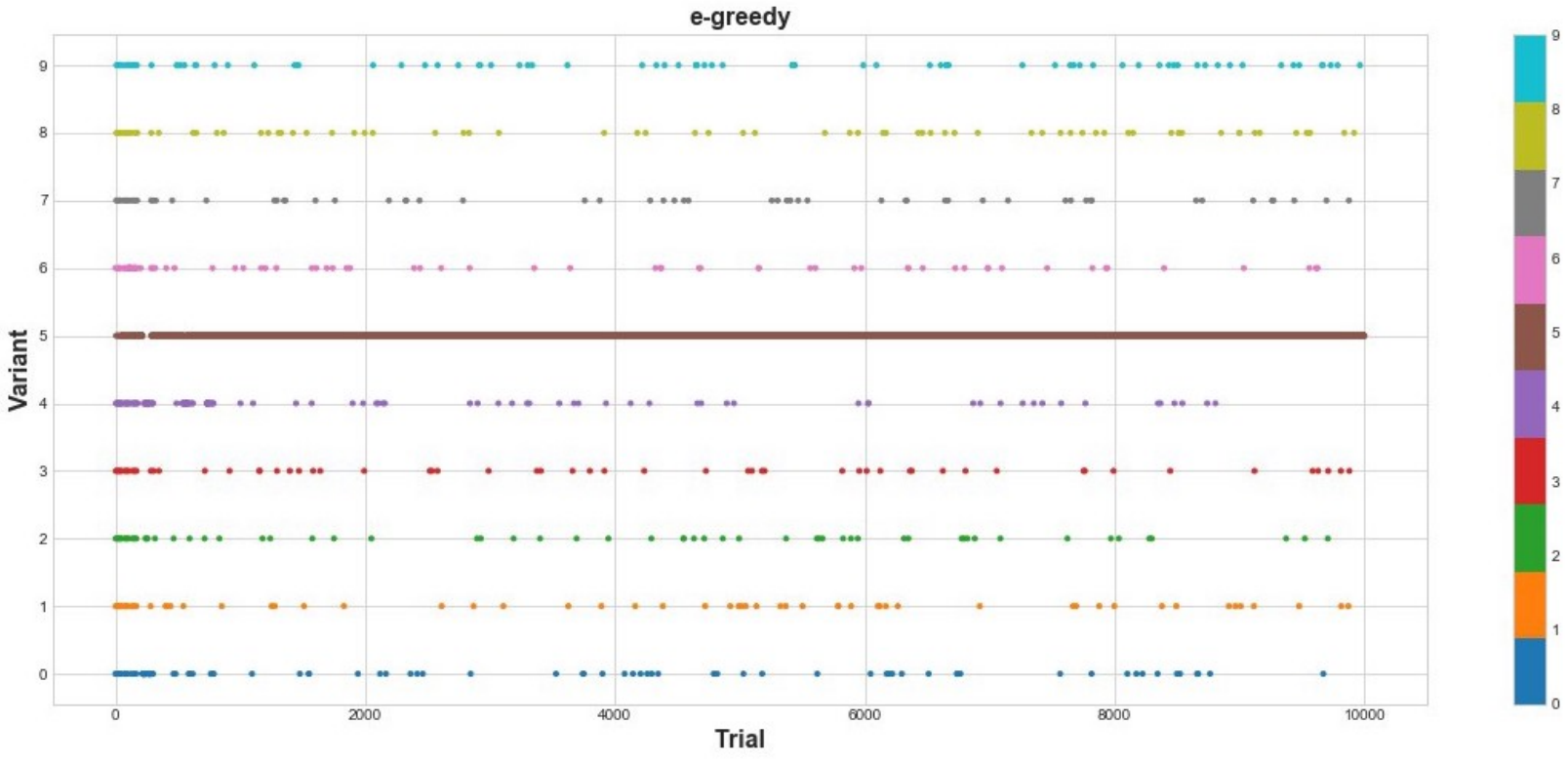
Very simple solution that is easy to implement. The idea is to exploit the best arm, but explore a random arm ϵ fraction of the time.

1. Try each arm and observe the reward.
2. Calculate the empirical average reward for each arm i:

$$\overline{r(i)} = \frac{\text{total reward from pulling this arm in the past}}{\text{number of times I pulled this arm}} = \frac{\sum_{t:j(t)=i} r_t}{T_i},$$

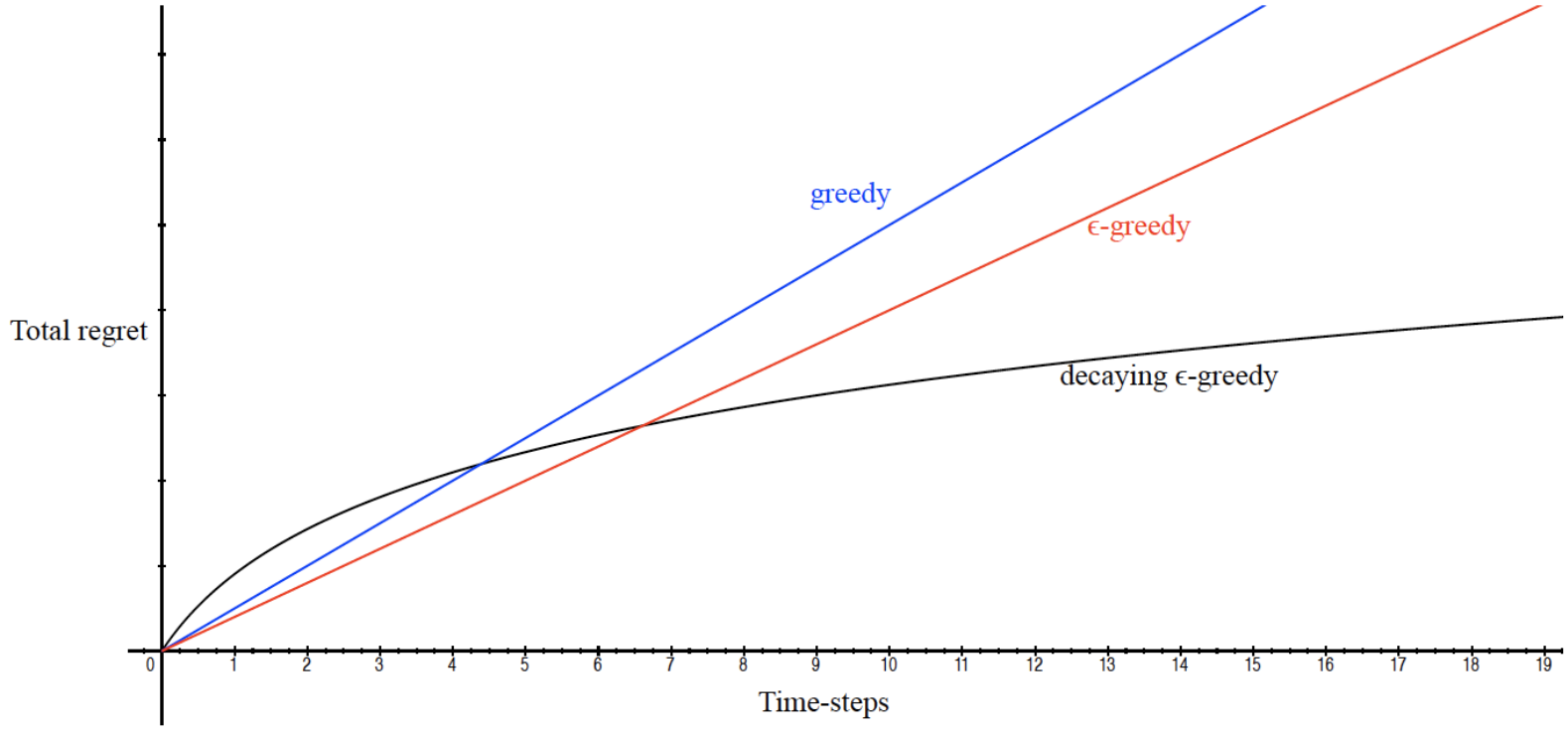
   where $j(t) = i$ indicates that arm i was played at time t, r_t is the reward, and T_i is the number of times arm i has been played.
3. With probability $1 - \epsilon$, play the arm with highest $\overline{r(i)}$. Otherwise, choose an arm at random.
4. Observe the reward and re-calculate $\overline{r(i)}$ for that arm.
5. Repeat steps 2-4 for T rounds.

Understanding ϵ-Greedy



- In the first thousand iterations, all arms are chosen fairly frequently.
- Eventually the algorithm realizes that arm 5 has the highest expected reward.

Regrets of Greedy Policies



[1]

- Greedy policy incurs linear regret since it can lock on a sub-optimal policy.
- ϵ-Greedy always explores by ϵ fraction and therefore its regret is still linear.
- A possible fix is using decaying ϵ, however it is hard to design the schedule.
- A better algorithm called UCB1 can fix the issue.

[1] Image courtesy: David Silver's lecture.

The UCB1 Algorithm

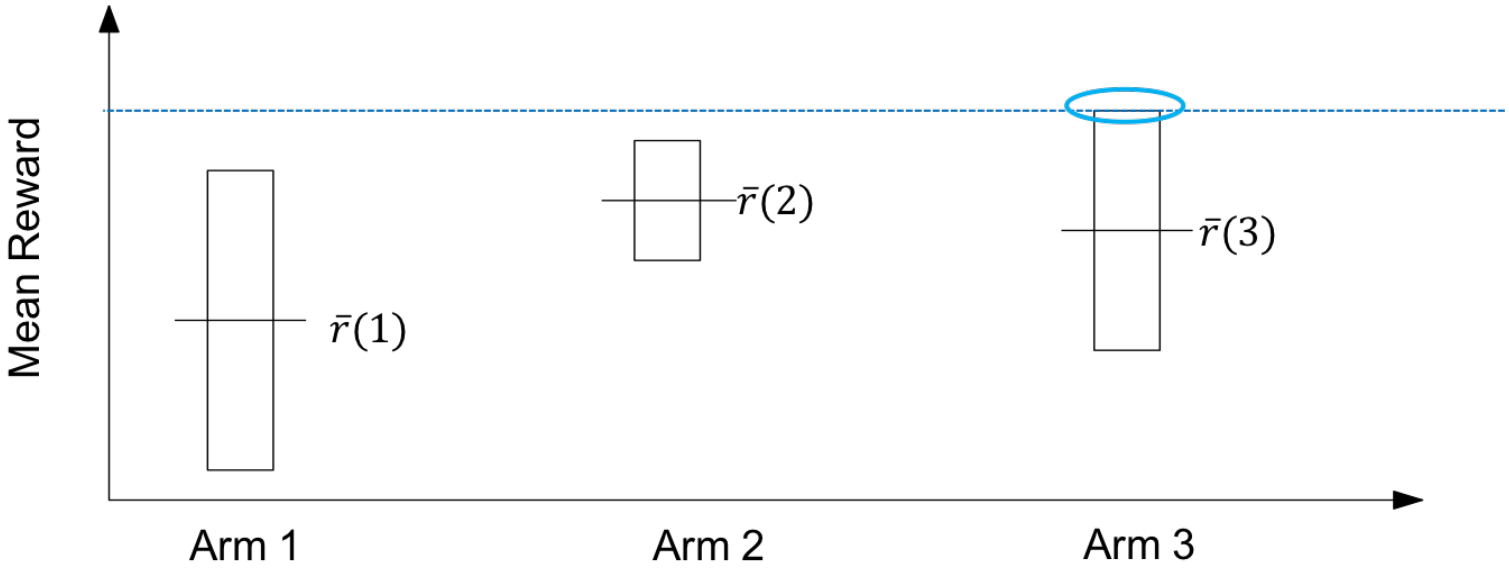
Very simple policy from Auer, Cesa-Bianchi, and Fisher (2002) with $O(\log T)$ regret. The idea is to always try the best arm, where "best" includes exploration and exploitation.

1. Try each arm and observe the reward.
2. Calculate the UCB (upper confidence bound) index for each arm i:

$$UCB_i = \overline{r(i)} + \sqrt{\frac{2 \log T}{T_i}},$$

   where $\overline{r(i)}$ is the empirical average reward for arm i and T_i is the number of times arm i has been played.
3. Play the arm with the highest UCB index.
4. Observe the reward and re-calculate the UCB index for each arm.
5. Repeat steps 2-4 for T rounds.

Illustrating UCB1



$$UCB_i = \overline{r(i)} + \sqrt{\frac{2 \log T}{T_i}},$$

Adds a "confidence interval" to the mean reward.

Understanding UCB1

$$UCB_i = \overline{r(i)} + \sqrt{\frac{2 \log T}{T_i}},$$

- Exploitation: $\overline{r(i)}$ is the average observed reward. A high UCB value corresponds to a high observed reward.
- Exploration: $\sqrt{\frac{2 \log T}{T_i}}$ decreases as we make more observations (T_i grows). A high UCB value corresponds to making few observations of that arm.

UCB1 leads to $O\left(\sqrt{KT \log T}\right)$ *worst-case* regret! If we allow "instance-dependent" constants, this becomes $O(\log T)$.

More advanced algorithms aim to reduce the constant on this bound.

Optimism in the Face of Uncertainty

UCB finds an optimistic estimate of the arm rewards by adding a confidence interval to the mean observed reward.

- Common technique in sequential decision making, as it encourages exploration.
- Many different ways to compute the confidence interval!
  - UCB-tuned (Auer et al. [2002]): include estimate of the variance in the confidence term
  - UCB-normal (Auer et al. [2002]): specialized to normal distributions
  - KL-UCB (Garivier and Cappé [2011]): use divergence of observed and estimated reward distributions

  Often, one scales the confidence interval in practice to encourage less exploration.

Thompson Sampling

Which arm should you pick next?

- ϵ-greedy: Best arm so far (exploit) with probability $1 - \epsilon$, otherwise random (explore).
- UCB1: Arm with the highest UCB value.

Thompson sampling instead fits a distribution to the reward parameters. For example, if r_i is Bernoulli with probability p_i of being equal to 1:

1. Define a prior $p_i \sim$beta(α_i, β_i) distribution on p_i for each arm i.
2. Sample p_i from beta(α_i, β_i) for each arm i.
3. Play the arm with highest expected reward $p_i = \mathbb{E}[r_i|p_i]$.
4. Observe the reward and update the beta(α_i, β_i) distribution for the chosen arm (use MAP!)
5. Repeat steps 2 to 4.

Thompson Sampling vs. UCB

Regret of Thompson Sampling
Since we model p_i as a random parameter, we have an additional source of randomness. Thus, we evaluate the Bayesian regret

$$\mathbb{E}_{p_i \sim prior}\left[T\rho^* - \mathbb{E}\left(\sum_{t=1}^{T} r(i_t)\right)\right].$$

Thompson sampling achieves a Bayesian regret of $O(\sqrt{KT\log T})$, where K is the number of arms.

- Thompson sampling requires a prior distribution, which may in practice slow convergence if the prior is incorrect.
- Bayesian regret is in general weaker than frequentist regret, due to the additional expectation.
- These are worst-case bounds on expected regret. More refined $O(\log T)$ bounds can be obtained for specific problem instances.

Other Bandit Variations

- Multi-player bandit: Many users are trying to play arms, and if multiple users play the same arm, they receive lower reward. Has important applications to wireless spectrum sharing.
- Cascading bandit: We can observe more than one arm. For instance, we can show multiple ads to a user and assume they click on the first appealing one.
- Combinatorial bandit: We need to choose a combination of arms, not a single arm. Often assumes semi-bandit feedback: we know the reward of each chosen arm, and the collective reward of their combination.
- Adversarial bandit: The rewards are chosen by an adversary who wants to fool us into making a bad decision.
- Contextual bandit: The distribution of rewards depends on some external, known context that may change over time. For instance, ads' appeal to users varies depending on the user demographics.

And many, many more...

Summary

You should know:

- What an "online" machine learning algorithm means, and scenarios where they are useful.
- How to define the regret of an online algorithm, and why we want it to be sublinear over time.
- Multi-armed bandit formulation.
- ϵ-greedy vs. UCB1 vs. Thompson sampling approaches.