18-661 Introduction to Machine Learning

Online Learning

Spring 2020

ECE – Carnegie Mellon University

Announcements

- HW 7 is due Friday, April 24 at 8:59pm PT/11:59pm ET.
- Lecture on Friday (April 17) is cancelled.
- Agenda for the rest of the course:
  - 4/15 (today) and 4/20: Online and reinforcement learning
  - Wednesday, 4/22: Ph.D. student guest lectures (Samarth Gupta, Jianyu Wang, Mike Weber and Yuhang Yao).
  - Monday, 4/27: Last lecture (final review) and practice multiple-choice exam on Gradescope/Zoom.
  - Wednesday, 4/29: Part I (multiple-choice) of the final exam.
  - Friday, 5/1 to Sunday, 5/3: Part II (take-home) of the final exam.

Preparing for the Final Exam

- Part I will be a 110-minute exam during the usual class time on 4/29. Students in conflicting timezones (Africa, India) may start 1 hour early. We will contact you to arrange this. This part of the exam will be closed book, but you may use a cheat sheet as on the midterm.
- Part II will be a take-home exam with no time limit. This part of the exam will be open everything except working with other people.
- The final covers all of the course (including pre-midterm topics). We will give you practice multiple choice and descriptive questions.
- Expect a similar format and difficulty as on the midterm.
- More details to come next week.

Outline

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

Review: PCA and Sparsity

Dimensionality Reduction

Issues

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

Goal:

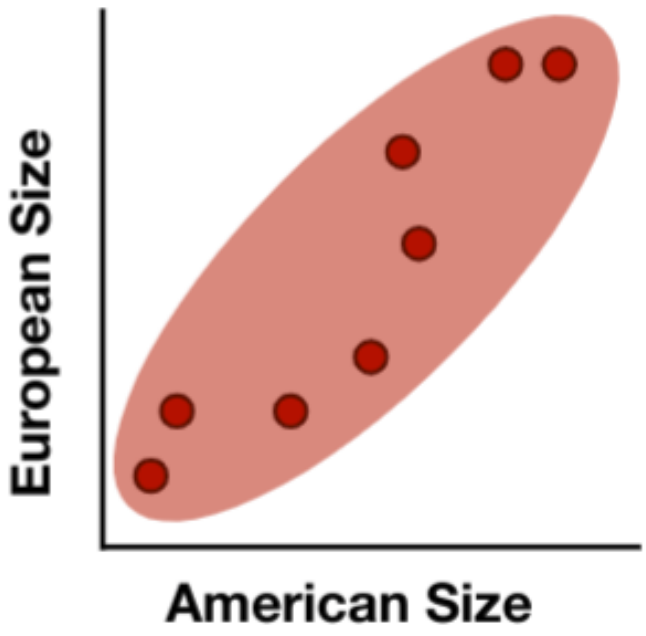
Find a "better" representation for data

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

How do we define "better"?

Goal: Maximize Variance and Minimize Feature-Correlation

- To identify patterns we want to study variation across observations
- Can we do "better", i.e., find a compact representation that captures variation?
- PCA solution finds directions of maximal variance that are orthogonal to each other



PCA Formulation

- $\mathbf{X}$ is $n \times d$ (raw data)
- $\mathbf{P}$ is $d \times k$ (columns are k principal components)
- $\mathbf{Z} = \mathbf{XP}$ is $n \times k$ (reduced representation, PCA "scores")
- Linearity assumption ($\mathbf{Z} = \mathbf{XP}$) simplifies problem

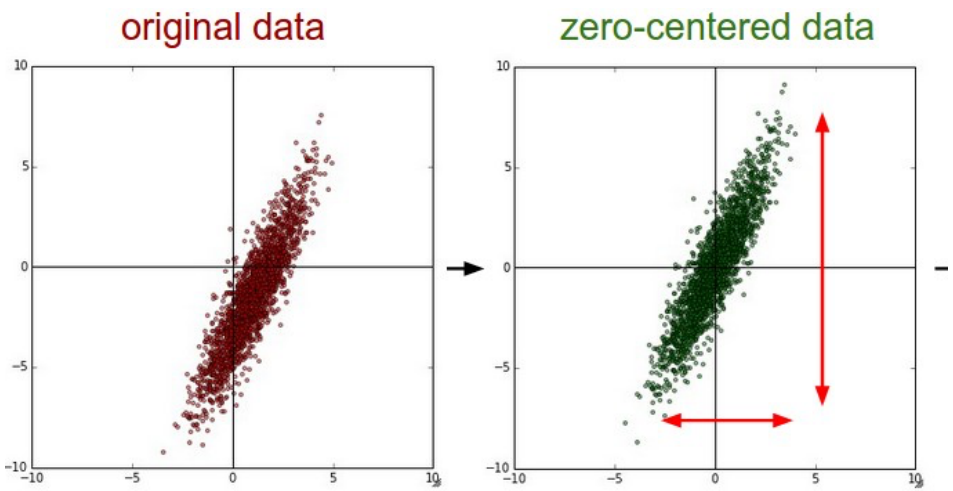
$$\begin{bmatrix} & \mathbf{Z} & \end{bmatrix} = \begin{bmatrix} & \mathbf{X} & \end{bmatrix} \begin{bmatrix} & \mathbf{P} & \end{bmatrix}$$

- Projecting each row of X along the column vectors of $\mathbf{P}$
- How do we design the matrix $\mathbf{P}$?

Step 1: 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 vector for i^{th} point
- μ_j: mean of j^{th} feature
- Deduct μ_j from all features



PCA Solution

- The co-variance matrix $\mathbf{C_X}$ is symmetric, positive semi-definite and it can be decomposed as follows

$$\mathbf{C_X} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$$
$$\mathbf{Q}^T\mathbf{C_X}\mathbf{Q} = \mathbf{\Lambda}$$

- Recall our desired PCA solution

$$\begin{aligned}\mathbf{C_Z} &= \frac{1}{n}\mathbf{Z}^T\mathbf{Z} \\ &= \frac{1}{n}\mathbf{P}^T\mathbf{X}^T\mathbf{X}\mathbf{P} \\ &= \mathbf{P}^T\mathbf{C_X}\mathbf{P}\end{aligned}$$

- $\mathbf{Z} = \mathbf{X}\mathbf{P}$ is $n \times k$ (reduced representation, PCA "scores")
- $\mathbf{C_Z}$ should have zero off-diagonal entries

PCA Solution

$$\mathbf{\Lambda} = \mathbf{Q}^T \mathbf{C_X} \mathbf{Q}$$

$$\mathbf{C_Z} = \mathbf{P}^T \mathbf{C_X} \mathbf{P}$$

- Can we just take $\mathbf{Q} = \mathbf{P}$ and $\mathbf{C_Z} = \mathbf{\Lambda}$? No, because $\mathbf{Z} = \mathbf{XP}$ is $n \times k$ (reduced representation, PCA "scores").
- Choose $\mathbf{P}$ as the first k columns of $\mathbf{Q}$
- Capture the k directions of maximum variance.

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

Iterative PCA Algorithm

Problem

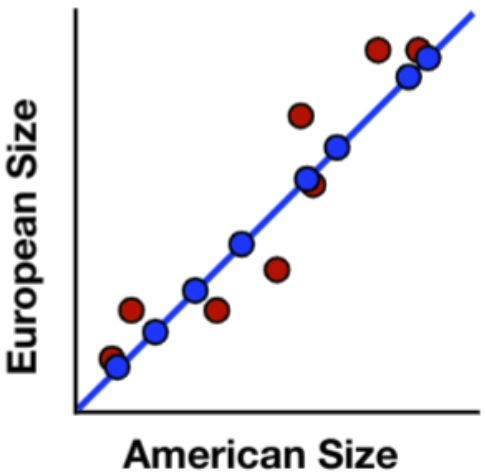
- For high-dimensional original features (large d) computing the eigenvalue decomposition of $\mathbf{C}_X$, and sorting the eigenvalues can be intractable.

Iterative Algorithm

1. Find the top principal component $\mathbf{v}_1$ (need an efficient method for this)
2. Replace each datapoint $\mathbf{x}$ by $\mathbf{x} - \mathbf{v}_1\mathbf{v}_1^T\mathbf{x}$, that is, remove the projection on $\mathbf{v}_1$
3. Recurse until we find k components

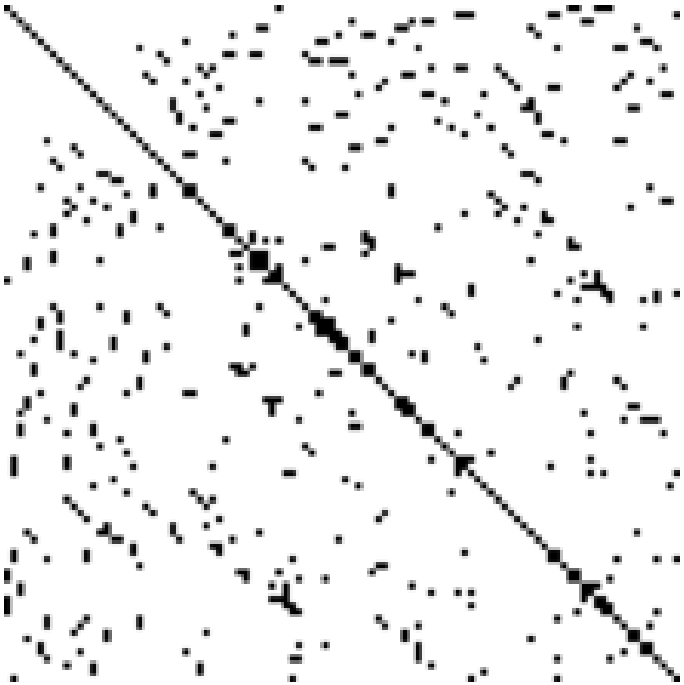
Power Iteration Method to find $\mathbf{v}_1$

1. Find $\mathbf{C_X} = \mathbf{X}^T\mathbf{X}$
2. Initialize $\mathbf{v}_1$ to a random normalized vector
3. For many iterations (or until $\mathbf{v}_1$ becomes stable):
   - $\mathbf{v}_1 \leftarrow \mathbf{C_X}\mathbf{v}_1$
   - Normalize $\mathbf{v}_1$ so that $||\mathbf{v}_1||_2^2 = 1$
4. Return $\mathbf{v}_1$



What Is Sparsity?

A sparse matrix is one in which most of the entries are zero.



$$\text{Sparsity score} = \frac{\#\text{ of zero entries}}{\text{total }\#\text{ of entries}}$$

- Set a threshold for "sparse."
- Black elements at left are nonzero.

Working with Sparse Data

Traditional algorithms are very inefficient on sparse data–just pushing around a lot of zeros.

More efficient data storage:

- Only store indices and nonzero entries, instead of an entire two-dimensional array.
- Dictionary of keys, list of lists, ...

Easier matrix computations:

- Fast matrix multiplication, since we can ignore zero entries.
- Useful for gradient descent, computing distances, ...

Sparse PCA

PCA's features are often linear combinations of *all* the input features. Sparse PCA aims to find sparse linear combinations that also maximize the variance (just like in normal PCA).

If we take $k = 1$ (find the dominant feature):

$$\max_{\mathbf{p}} \mathbf{p}^T \mathbf{C_X} \mathbf{p}$$
$$\text{s.t. } \|\mathbf{p}\|_2 = 1,\ \|\mathbf{p}\|_0 \leq p.$$

Here p, $1 \leq p \leq d$ is a parameter setting the maximum sparsity and $\|\cdot\|_0$ counts the number of nonzero entries in a vector (this is the 0-norm).

This problem is non-convex and difficult to solve...

Online Learning

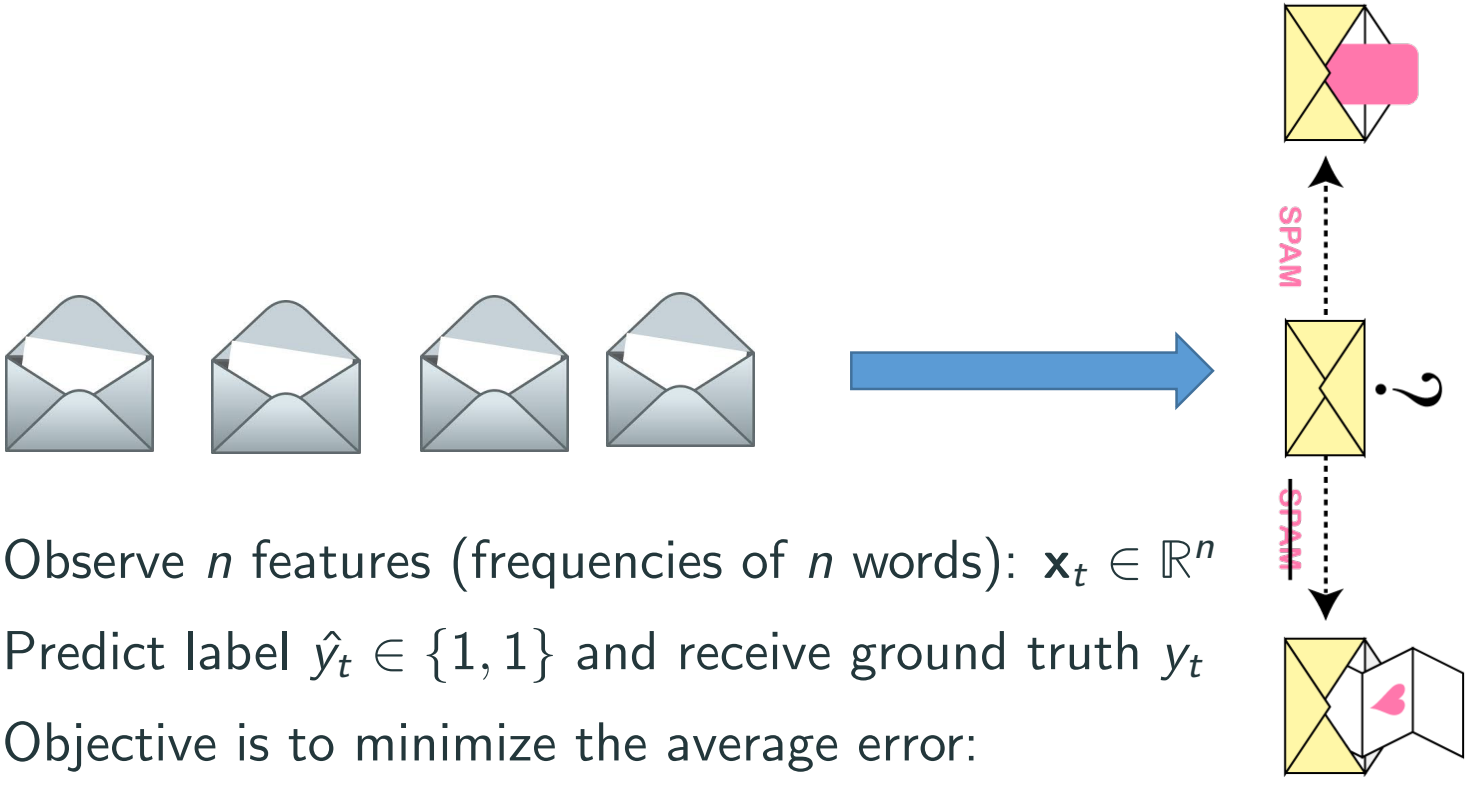
What Is Online Learning?

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
- Perceptron training algorithm
- 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.

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.
- 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):

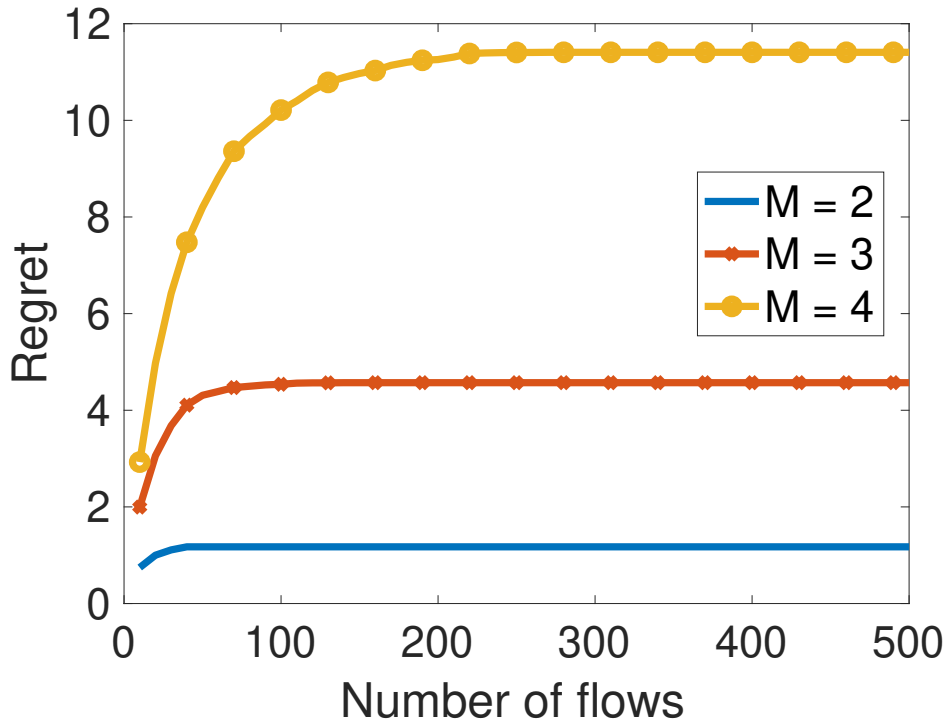
$$R(T) = \sum_{t=1}^{T} [l(x_t, y_t, \beta_t) - l(x_t, y_t, \beta^*)]$$

  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.

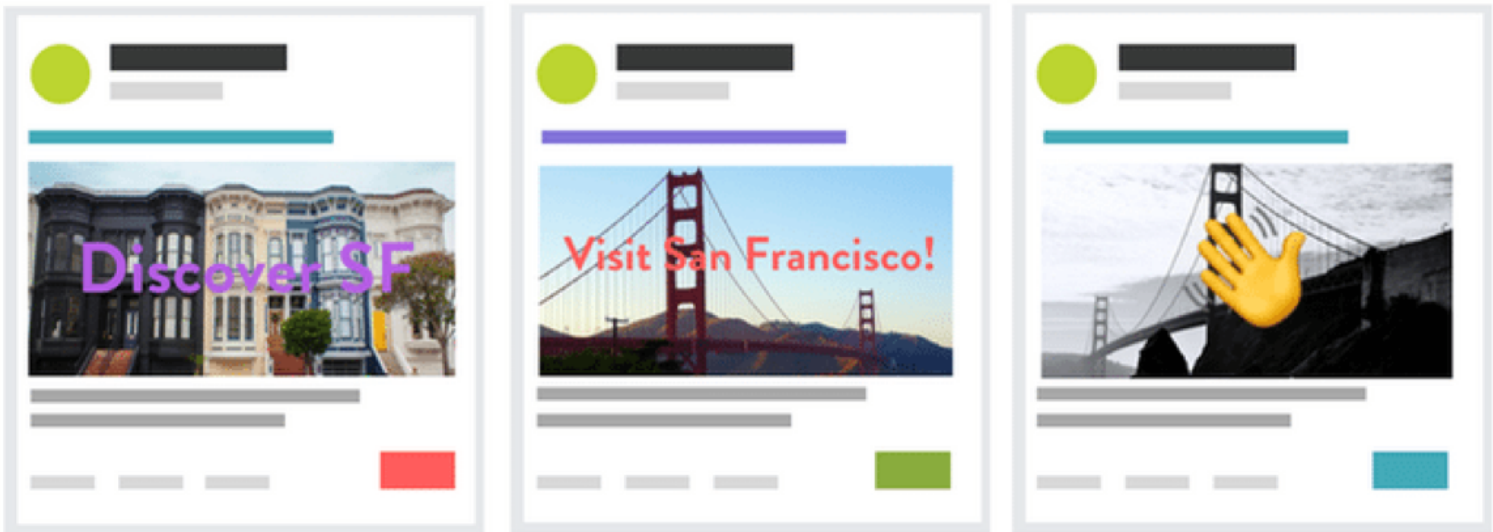
Regret over Time

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

Regret eventually converges to a constant! 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

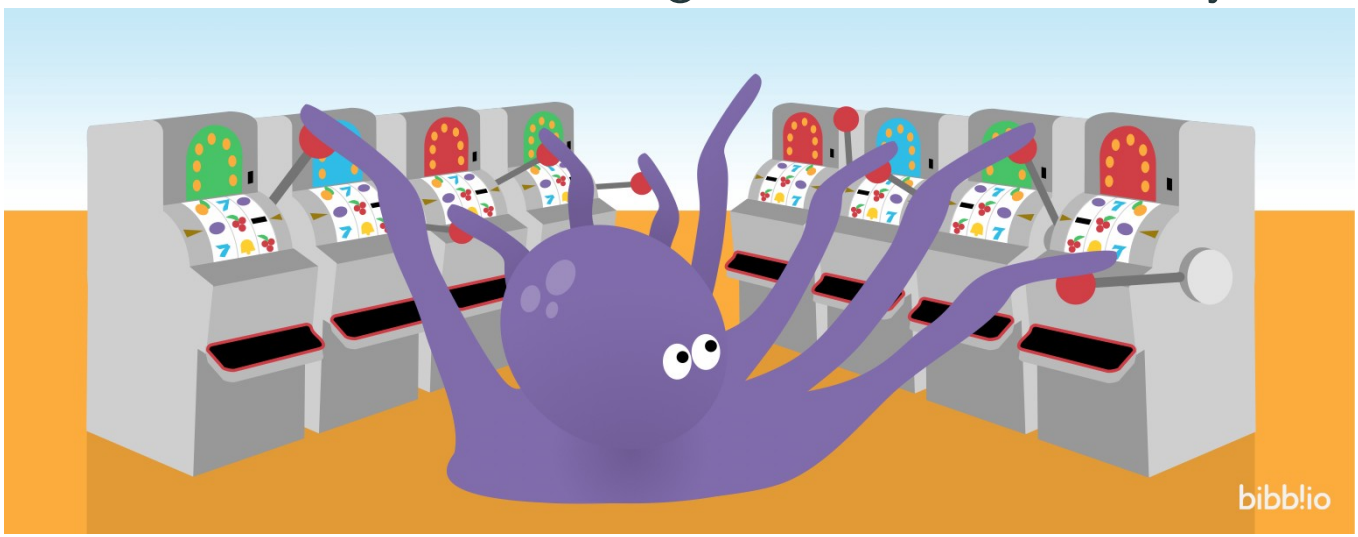
$$R(T) = (r_1 - 0.3) + (r_2 - 0.3) + (r_1 - 0.3) + (r_2 - 0.3) + (r_3 - 0.3) \ldots$$

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

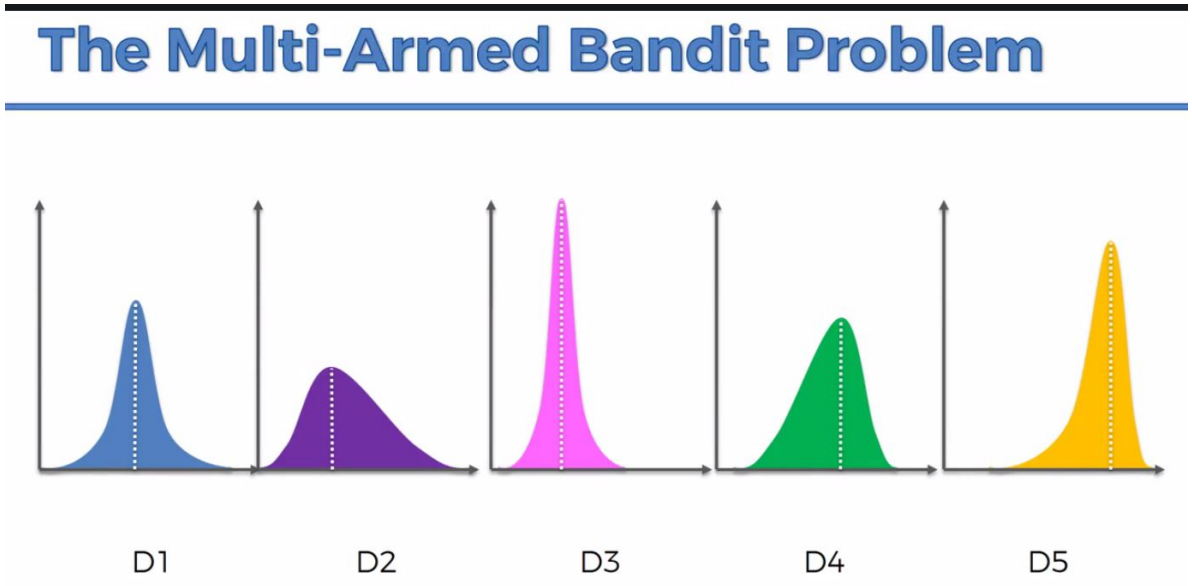
Arm 1 Arm 2 Arm 3

- The reward at each arm is stochastic (e.g., 0 with probability p_i and otherwise 1).
- Usually, the rewards are i.i.d over time. The best arm is then the arm with highest expected reward.
- We cannot observe the reward of each arm (the entire reward function): we just know the reward of the arm that we played.

Online ads example: arm = ad, reward = 1 if the user clicks on the ad and 0 otherwise

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

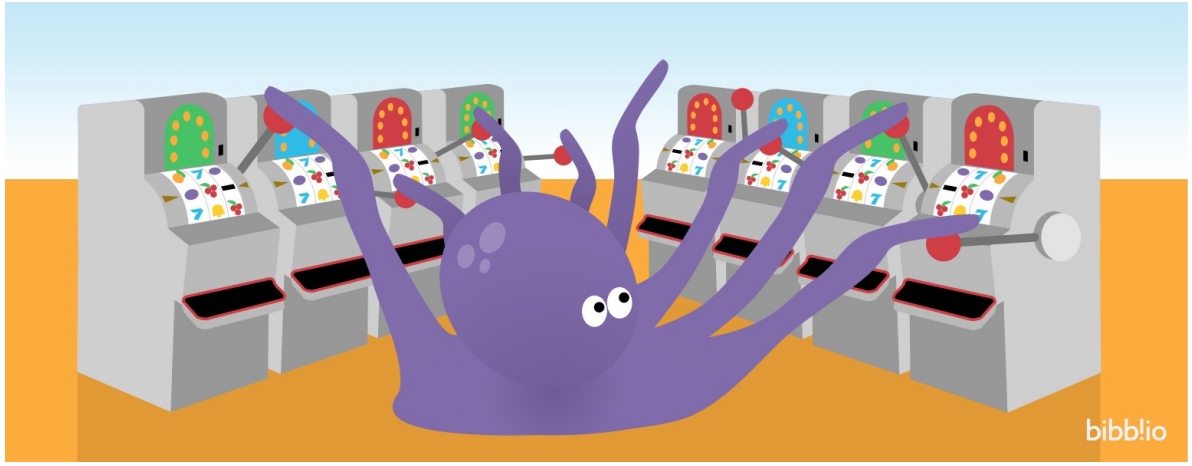
Which arm do I pick next, so that I maximize my reward over time?

$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 Lower Bound

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.

Suppose the rewards $r(i_t)$ follow a Bernoulli distribution at each arm, with expected rewards $\rho_1, \ldots, \rho_n$. Then [Lai & Robbins, 1985]

$$\liminf_{T\to\infty} \frac{R_T}{\log(T)} \geq \sum_{i:\rho^*>\rho_i} \frac{\rho^* - \rho_i}{\mathrm{kl}(\rho_i, \rho^*)}.$$

Implications of the Regret

$$\liminf_{T\to\infty} \frac{R_T}{\log(T)} \geq \sum_{i:\rho^*>\rho_i} \frac{\rho^* - \rho_i}{\mathrm{kl}(\rho_i, \rho^*)}.$$

- The minimum possible regret is $O(\log T)$. So the regret will be infinite as $T \to \infty$.
- But, the minimum average regret is $O\left(\frac{\log T}{T}\right) \to 0$ 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 $T \to \infty$.

The ϵ-Greedy Algorithm

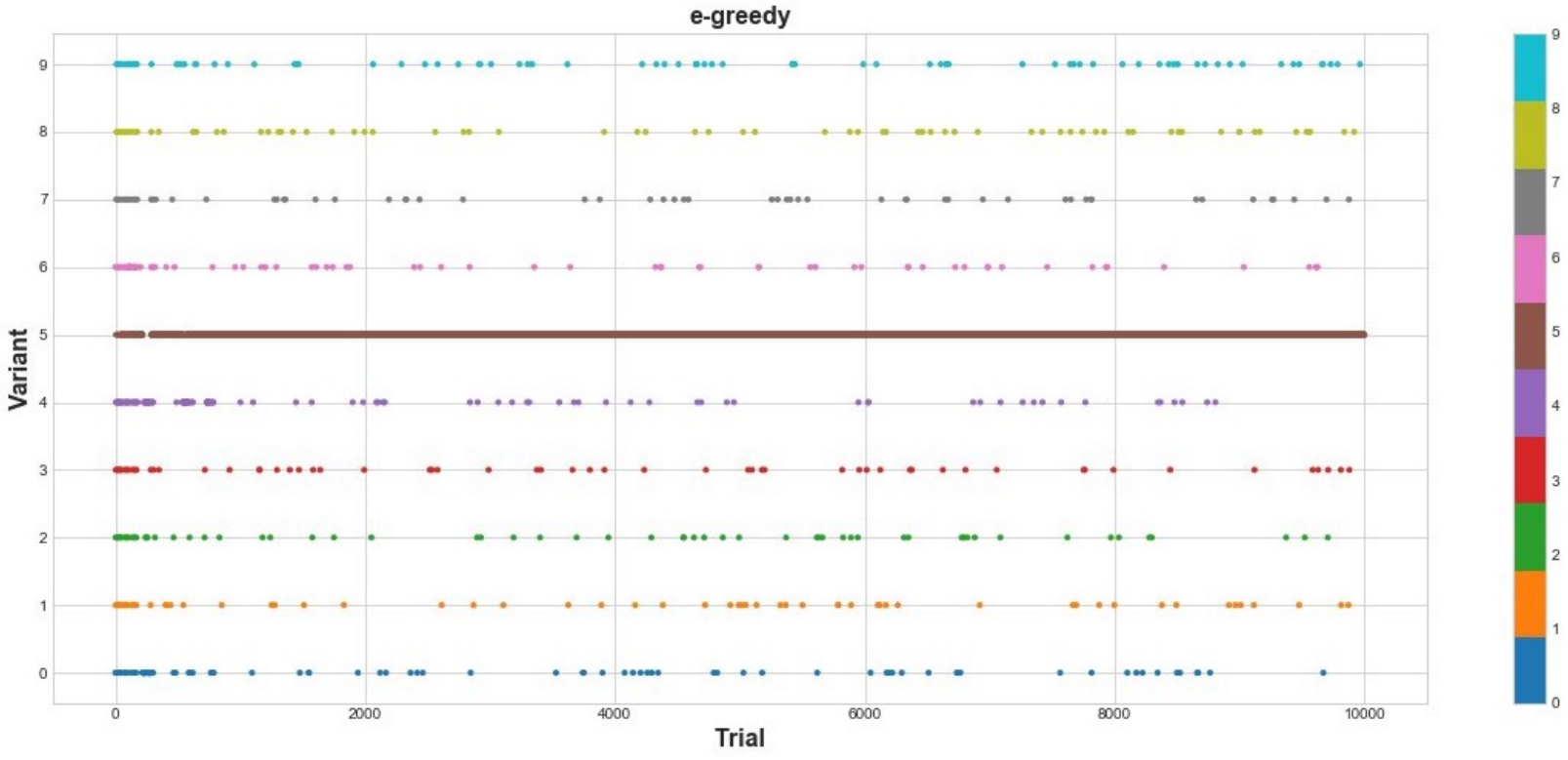
Very simple solution that is easy to implement. The idea is to exploit the best arm, but explore a random arm ϵ fraction of the time.

- Try each arm and observe the reward.
- 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.
- With probability $1 - \epsilon$, play the arm with highest $\overline{r(i)}$. Otherwise, choose an arm at random.
- Observe the reward and re-calculate $\overline{r(i)}$ for that arm.
- 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.

The UCB1 Algorithm

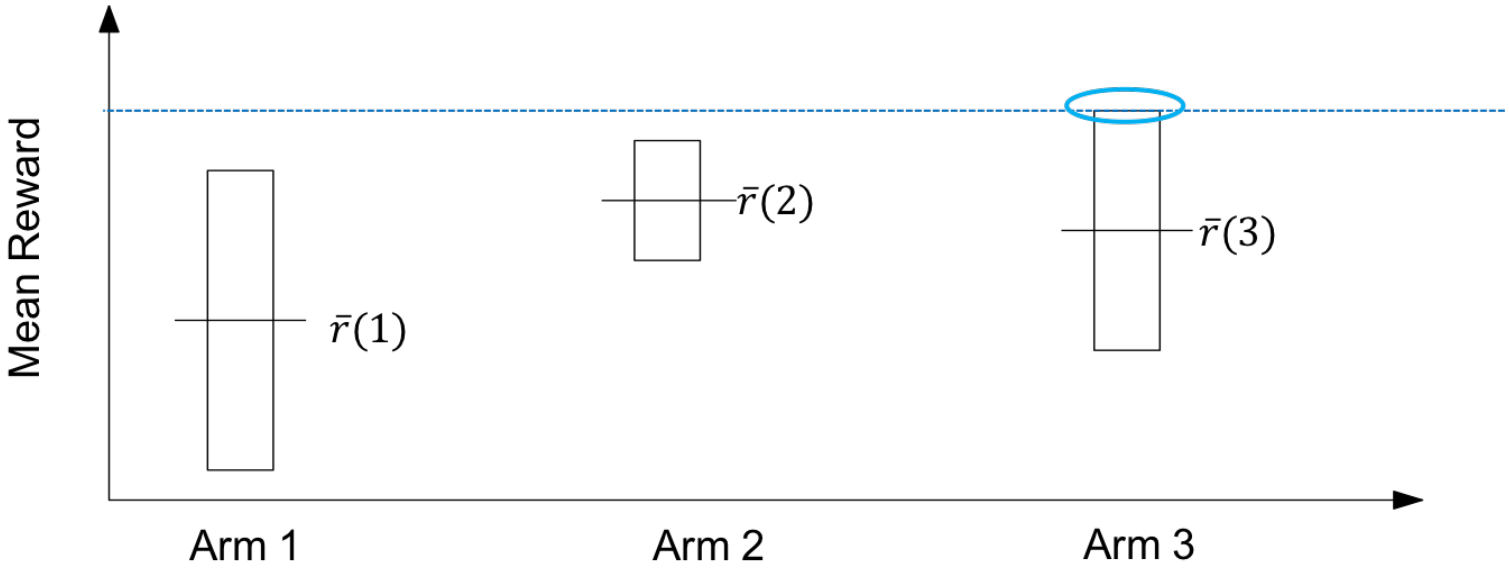
Very simple policy from Auer, Cesa-Bianchi, and Fisher (2002) with $O(\log T)$ regret. The idea is to always try the best arm, where "best" includes exploration and exploitation.

- Try each arm and observe the reward.
- 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.
- Play the arm with the highest UCB index.
- Observe the reward and re-calculate the UCB index for each arm.
- 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 $O(\log T)$ regret!

More advanced algorithms aim to reduce the constant in the $O(\log T)$ bound.

Thompson Sampling

Which arm should you pick next?

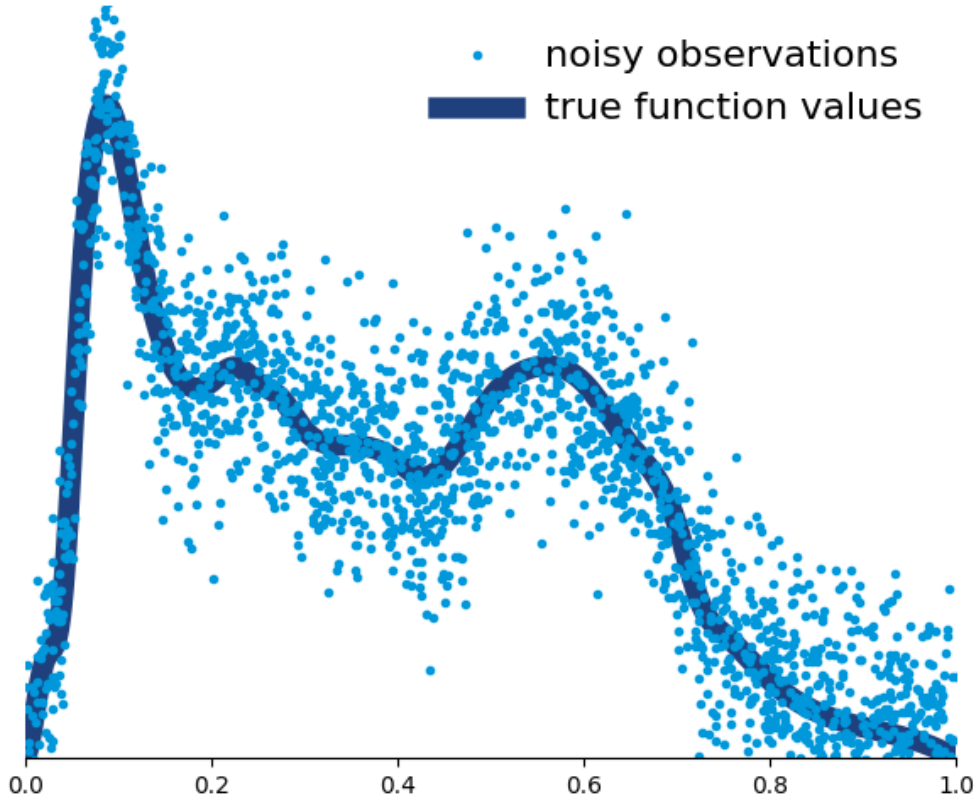
- ϵ-greedy: Best arm so far (exploit) with probability $1 - \epsilon$, otherwise random (explore).
- UCB1: Arm with the highest UCB value.

Thompson sampling instead fits a Gaussian distribution to the observations for each arm.

- Assume the reward of arm i is drawn from a normal distribution.
- Find the posterior distribution of the expected reward for each arm: $\mathcal{N}\left(\overline{r(i)}, (T_i + 1)^{-1}\right)$.
- Generate synthetic samples from this posterior distribution for each arm, which represent your understanding of its average reward.
- Play the arm with the highest sample.

Continuous Bandits

- So far we have assumed a finite number of discrete arms.
- What happens if we assume continuous arms?



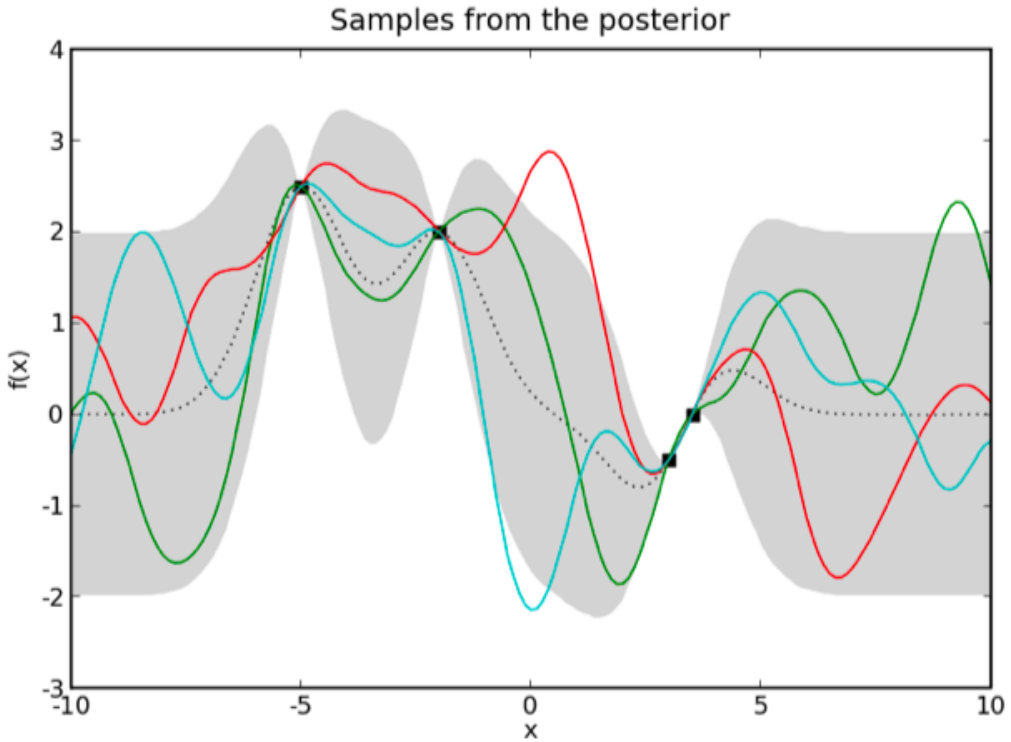
x-axis is the arm, y-axis is the (stochastic) reward.

Bayesian Optimization

Assume that $f(x)$ is drawn from a Gaussian process

$$\{f(x_1), f(x_2), \ldots, f(x_n)\} \sim N(\mathbf{0}, \mathbf{K})$$

where $\mathbf{K}$ is a kernel function, e.g., $K_{ij} = \exp(-\|x_i - x_j\|_2^2)$.



UCB and Thompson Sampling for Bayesian Optimization

Adapting UCB

UCB: computes the mean of each arm plus an upper confidence bound. We can do the same here!

- Computing the mean: Maximum likelihood estimate of $f(x)$ given our observations of $\{f(x_1), f(x_2), \ldots, f(x_n)\}$.
- Computing the confidence bound: Similarly, estimate the variance of $f(x)$ and use this as the confidence bound.

Adapting Thompson Sampling

Thompson sampling: samples from the posterior distribution of the mean reward for each arm (assuming a Gaussian distribution).

- Sampling the posterior: We already have our posterior distribution of $f(x)$, from our Gaussian process!
- Choose x: to maximize the average value of $f(x)$.

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.
- 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.
- How to extend the bandit framework to continuous arms.