Bandits

Spyros Samothrakis
Research Fellow, IADS
University of Essex

February 7, 2017

Bandits

- We will discuss bandits
- We are in effect revisiting some ideas from lecture two
  - Hypothesis testing
- I think this is a much easier to understand framework vs hypothesis testing

Examples

- You send a user an e-mail
  - User clicks on the link you get $r = 1$
  - User fails to click on the link after 3 days $r = 0$
- Playing games
  - What is the next best action to take in Chess?
    - Chess has a sequential element - hence “Reinforcement Learning”
    - But close enough...
- Online adverts
  - User clicks on an advert $(r = 1)$
  - User clicks fails to click on an advert $(r = 0)$

The bandit problem

- Bandits are a tuple $< A, R >$
- Where $a \in A$ is a set of actions
  - Sometimes actions are called “arms”
- $r \in R$ is a set of rewards
- $R(a, r) = P(r|a)$
  - The probability of getting a reward r given that I have done action a
- “You do an action, you get some feedback”

The goal

- Find an optimal policy $\pi(a) = P(a)$ that maximises the long term sum of rewards
  - Long term sum is $\sum_{t=0}^{T} = r_t$
- The "action-value" function $Q(a)$ is the expected reward for taking action a
  - $\mathrm{Q} = E[r|a]$
- The "value" function is $V = E_\pi[r]$
  - The average Q values, given a policy that I follow

Example problem

```
Dear Sir/Madam,

Best quality flasks and vials for your experiments

Click the link below to buy - discounted prices

(Link)
```

```
Dear <Name>,

This is Nick from www.MegaFlasksAndVials.com - super discounts below

(Link)
```

Let's simulate

- First e-mail is a_0
- Second e-mail is a_1
- Policy is $\pi(a_0) = 0.5, \pi(a_1) = 0.5$
- Let's manually calculate some Q's and V's on the e-mail sending problem

Goals (1)

- So our goal is to find the best action
- Optimal $V^* = \max_{a \in A} Q(a)$
- But these values can only be found through averages
  - $\hat{Q}(a)$, $\hat{V}$
- We could have done hypothesis testing...
  - But this would entail a random policy
  - Maybe we can do better

Goals (2)

- We would like to find the best action using the minimum amount of samples possible
- Keep focusing on the best action
  - While also checking making sure that other actions are sufficiently explored
- This is known as the "exploration/exploitation" dilemma

Regret (1)

- Regret is $I_t = E\left[\sum_{t=0}^{T} (V^* - Q(a_t))\right]$
  - Or, equivalently $E\left[\sum_{t=0}^{T}\left(\max_{a \in A} Q(a) - Q(a_t)\right)\right]$
- The count is $N_t(a)$, the number of times we took action a until time t
- The gap $\Delta_a = V^*(a) - Q(a)$, the difference between the optimal action and the action taken

Regret (2)

- It turns out that
  - $\sum_{a \in A} \left(E\left[N_t(a)\Delta_a\right]\right)$
- We would like to minimise the times we have large gaps
- But we have no clue what the gaps are…

Another example

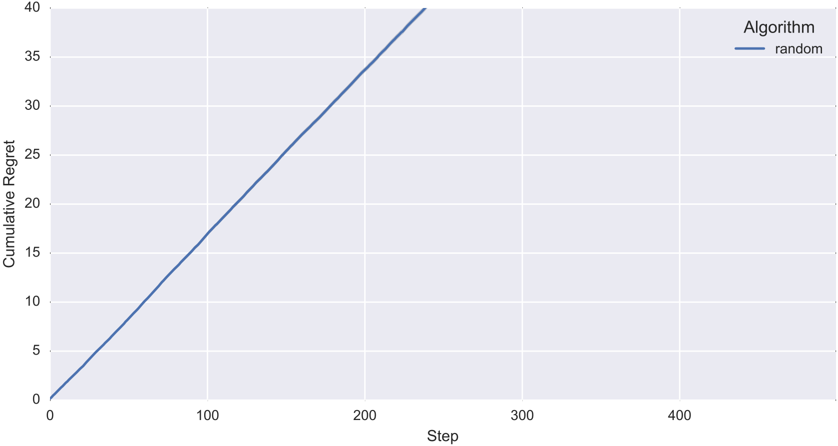
- Three actions to choose from
- Link in internal promo e-mail
  - Thus users more likely to click

```
n_actions = 3

def action_0():
    return np.random.choice([1,0], p=[0.5, 0.5])

def action_1():
    return np.random.choice([1,0], p=[0.6, 0.4])

def action_2():
    return np.random.choice([1,0], p=[0.2, 0.8])

rewards = [action_0, action_1, action_2]
```

Pure exploration

- Somewhat similiar to the A/B case
  - But in A/B you should have set a cut-off point
- You send more or less the equal number of e-mails
- Very simple setup

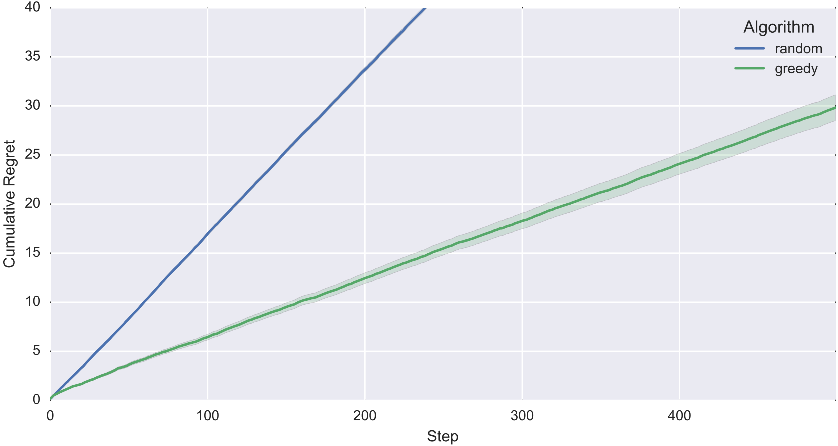
Regret of pure exploration



Greedy

- You choose the action with the highest $\hat{Q}(a)$
- Can you see a problem with this?
  - It might get stuck in suboptimal actions
- Let's try it out

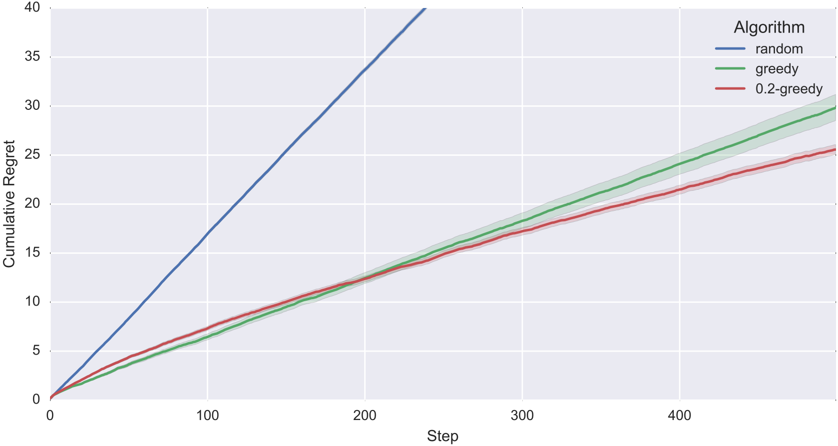
Regret of greedy



ϵ-GREEDY

- You set a small probability ϵ with which you act randomly
- The rest of the time you add greedily
  - i.e. you choose the best action
- This is a very common (but inefficient setup)
- What is the optimal ϵ?

Regret of ϵ-greedy

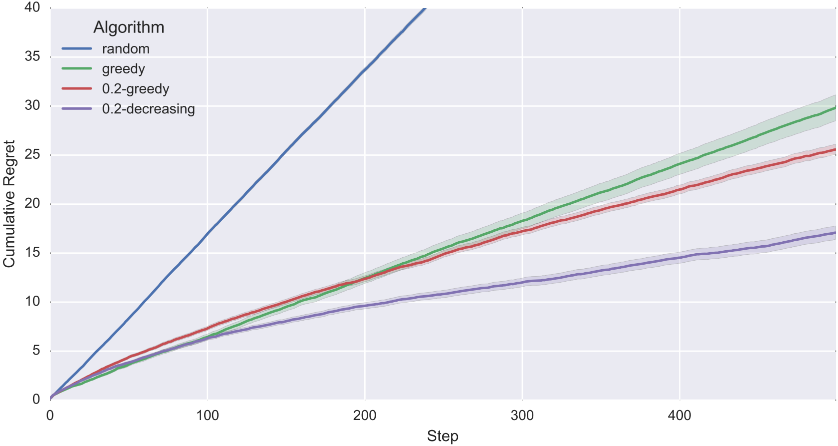


ϵ-DECREASING

- Same as epsilon greedy, but now you decrease epsilon as you choose actions
- We do

```
e *= 0.99
```

Regret of ϵ-decreasing



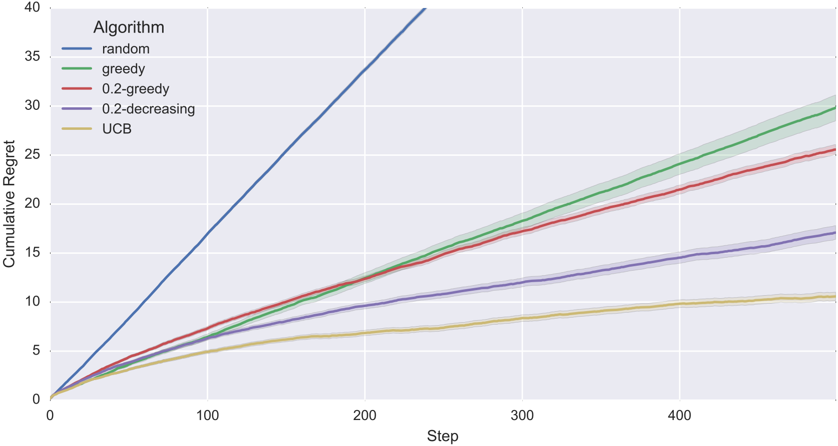
Optimism in the face of uncertainty

- There is a principle termed “optimism in face of uncertainty”
- In practical terms this means that you should try actions with highly uncertain outcomes
    - You believe the best action is the one you haven’t explored enough
- Works well in practice

Upper Confidence Bounds

- A very popular algorithm
- Fairly robust
- $UCB(a) = \hat{Q}(a) + U(a)$
- $UCB1(a) = \hat{Q}(a) + C\sqrt{\frac{log(t)}{N_t(a)}}$
- $N_t(a)$ is the times action a was executed
- t is the current timepoint/time
- $C \in [0, \inf]$ is a constant - I set it to 0.5 for the plots below
  - Can you guess what the effect of C is?

Regret of upper confidence bounds



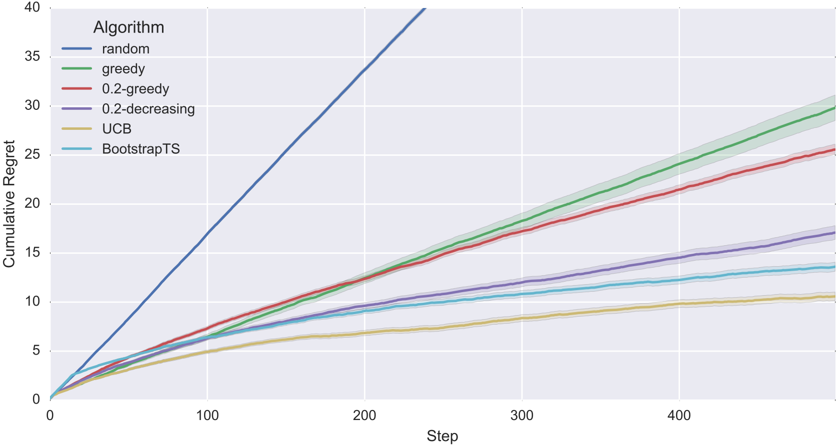
Bootstrap thompson sampling

- What if you could take bootstrap samples of action rewards that we have collected?
- You would have incorporated the uncertainty within your bootstrap samples
- If you have a large number of bootstrap samples you have a distribution over possible $\hat{Q}(s)$
- Sample from this distribution
- A version of probability matching
  - $\pi(a) = P[\hat{Q}(a) > \hat{Q}(a'), \quad \forall a' \in A]$

Priors

- You can get stuck here as well (like greedy)
- Add some pseudo-rewards
- Or act randomly a bit

Regret of Bootstrap Thomson Sampling



Code

```
class Bandit(object):
    def __init__(self,n_actions):
        self.counts = np.zeros(n_actions)
        self.action_rewards = [[] for i in range(n_actions)]
        self.rewards = []
        self.n_actions = n_actions

    def select_action(self):
        """Selection which arm/action to pull"""
        pass

    def update(self,action,reward):
        """Update the actions"""
        self.counts[action] = self.counts[action] + 1
        self.action_rewards[action].append(reward)
        self.rewards.append(reward)


    def get_Q_values(self):
        Q_values = []
        for q_v in self.action_rewards:
             Q_values.append(np.array(q_v).mean())

        return np.array(Q_values)

    def get_V_value(self):
        return np.array(self.v_value.mean())
```

Change of rewards

- What if rewards just change
- Because people are bored of your e-mails
    - They talk to each other
    - Out of fashion
- You might want to have continuous adaptation
- Keeping all values and finding $\hat{Q}(a)$ is expensive
    - What happens in e-mail 1000? e-mail 100K?
    - How many additions?

The sequential case

- What if you are to take a series of actions?
- Surely your current action depends on your future actions
- Hence there is going to be a change in the distribution of rewards
    - Induced by the experimenter
- “Reinforcement Learning”

Example e-mail campaign

- You send your first e-mail
  - "Please buy this product"
- Send second e-mail
  - "Will you buy the add-on?"
- Send third e-mail
  - "Let us service your product"
- You want to maximise your rewards
- Creates a tree of possible actions

Tree

- Let's draw the tree of the above example
  - Three different actions for each "state"
- What do you observe?

Introducing state

- $s \in S$ can be used to differentiate between different "states", conditioning π, V and Q values on states
- $\pi(s, a)$, $V(s)$, $Q(s, a)$
- e.g. in the example above, you have $Q(``firstemail'', ``emailtypeA'')$
- Let's write the rest of the states, the policies, V and Q-Values

Incremental calculation of a mean

$\mathrm{v_t}$ can be the reward or the sum of rewards you got at different steps

$$\hat{Q}_t(s,a) = \hat{Q}_{t-1}(s,a) + \overbrace{\frac{\mathrm{v_t} - \hat{Q}_{t-1}(s,a)}{t}}^{\textbf{Error}}$$

$$\hat{Q}_t(s,a) = \hat{Q}_{t-1}(s,a) + \frac{1}{t}\overbrace{\mathrm{v_t} - \hat{Q}_{t-1}(s,a)}^{\textbf{Error}}$$

$$\hat{Q}_t(s,a) = \hat{Q}_{t-1}(s,a) + \alpha\left[\mathrm{v_t} - \hat{Q}_{t-1}(s,a)\right]$$

Incremental bootstrap

Oza, Nikunj C., and Stuart Russell "Online bagging and boosting." Systems, man and cybernetics, 2005 IEEE international conference on. Vol. 3. IEEE, 2005.

- We will implement this in the labs

Equilibria

- We will discuss (very) briefly the notion of equilibria
  - Imagine you are putting up large advert banners on your website
  - They hide content
  - User can click on the top right corner and quit the banner
- Where should you put the banner?
- How often should the banner pop-up?

Adversarial bandits

- Most bandits we discussed until now assume the environment is indifferent
- i.e. the user will click in the link if she thinks it is interesting for her to click
- But quite often, people are annoyed by your efforts - so they will try to “adapt” around you
  - Close the advert-super fast without thinking
- Solution - put the advert in random places
  - Mixed policies
- Exp3 - but not now

Rethinking states

- States as we have defined them until now are black solid boxes
  - They can only be enumerated
- i.e. state s_0, state s_1
- What if a state could be decomposed into a set of features?
  - $sex, age, married, job...$
- Highly reminiscent of supervised learning
  - We are given features, we would like to predict the reward - i.e. the outcome!
- We could now do something that looks like regression!

Combining states and actions

- So you now have features that you can encode
- Various encoding strategies
  - One regressor per action
  - A single regressor with dummy encoded actions
- Let's do an example
- What could be a problem if you don't have separate regressors for each action?

ϵ-greedy and ϵ-decreasing

- Set ϵ to some small value
- Keep decreasing...
- Very popular because of its simplicity
- You need to be smart about your decreasing schedule
  - Possibly set some lower bound

Bootstrap Thomson Sampling

- Get a bootstrap sample of all your data
- Learn a regressor
- Act greedily using the regressor you learned
- Repeat

Conclusion

- First hit on bandits
- Super-exciting research area
- Used quite a bit on website optimisation and recommender systems
- We will delve deeper in the adversarial case and recommender systems in the future
- Again, the bootstrap saves the day