Part 3. Bandit Feedback and Likelihood Models for Recommendation

David Rohde, Flavian Vasile

Criteo Research

June 24, 2019

Collaborators:

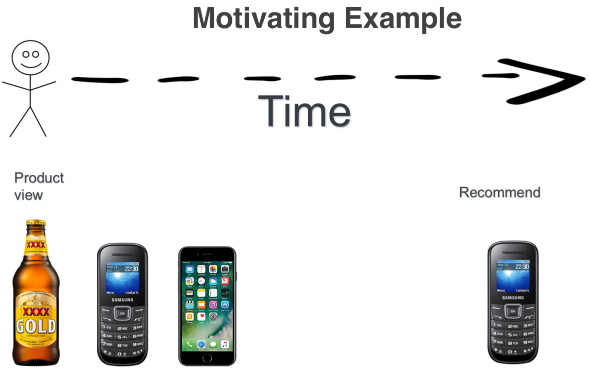
Special thanks to:

- Olivier Jeunen
- Dmytro Mykhaylov
- Stephen Bonner

Structure of the talk

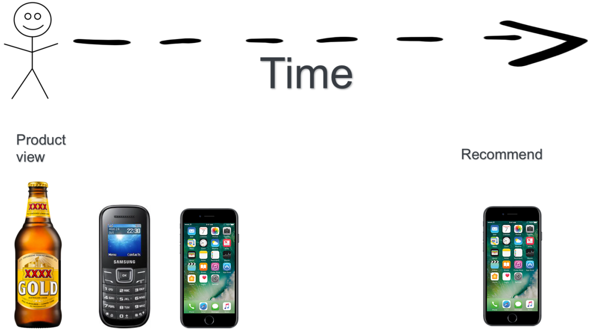
- Recommendation Revisited: Acadmic vs Industry Reco

The Reco Problem Revisited

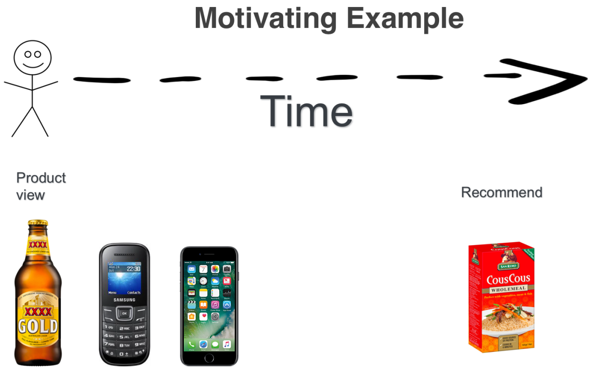


The Reco Problem Revisited

Motivating Example



The Reco Problem Revisited

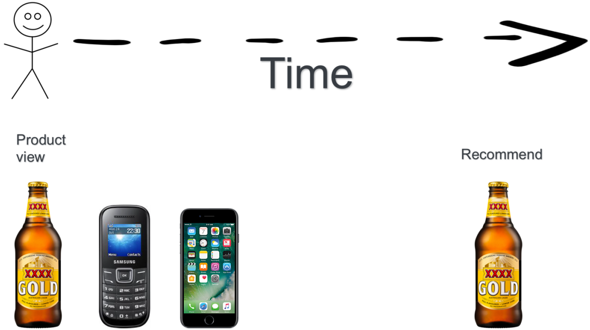


The Reco Problem Revisited

Motivating Example

The Reco Problem Revisited

Motivating Example



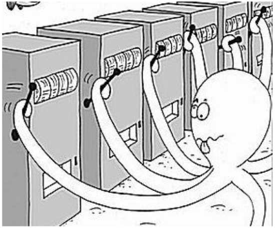
History: Split History of Recommender Systems

Next Item or Missing Link Prediction

e.g. Netflix Prize or Movie Lens

	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	movie 7	movie 8	movie 9	movie 10
user 1			1		2					
user 2		2		3	3			4		
user 3							5	3		4
user 4	2				3			2		
user 5		4				5			3	
user 6			2							
user 7			2					4	2	3
user 8	3	4				4				
user 9									3	
user 10			1		2					

Computational Advertising, e.g. Bandits, Conterfactual Risk Minimisation, Contextual Bandits, Reinforcement Learning.



Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens:

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize:

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize: no, explicit feedback of movie ratings

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize: no, explicit feedback of movie ratings
- Yoochoose (RecSys 15):

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize: no, explicit feedback of movie ratings
- Yoochoose (RecSys 15): no, implicit session based behaviour

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize: no, explicit feedback of movie ratings
- Yoochoose (RecSys 15): no, implicit session based behaviour
- 30 Music

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize: no, explicit feedback of movie ratings
- Yoochoose (RecSys 15): no, implicit session based behaviour
- 30 Music no, implicit session based behaviour

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize: no, explicit feedback of movie ratings
- Yoochoose (RecSys 15): no, implicit session based behaviour
- 30 Music no, implicit session based behaviour
- Criteo Dataset for evaluation of Counterfactual Algorithms:

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize: no, explicit feedback of movie ratings
- Yoochoose (RecSys 15): no, implicit session based behaviour
- 30 Music no, implicit session based behaviour
- Criteo Dataset for evaluation of Counterfactual Algorithms: yes!

Classic RecSys Datasets: Are they logs of recommender systems?

- MoiveLens: no, explicit feedback of movie ratings
- Netflix Prize: no, explicit feedback of movie ratings
- Yoochoose (RecSys 15): no, implicit session based behaviour
- 30 Music no, implicit session based behaviour
- Criteo Dataset for evaluation of Counterfactual Algorithms: yes!

The Criteo Dataset shows a log of recommendations and if they were successful in getting users to clicks.

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K,

criteo

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG,

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG, no, evaluates next item prediction

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG, no, evaluates next item prediction
- Log likelihood

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG, no, evaluates next item prediction
- Log likelihood unclear, but ususally next item prediction

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG, no, evaluates next item prediction
- Log likelihood unclear, but ususally next item prediction
- Inverse Propensity Score estimate of click through rate

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG, no, evaluates next item prediction
- Log likelihood unclear, but ususally next item prediction
- Inverse Propensity Score estimate of click through rate yes! (although it is often noisy)

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG, no, evaluates next item prediction
- Log likelihood unclear, but ususally next item prediction
- Inverse Propensity Score estimate of click through rate yes! (although it is often noisy)
- AB Test

criteo

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG, no, evaluates next item prediction
- Log likelihood unclear, but ususally next item prediction
- Inverse Propensity Score estimate of click through rate yes! (although it is often noisy)
- AB Test yes, but the academic literature has no access to this

Classic RecSys Evaluation Metrics: Do they evaluate the quality of recommendation?

- Recall@K, Precision@K, HR@K, no, evaluates next item prediction
- DCG, no, evaluates next item prediction
- Log likelihood unclear, but ususally next item prediction
- Inverse Propensity Score estimate of click through rate yes! (although it is often noisy)
- AB Test yes, but the academic literature has no access to this

If the dataset does not contain a log of recommendations and if they were successful, you cannot compute metrics of the recommendation quality.

Let's take a step back...

How are we improving large-scale Recommender Systems in Real World

- Learn a supervised model from past user activity
- Evaluate offline and decide or not to A/B test
- A/B test
- If positive and scalable, roll-out
- If not, try to understand what happened and try to create a better model of the world using the same past data
- Repeat

Supervised learning with the wrong objective

Most of the time, we frame the problem either as a:

- Missing link prediction
- Next user event prediction

Supervised learning with the wrong objective

We are operating under the assumption that the best Recommendation Policy is in some sense the **optimal auto-complete of natural user behavior**

Supervised learning with the wrong objective

From the point of view of business metrics, **learning to autocomplete behavior is a great initial recommendation policy**, especially when no prior user feedback is available.

Supervised learning with the wrong objective

However, after a first golden age, where all offline improvements turn into positive A/B tests, the *naive Recommendation* optimization objective and the business objective will start to diverge.

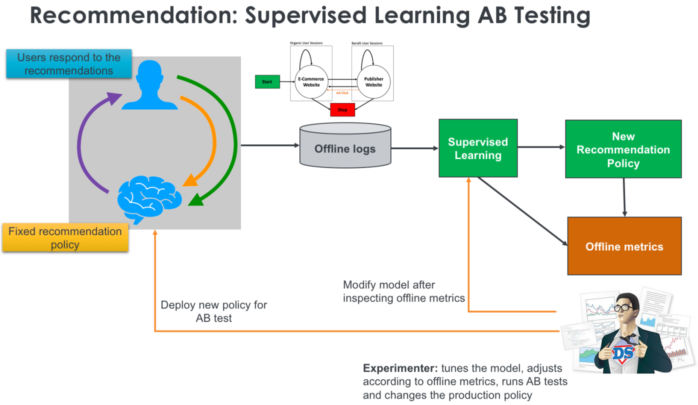
Aligning the Recommendation objective with the business

- Of course, we could start incorporating user feedback that we collected while running the initial recommendation policies
- We should be able to continue bringing improvements using feedback that is now aligned with our business metrics (ad ctr, post click sales, dwell time, number of videos watched, ...)

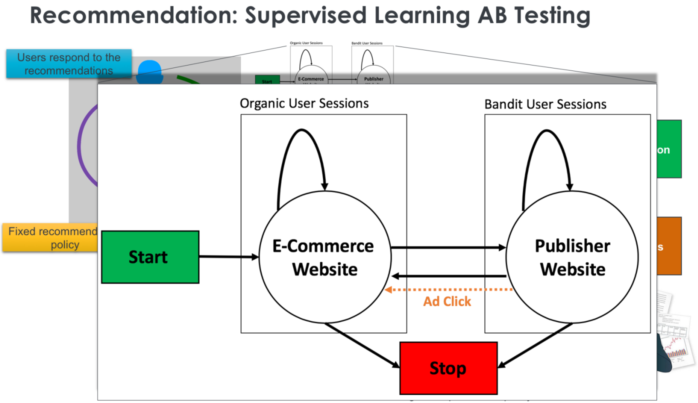
Aligning the Recommendation objective with the business

However, this does not come without caveats...

Recommendation as Supervised Learning and AB Testing



Recommendation as Supervised Learning and AB Testing



Modern Recommendation Systems Research

Q: How does the literature on Large Scale Recommendation Systems look right now?

A: Most of the latest publications are talking about ways of using Deep Learning for Recommendation:

- **Matrix Factorization extensions:** Word2Vec, Deep and Wide, Neural MF, Node2Vec
- **Content-based recommendation:** CNNs for Image, Text, Sounds to compute item similarities
- **Next event prediction / user activity modeling:** RNNs, TCNs

Modern Recommendation Systems Research

We see great improvements in offline metrics!

- **Regression/Classification metrics:** MSE, AUC, NLL
- **Ranking metrics:** MPR, Precision@k, NDCG

criteo

What is wrong with this picture?

In the same time, as practicioners, we see difficulties in improving the Real World Recommendation models!

Offline - online metrics alignment for Recommendation is a recognized problem:

- **Previous work:** Missing Not At Random (MNAR) framework for Matrix factorization, Bandits Literature
- **New avenues:** *REVEAL: Offline evaluation for recommender systems Workshop at RecSys 2019* (this September in Copenhagen) `https://recsys.acm.org/recsys19/reveal/`

The relationship between Reinforcement Learning and Recommendation

- We are doing Reinforcement Learning by hand!

The relationship between Reinforcement Learning and Recommendation

- We are doing Reinforcement Learning by hand!
- Furthermore, we are trying to do RL using the Supervised Learning Framework

The relationship between Reinforcement Learning and Recommendation

- We are doing Reinforcement Learning by hand!
- Furthermore, we are trying to do RL using the Supervised Learning Framework
- Standard test data sets do not let us explore this aspect of recommender systems

The relationship between Reinforcement Learning and Recommendation

- We are doing Reinforcement Learning by hand!
- Furthermore, we are trying to do RL using the Supervised Learning Framework
- Standard test data sets do not let us explore this aspect of recommender systems

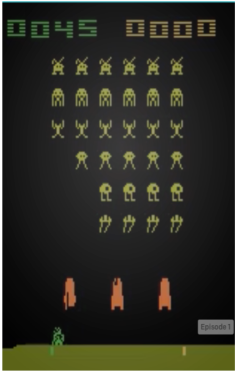
.. but how do you evaluate a reinforcement learning algorithm?

Open AI Gym

Problem: The reinforcement learning community lacked a common set of tools to allow comparison of reinforcement learning algorithms.

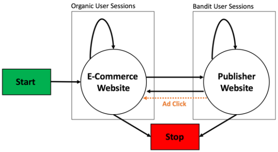
Open AI Gym: 2016 (Brockman et al.): Software standard (Python api) that allows comparison of the performance of reinforcement learning algorithms.

Core idea: environments (the problem) and agents (the RL algorithm) interact.



Introducing RecoGym

RecoGym: Simulates User Behavior (both Organic and Bandit)



- Allows online evaluation of new recommendation policies in a simulated environment
- Holistic view of user (organic and bandit) provides a framework for categorizing recommendation algorithms
- Key feature: adjustable parameter that changes the effectiveness of “Pure Organic” or next item prediction algorithms.

Algorithm 1: A simple Simulator

Input : $S \in \mathcal{R}^{3\times3}$ transition matrix between organic and bandit, $\Gamma \in \mathcal{R}^{P\times K}$ organic embeddings, μ_Γ organic popularity $\boldsymbol{\beta} \in \mathcal{R}^{P\times K}$ bandit embeddings, μ_β non-personalised ctr contribution $f(\cdot)$ montonic increasing function accounting for ad fatigue $\boldsymbol{m}$, U number of users, P number of products.

Output: Sequence of organic and bandit events

for $u \in 1..U$ **do**
  $t \leftarrow 0$
  $z_{u,0} \leftarrow$ organic
  $r_{u,0} \leftarrow$ undef
  $c_{u,0} \leftarrow$ undef
  $\boldsymbol{\omega}_{u,0} \sim \mathcal{N}(\boldsymbol{0}_{K\times1}, \boldsymbol{I}_K)$
  $v_{u,0} \sim \text{Categorical}(\text{softmax}(\Gamma\boldsymbol{\omega}_{u,0}))$
  while $z_{u,t} \neq stop$ **do**
    $t \leftarrow t+1$
    $\boldsymbol{\omega}_{u,t} \sim \mathcal{N}(\boldsymbol{\omega}_{u,t-1}, \sigma_\omega^2 \boldsymbol{I}_K)$
    if $c_{u,t-1} \neq 1$ **then**
      $z_{u,t} \sim \text{Categorical}(S_{z_{u,t-1},\text{organic}}, S_{z_{u,t-1},\text{bandit}}, S_{z_{u,t-1},\text{stop}})$
    else
      $z_{u,t} =$ organic
    end
    if $z_{u,t} = organic$ **then**
      $v_{u,t} \sim \text{Categorical}(\text{softmax}(\Gamma\boldsymbol{\omega}_{u,t} + \mu_\Gamma))$
      $r_{u,t} \leftarrow$ undef
      $c_{u,t} \leftarrow$ undef
    end
    if $z_{u,t} = bandit$ **then**
      $r_{u,t}$ is geneated from the policy
      $c_{u,t} \sim \text{Bernoulli}([f(\boldsymbol{\beta}\boldsymbol{\omega}_{u,t} + \mu_\beta)]_{r_{u,t}})$
    end
  end
end

criteo.

$$v_0 \sim \text{categorical}\left(\text{softmax}\left(\begin{pmatrix}\Gamma_{1,1} & . & \Gamma_{1,K}\\ . & . & .\\ . & . & .\\ \Gamma_{P,1} & . & \Gamma_{P,K}\end{pmatrix}\begin{pmatrix}\omega_{0,1}\\ .\\ \omega_{0,K}\end{pmatrix}+\begin{pmatrix}\mu_1\\ .\\ .\\ \mu_P\end{pmatrix}\right)\right)$$

$$v_1 \sim \text{categorical}\left(\text{softmax}\left(\begin{pmatrix}\Gamma_{1,1} & . & \Gamma_{1,K}\\ . & . & .\\ . & . & .\\ \Gamma_{P,1} & . & \Gamma_{P,K}\end{pmatrix}\begin{pmatrix}\omega_{1,1}\\ .\\ \omega_{1,K}\end{pmatrix}+\begin{pmatrix}\mu_1\\ .\\ .\\ \mu_P\end{pmatrix}\right)\right)$$

$$v_2 \sim \text{categorical}\left(\text{softmax}\left(\begin{pmatrix}\Gamma_{1,1} & . & \Gamma_{1,K}\\ . & . & .\\ . & . & .\\ \Gamma_{P,1} & . & \Gamma_{P,K}\end{pmatrix}\begin{pmatrix}\omega_{2,1}\\ .\\ \omega_{2,K}\end{pmatrix}+\begin{pmatrix}\mu_1\\ .\\ .\\ \mu_P\end{pmatrix}\right)\right)$$

f is an increasing function

$$f : \mathbb{R} \to (0,1)$$

$$c_3|r_3 \sim \text{Bernoulli}\left(\left[f\left(\begin{pmatrix}\beta_{1,1} & . & \beta_{1,K}\\ . & . & .\\ . & . & .\\ \beta_{P,1} & . & \beta_{P,K}\end{pmatrix}\begin{pmatrix}\omega_{3,1}\\ .\\ \omega_{3,K}\end{pmatrix}+\begin{pmatrix}\mu_1^*\\ .\\ .\\ \mu_P^*\end{pmatrix}\right)\right]_{r_3}\right)$$

$$c_4|r_4 \sim \text{Bernoulli}\left(\left[f\left(\begin{pmatrix}\beta_{1,1} & . & \beta_{1,K}\\ . & . & .\\ . & . & .\\ \beta_{P,1} & . & \beta_{P,K}\end{pmatrix}\begin{pmatrix}\omega_{4,1}\\ .\\ \omega_{4,K}\end{pmatrix}+\begin{pmatrix}\mu_1^*\\ .\\ .\\ \mu_P^*\end{pmatrix}\right)\right]_{r_4}\right)$$

$$v_5 \sim \text{categorical}\left(\text{softmax}\left(\begin{pmatrix}\Gamma_{1,1} & . & \Gamma_{1,K}\\ . & . & .\\ . & . & .\\ \Gamma_{P,1} & . & \Gamma_{P,K}\end{pmatrix}\begin{pmatrix}\omega_{5,1}\\ .\\ \omega_{5,K}\end{pmatrix}+\begin{pmatrix}\mu_1\\ .\\ .\\ \mu_P\end{pmatrix}\right)\right)$$

time

The RecoGym Session

```
In [1]: import gym, reco_gym

        # env_0_args is a dictionary of default parameters (i.e. number of products)
        from reco_gym import env_1_args, Configuration

        # You can overwrite environment arguments here:
        env_1_args['random_seed'] = 41

        # Initialize the gym for the first time by calling .make() and .init_gym()
        env = gym.make('reco-gym-v1')
        env.init_gym(env_1_args)


        env.reset() # we call request to move to the first user
```

```
In [2]: observation, reward, done, info = env.step(None)
        # We specify None because we have need to learn about the user before we can act
```

```
In [3]: observation.current_sessions

Out[3]: [{'t': 0, 'u': 0, 'z': 'pageview', 'v': 4},
         {'t': 1, 'u': 0, 'z': 'pageview', 'v': 4},
         {'t': 2, 'u': 0, 'z': 'pageview', 'v': 4},
         {'t': 3, 'u': 0, 'z': 'pageview', 'v': 0}]
```

```
In [4]: # We see the user is interested in product id 4 (they viewed it 3 times) and product id 0 (they viewed it once)
```

criteo

The RecoGym Session

```
In [5]: observation, reward, done, info = env.step(0)

In [6]: reward # the user does not click on our recommendation of product 0
Out[6]: 0

In [7]: observation.current_sessions # the user does not view any more products, so we learn nothing more about them
Out[7]: []
```

The RecoGym Session

```
In [8]: observation, reward, done, info = env.step(0) # we recommend product 0 again

In [9]: reward # they do not click again, ctr are usually low - this is not surprising

Out[9]: 0

In [10]: observation.current_sessions # again no additional product views the bandit session continues

Out[10]: []
```

The RecoGym Session

```
In [11]: observation, reward, done, info = env.step(4) # we now recommend product 4

In [12]: reward # they clicked!  We must have done something right on that last recommendation

Out[12]: 1

In [13]: observation.current_sessions # the user moved to the retailer website and viewed the product

Out[13]: [{'t': 7, 'u': 0, 'z': 'pageview', 'v': 4}]
```

Unlike RL we continue to use logs

Let's start with a random logging policy (a crazy thing to do, but a useful theoretical concept).

```
In [17]: env.generate_logs(100)
Out[17]:
```

	a	c	ps	ps-a	t	u	v	z
0	NaN	NaN	NaN	None	0	0	2.0	organic
1	NaN	NaN	NaN	None	1	0	4.0	organic
2	NaN	NaN	NaN	None	2	0	4.0	organic
3	NaN	NaN	NaN	None	3	0	4.0	organic
4	NaN	NaN	NaN	None	4	0	2.0	organic
5	NaN	NaN	NaN	None	5	0	4.0	organic
6	NaN	NaN	NaN	None	6	0	5.0	organic
7	NaN	NaN	NaN	None	7	0	4.0	organic
8	NaN	NaN	NaN	None	8	0	4.0	organic
9	NaN	NaN	NaN	None	9	0	4.0	organic
10	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	10	0	NaN	bandit
11	1.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	11	0	NaN	bandit
12	2.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	12	0	NaN	bandit
13	1.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	13	0	NaN	bandit
14	8.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	14	0	NaN	bandit
15	2.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	15	0	NaN	bandit
16	5.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	16	0	NaN	bandit
17	1.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	17	0	NaN	bandit
18	7.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	18	0	NaN	bandit
19	7.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	19	0	NaN	bandit
20	5.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	20	0	NaN	bandit

Organic Best Of vs Bandit Best Of

Exercise - Getting Started with RecoGym

Go to the notebook “Module III A.ipynb”

- Examine the logs, to start with we will look at non-personalised behavior only.
- Plot a histogram of organic product popularity. What is the most popular product?
- Plot the (non-personalised) click through rate of each product (with an error analysis). What is the most clicked on product?
- Plot popularity against click through rate. How good is popularity a proxy for click through rate?
- Simmulate an AB test comparing a simple recommender system (agent) that always recommends the highest ctr product with a recommender system that always recommends the organically most popular product.

Harder questions

Reflect on how the logging policy would affect these results.

Harder questions

Reflect on how the logging policy would affect these results.

What is personalisation? What impact does personalisation bring?

Harder questions

Reflect on how the logging policy would affect these results.

What is personalisation? What impact does personalisation bring?

Where does traditional academmic recommender systems research fit into this picture?

To consult the statistician after an experiment is finished is often merely to ask him to conduct a post mortem examination. He can perhaps say what the experiment died of. Ronald Fisher

Roland Fisher

Answer Q1 Histogram of product popularity

```
clicks = np.zeros(NumberOfProducts)
bandits = data[data['z'] == 'bandit']
for product_id in range(NumberOfProducts):
    actions = bandits[bandits['a'] == product_id]
    clicks[product_id] = np.sum(actions[actions['c'] == 1]['c'])

print("Clicks: ", clicks)

_, ax = plt.subplots()
ax.set_title('Histogram of Clicks per Product')

ax.bar(range(NumberOfProducts), clicks)
plt.show()
```

Answer Q2 non-personalised CTR

```
from scipy.stats.distributions import beta

clicks = np.zeros(NumberOfProducts)
impressions = np.zeros(NumberOfProducts)
lower_errors = np.zeros(NumberOfProducts)
upper_errors = np.zeros(NumberOfProducts)
bandits = data[data['z'] == 'bandit']
for product_id in range(NumberOfProducts):
    actions = bandits[bandits['a'] == product_id]
    clicks[product_id] = np.sum(actions[actions['c'] == 1]['c'])
    impressions[product_id] = sum(actions['a']==product_id)
```

Answer Q3 Best of CTR

```
top_ctr_item = np.argmax(clicks/impressions)
```

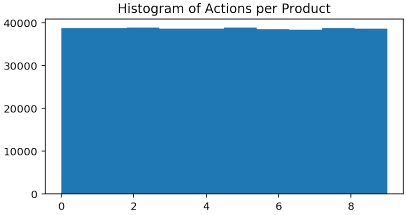
Answer Q4 most views

```
top_viewed_item = np.argmax(views)
```

```
data = deepcopy(env).generate_logs
        (ABTestNumberOfUsers, agent=organic_counter_agent)
```

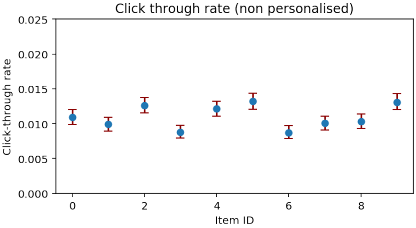
Organic Best Of vs Bandit Best Of

We can exampine the click through rate of each action under this policy, we see that action 5, 2 and 9 have high click through rates.



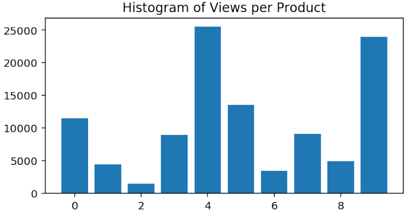
Organic Best Of vs Bandit Best Of

We can also examine which items are organically the most popular, this time we see that item 4 and item 9 are the most popular.



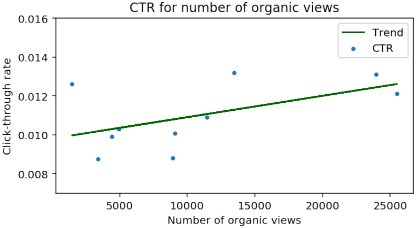
Organic Best Of vs Bandit Best Of

The most organically popular items are items 4 and 9.



Organic Best Of vs Bandit Best Of

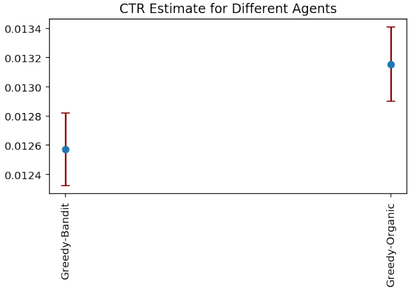
If we plot popularity vs non-personalised click through rate we see some kind of relationship, but the correlation is noisy.



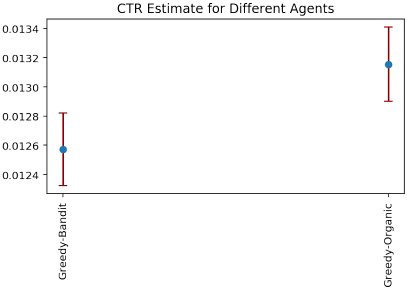
Organic Best Of vs Bandit Best Of

A bandit best of agent always recommends the highst ctr product (product 5). An organic best of always recommends the most frequently viewed product (product 4). A simmulated AB test shows using the bandit best of yields a better result.

Organic Best Of vs Bandit Best Of



Organic Best Of vs Bandit Best Of



Personalisation of course will result in a better result still.

Evaluate an organic agent using the Bandit Signal

criteo

The cold start scenario

- You need to build a recommender system for a client, they have an existing system with existing organic traffic.

The cold start scenario

- You need to build a recommender system for a client, they have an existing system with existing organic traffic. This means you can build standard next item predictions to build models of user behavior and understand how user's preferences cluster together.

The cold start scenario

- You need to build a recommender system for a client, they have an existing system with existing organic traffic. This means you can build standard next item predictions to build models of user behavior and understand how user's preferences cluster together.
- They are currently running a recommender system with a random policy.

The cold start scenario

- You need to build a recommender system for a client, they have an existing system with existing organic traffic. This means you can build standard next item predictions to build models of user behavior and understand how user's preferences cluster together.
- They are currently running a recommender system with a random policy.
- Your task is to build a next item prediction model as a proxy for a recommendation algorithm. Then produce offline metrics to convince the engineering team to run an AB test.

The cold start scenario

- You need to build a recommender system for a client, they have an existing system with existing organic traffic. This means you can build standard next item predictions to build models of user behavior and understand how user's preferences cluster together.
- They are currently running a recommender system with a random policy.
- Your task is to build a next item prediction model as a proxy for a recommendation algorithm. Then produce offline metrics to convince the engineering team to run an AB test.
- Finally you evaluate your performance against production.

criteo

The cold start scenario

- You need to build a recommender system for a client, they have an existing system with existing organic traffic. This means you can build standard next item predictions to build models of user behavior and understand how user's preferences cluster together.
- They are currently running a recommender system with a random policy.
- Your task is to build a next item prediction model as a proxy for a recommendation algorithm. Then produce offline metrics to convince the engineering team to run an AB test.
- Finally you evaluate your performance against production.

Please look at notebook "Module III B.ipynb"

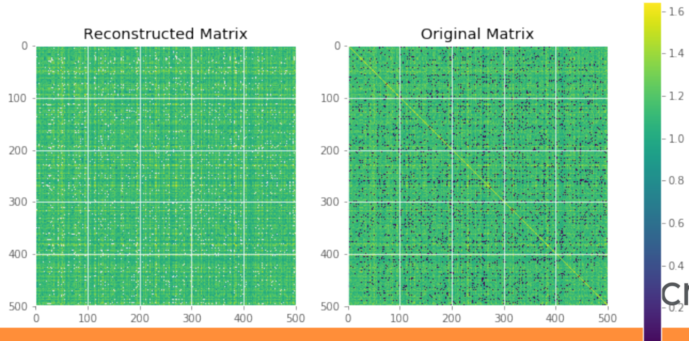
Answer

```
def observe(self, observation):
  for session in observation.sessions():
      self.user_embedding += self.embeddings[session['v'],:]
      self.history_length += 1

def act(self, observation, reward, done):
  """Act method returns an Action based on current observation and past history"""
  self.observe(observation)
  next_item_score = np.matmul(self.embeddings,self.user_embedding/self.history_length)

  action = np.argmax(next_item_score)
  prob = np.zeros_like(next_item_score)
  prob[action]=1.0
  return {
      **super().act(observation, reward, done),
      **{
          'a': action,
          'ps': 1.0,
          'ps-a': prob,
      },
  }
```

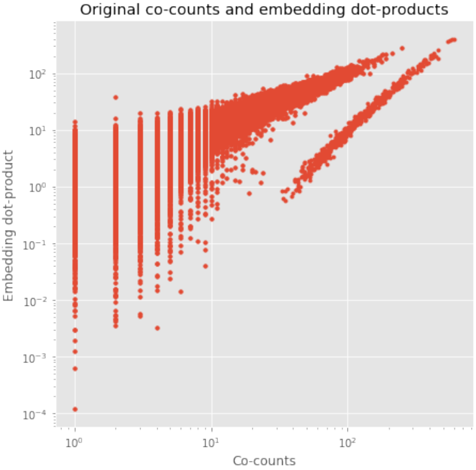
Evaluate an organic model using Bandit

Methods like word2vec and SVD can be used to produce a low rank factorization of a matrix of co counts. Imagine we have 500 products and we have observed the co counts on the right, the matrix on the left can produce a good reconstruction (perhaps better than the original as it can fill in missing signal). It is also uses less parameters, which has both computational and statistical advantages.

You could also use deep learning (see modules 1 and 2 of our course)

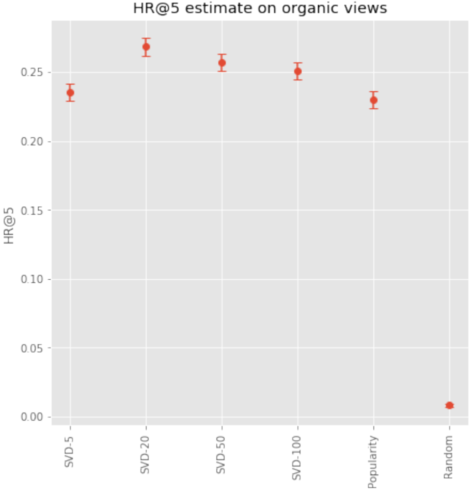


Evaluate an organic model using Bandit



Evaluate an organic model using Bandit

For such an organic method, a standard metric is hitrate at 5 HR@5 also called recall@5 or precision@5.



Evaluate an organic model using Bandit: IPS evaluation

Hit rate is a purely organic measure that is available offline and is seen in many recommender systems papers.

Evaluate an organic model using Bandit: IPS evaluation

Hit rate is a purely organic measure that is available offline and is seen in many recommender systems papers. ... but it is a purely organic metric, it doesn't predict performance at AB test time, at least not explicitly..

Evaluate an organic model using Bandit: IPS evaluation

Hit rate is a purely organic measure that is available offline and is seen in many recommender systems papers. ... but it is a purely organic metric, it doesn't predict performance at AB test time, at least not explicitly..

Can we change to an offline measure that predicts the result of an AB test?

Evaluate an organic model using Bandit: IPS evaluation

Hit rate is a purely organic measure that is available offline and is seen in many recommender systems papers. ... but it is a purely organic metric, it doesn't predict performance at AB test time, at least not explicitly..

Can we change to an offline measure that predicts the result of an AB test?

Imagine the past policy is random i.e. with some probability it will choose any action for a given user.

Evaluate an organic model using Bandit: IPS evaluation

Hit rate is a purely organic measure that is available offline and is seen in many recommender systems papers. ... but it is a purely organic metric, it doesn't predict performance at AB test time, at least not explicitly..

Can we change to an offline measure that predicts the result of an AB test?

Imagine the past policy is random i.e. with some probability it will choose any action for a given user. Random isn't the same as uniform and the policy typically will not be random.

Evaluate an organic model using Bandit: IPS evaluation

Hit rate is a purely organic measure that is available offline and is seen in many recommender systems papers. ... but it is a purely organic metric, it doesn't predict performance at AB test time, at least not explicitly..

Can we change to an offline measure that predicts the result of an AB test?

Imagine the past policy is random i.e. with some probability it will choose any action for a given user. Random isn't the same as uniform and the policy typically will not be random. *why?*

Evaluate an organic model using Bandit: IPS evaluation

Hit rate is a purely organic measure that is available offline and is seen in many recommender systems papers. ... but it is a purely organic metric, it doesn't predict performance at AB test time, at least not explicitly..

Can we change to an offline measure that predicts the result of an AB test?

Imagine the past policy is random i.e. with some probability it will choose any action for a given user. Random isn't the same as uniform and the policy typically will not be random. *why?* .. still a random logging policy is an interesting thought experiment or reference.

The central idea of an inverse propensity score is that we evaluate a new policy using the logs of an old policy.

Evaluate an organic model using Bandit: IPS evaluation

Hit rate is a purely organic measure that is available offline and is seen in many recommender systems papers. ... but it is a purely organic metric, it doesn't predict performance at AB test time, at least not explicitly..

Can we change to an offline measure that predicts the result of an AB test?

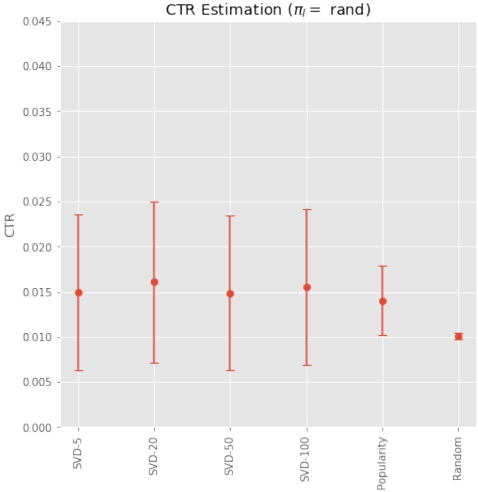
Imagine the past policy is random i.e. with some probability it will choose any action for a given user. Random isn't the same as uniform and the policy typically will not be random. *why?* .. still a random logging policy is an interesting thought experiment or reference.

The central idea of an inverse propensity score is that we evaluate a new policy using the logs of an old policy. We identify times in history where the new policy would make the same recommendation as the old policy, we then look at the contribution of the reward and

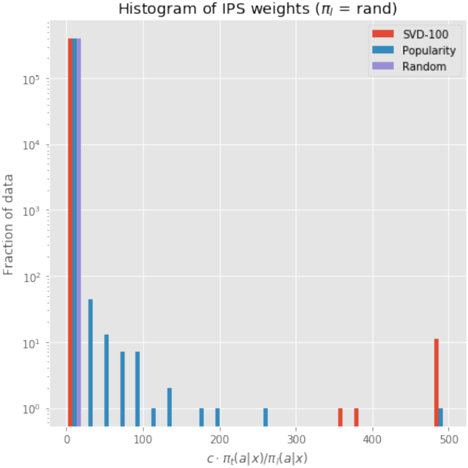
$$\mathrm{CTR}_{\pi_t}(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{(a,x,c)\in\mathcal{L}} c\,\frac{\pi_t(a|x)}{\pi_l(a|x)}$$

criteo

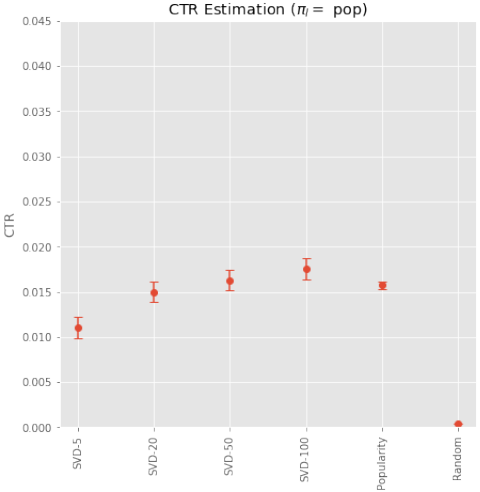
Evaluate an organic model using Bandit



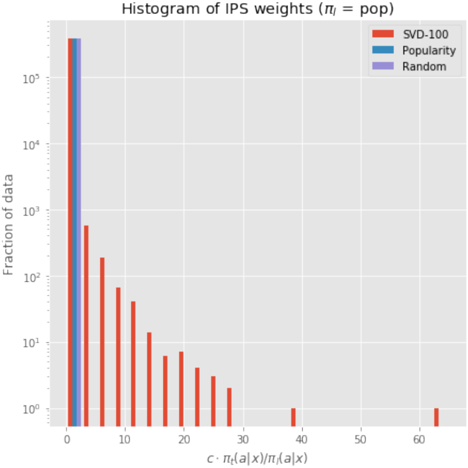
Evaluate an organic model using Bandit



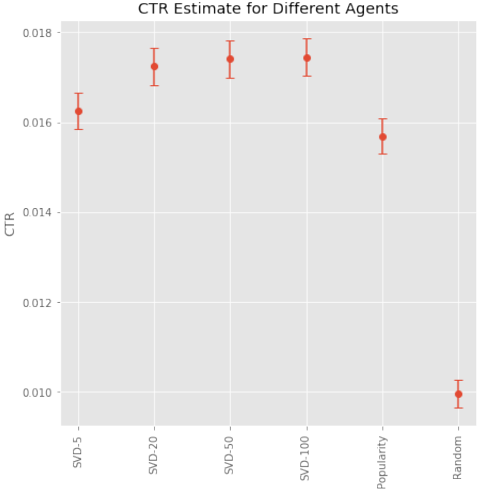
Evaluate an organic model using Bandit



Evaluate an organic model using Bandit



Evaluate an organic model using Bandit



LIkelihood Based Agents

Likelihood Based Agent

$$c_n \sim \text{Bernoulli}\left(\sigma\left(\Phi([\boldsymbol{X}_n\ \boldsymbol{a}_n])^T\beta\right)\right)$$

where

- β are the parameters;
- $\sigma(\cdot)$ is the logistic sigmoid;
- $\Phi(\cdot)$ is a function that maps $\boldsymbol{X}, \boldsymbol{a}$ to a higher dimensional space and includes some interaction terms between $\boldsymbol{X}_n$ and $\boldsymbol{a}_n$

Likelihood Based Agent

$$c_n \sim \text{Bernoulli}\left(\sigma\left(\boldsymbol{\Phi}([\boldsymbol{X}_n \ \boldsymbol{a}_n])^T \boldsymbol{\beta}\right)\right)$$

where

- $\boldsymbol{\beta}$ are the parameters;
- $\sigma(\cdot)$ is the logistic sigmoid;
- $\boldsymbol{\Phi}(\cdot)$ is a function that maps $\boldsymbol{X}, \boldsymbol{a}$ to a higher dimensional space and includes some interaction terms between $\boldsymbol{X}_n$ and $\boldsymbol{a}_n$ *Why?*

Likelihood Based Agent

Or using $\mathbf{\Phi}\left([\boldsymbol{X}_n \; \boldsymbol{a}_n]\right) = \boldsymbol{X}_n \otimes \boldsymbol{a}_n$:

$$\hat{\boldsymbol{\beta}}_{\mathrm{lh}} = \mathrm{argmax}_{\boldsymbol{\beta}} \sum_n c_n \log \sigma \left(\left(\boldsymbol{X}_n \otimes \boldsymbol{a}_n \right)^T \boldsymbol{\beta} \right) + (1 - c_n) \log \left(1 - \sigma \left(\left(\boldsymbol{X}_n \otimes \boldsymbol{a}_n \right)^T \boldsymbol{\beta} \right) \right)$$

Feature Engineering to get the context vector $\boldsymbol{X}$

The above model requires us to specify the context $\boldsymbol{X}$, how can we do this?

Feature Engineering

	a	c	ps	ps-a	t	u	v	z
0	NaN	NaN	NaN	None	0	0	0.0	organic
1	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	1	0	NaN	bandit
2	4.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	2	0	NaN	bandit
3	5.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	3	0	NaN	bandit
4	NaN	NaN	NaN	None	0	1	1.0	organic
5	2.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	1	1	NaN	bandit
6	8.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	2	1	NaN	bandit
7	4.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	3	1	NaN	bandit
8	NaN	NaN	NaN	None	4	1	6.0	organic
9	NaN	NaN	NaN	None	5	1	6.0	organic
10	NaN	NaN	NaN	None	6	1	4.0	organic
11	NaN	NaN	NaN	None	7	1	6.0	organic
12	NaN	NaN	NaN	None	8	1	2.0	organic
13	NaN	NaN	NaN	None	9	1	6.0	organic
14	NaN	NaN	NaN	None	10	1	2.0	organic
15	NaN	NaN	NaN	None	11	1	1.0	organic
16	NaN	NaN	NaN	None	12	1	6.0	organic
17	NaN	NaN	NaN	None	13	1	6.0	organic
18	NaN	NaN	NaN	None	14	1	4.0	organic
19	NaN	NaN	NaN	None	15	1	1.0	organic
20	NaN	NaN	NaN	None	16	1	1.0	organic
21	NaN	NaN	NaN	None	17	1	1.0	organic
22	NaN	NaN	NaN	None	18	1	6.0	organic
23	NaN	NaN	NaN	None	19	1	6.0	organic
24	NaN	NaN	NaN	None	20	1	1.0	organic
25	NaN	NaN	NaN	None	21	1	6.0	organic
26	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	22	1	NaN	bandit

```
history[0:8]
```

```
[array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 6., 2., 0., 2., 0., 9., 0., 0., 0.]),
 array([0., 6., 2., 0., 2., 0., 9., 0., 0., 0.])]
```

```
actions[0:8]
```

```
[array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0], dtype=int8),
 array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int8)]
```

Feature Engineering

	a	c	ps	ps-a	t	u	v	z
0	NaN	NaN	NaN	None	0	0	0.0	organic
1	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	1	0	NaN	bandit
2	4.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	2	0	NaN	bandit
3	5.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	3	0	NaN	bandit
4	NaN	NaN	NaN	None	0	1	1.0	organic
5	2.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	1	1	NaN	bandit
6	8.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	2	1	NaN	bandit
7	4.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	3	1	NaN	bandit
8	NaN	NaN	NaN	None	4	1	6.0	organic
9	NaN	NaN	NaN	None	5	1	6.0	organic
10	NaN	NaN	NaN	None	6	1	4.0	organic
11	NaN	NaN	NaN	None	7	1	6.0	organic
12	NaN	NaN	NaN	None	8	1	2.0	organic
13	NaN	NaN	NaN	None	9	1	6.0	organic
14	NaN	NaN	NaN	None	10	1	2.0	organic
15	NaN	NaN	NaN	None	11	1	1.0	organic
16	NaN	NaN	NaN	None	12	1	6.0	organic
17	NaN	NaN	NaN	None	13	1	6.0	organic
18	NaN	NaN	NaN	None	14	1	4.0	organic
19	NaN	NaN	NaN	None	15	1	1.0	organic
20	NaN	NaN	NaN	None	16	1	1.0	organic
21	NaN	NaN	NaN	None	17	1	1.0	organic
22	NaN	NaN	NaN	None	18	1	6.0	organic
23	NaN	NaN	NaN	None	19	1	6.0	organic
24	NaN	NaN	NaN	None	20	1	1.0	organic
25	NaN	NaN	NaN	None	21	1	6.0	organic
26	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	22	1	NaN	bandit

```
history[0:8]
```

```
[array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 6., 2., 0., 2., 0., 9., 0., 0., 0.]),
 array([0., 6., 2., 0., 2., 0., 9., 0., 0., 0.])]
```

```
actions[0:8]
```

```
[array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0], dtype=int8),
 array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int8)]
```

How would you interact the features?

criteo

Feature Engineering

	a	c	ps	ps-a	t	u	v	z
0	NaN	NaN	NaN	None	0	0	0.0	organic
1	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	1	0	NaN	bandit
2	4.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	2	0	NaN	bandit
3	5.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	3	0	NaN	bandit
4	NaN	NaN	NaN	None	0	1	1.0	organic
5	2.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	1	1	NaN	bandit
6	8.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	2	1	NaN	bandit
7	4.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	3	1	NaN	bandit
8	NaN	NaN	NaN	None	4	1	6.0	organic
9	NaN	NaN	NaN	None	5	1	6.0	organic
10	NaN	NaN	NaN	None	6	1	4.0	organic
11	NaN	NaN	NaN	None	7	1	6.0	organic
12	NaN	NaN	NaN	None	8	1	2.0	organic
13	NaN	NaN	NaN	None	9	1	6.0	organic
14	NaN	NaN	NaN	None	10	1	2.0	organic
15	NaN	NaN	NaN	None	11	1	1.0	organic
16	NaN	NaN	NaN	None	12	1	6.0	organic
17	NaN	NaN	NaN	None	13	1	6.0	organic
18	NaN	NaN	NaN	None	14	1	4.0	organic
19	NaN	NaN	NaN	None	15	1	1.0	organic
20	NaN	NaN	NaN	None	16	1	1.0	organic
21	NaN	NaN	NaN	None	17	1	1.0	organic
22	NaN	NaN	NaN	None	18	1	6.0	organic
23	NaN	NaN	NaN	None	19	1	6.0	organic
24	NaN	NaN	NaN	None	20	1	1.0	organic
25	NaN	NaN	NaN	None	21	1	6.0	organic
26	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	22	1	NaN	bandit

```
history[0:8]
```

```
[array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 6., 2., 0., 2., 0., 9., 0., 0., 0.]),
 array([0., 6., 2., 0., 2., 0., 9., 0., 0., 0.])]
```

```
actions[0:8]
```

```
[array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0], dtype=int8),
 array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int8)]
```

How would you interact the features?

Is this a good feature engineering scheme? Can you think of a better one?

Likelihood Bandit Agent

	a	c	ps	ps-a	t	u	v	z
0	NaN	NaN	NaN	None	0	0	0.0	organic
1	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	1	0	NaN	bandit
2	4.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	2	0	NaN	bandit
3	5.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	3	0	NaN	bandit
4	NaN	NaN	NaN	None	0	1	1.0	organic
5	2.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	1	1	NaN	bandit
6	8.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	2	1	NaN	bandit
7	4.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	3	1	NaN	bandit
8	NaN	NaN	NaN	None	4	1	6.0	organic
9	NaN	NaN	NaN	None	5	1	6.0	organic
10	NaN	NaN	NaN	None	6	1	4.0	organic
11	NaN	NaN	NaN	None	7	1	6.0	organic
12	NaN	NaN	NaN	None	8	1	2.0	organic
13	NaN	NaN	NaN	None	9	1	6.0	organic
14	NaN	NaN	NaN	None	10	1	2.0	organic
15	NaN	NaN	NaN	None	11	1	1.0	organic
16	NaN	NaN	NaN	None	12	1	6.0	organic
17	NaN	NaN	NaN	None	13	1	6.0	organic
18	NaN	NaN	NaN	None	14	1	4.0	organic
19	NaN	NaN	NaN	None	15	1	1.0	organic
20	NaN	NaN	NaN	None	16	1	1.0	organic
21	NaN	NaN	NaN	None	17	1	1.0	organic
22	NaN	NaN	NaN	None	18	1	6.0	organic
23	NaN	NaN	NaN	None	19	1	6.0	organic
24	NaN	NaN	NaN	None	20	1	1.0	organic
25	NaN	NaN	NaN	None	21	1	6.0	organic
26	3.0	0.0	0.1	[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, ...	22	1	NaN	bandit

```
history[0:8]
```

```
[array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 6., 2., 0., 2., 0., 9., 0., 0., 0.]),
 array([0., 6., 2., 0., 2., 0., 9., 0., 0., 0.])]
```

```
actions[0:8]
```

```
[array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0], dtype=int8),
 array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0], dtype=int8),
 array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0], dtype=int8),
 array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int8)]
```

criteo

Other ways to train on bandit feedback - reweighted likelihood

What if we fit an overly simplified parametric model to the data?

Other ways to train on bandit feedback - reweighted likelihood

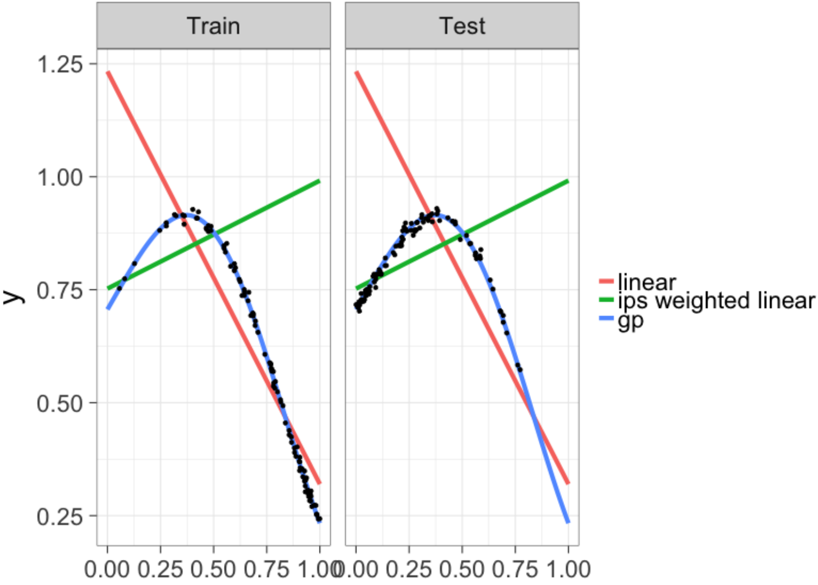
What if we fit an overly simplified parametric model to the data?

The simple model will predict accurately for the most frequent actions and compromise the fit for the least frequent actions.

$$\hat{\beta}_{\text{re-weight}} = \operatorname{argmax}_{\boldsymbol{\beta}} \sum_n w_n c_n \log \sigma \left(\boldsymbol{\Phi} \left([\boldsymbol{X}_n \;\; \boldsymbol{a}_n] \right)^T \beta \right) \\ + w_n (1 - c_n) \log \left(1 - \sigma \left(\boldsymbol{\Phi} \left([\boldsymbol{X}_n \;\; \boldsymbol{a}_n] \right)^T \beta \right) \right)$$

where the weight is defined: $w_n = \frac{1}{\pi(\boldsymbol{a}_n | \boldsymbol{X}_n)}$.

Pure Organic vs Pure Bandit



Other ways to train on bandit feedback - counterfactual risk

Often in machine learning we search for a decision rule that would have worked well in the past.

Other ways to train on bandit feedback - counterfactual risk

Often in machine learning we search for a decision rule that would have worked well in the past. We call this empirical risk.

Other ways to train on bandit feedback - counterfactual risk

Often in machine learning we search for a decision rule that would have worked well in the past. We call this empirical risk.

We can't evaluate decisions we didn't do in the past.

Other ways to train on bandit feedback - counterfactual risk

Often in machine learning we search for a decision rule that would have worked well in the past. We call this empirical risk.

We can't evaluate decisions we didn't do in the past. We can however use IPS to estimate them.

Other ways to train on bandit feedback - counterfactual risk

Often in machine learning we search for a decision rule that would have worked well in the past. We call this empirical risk.

We can't evaluate decisions we didn't do in the past. We can however use IPS to estimate them.

A direct estimation of this leads to the contextual bandit formulation:

Other ways to train on bandit feedback - counterfactual risk

Often in machine learning we search for a decision rule that would have worked well in the past. We call this empirical risk.

We can't evaluate decisions we didn't do in the past. We can however use IPS to estimate them.

A direct estimation of this leads to the contextual bandit formulation:

$$\pi_{\boldsymbol{\beta}}\left(\boldsymbol{a}_n|\boldsymbol{X}_n\right) = \mathrm{softmax}\left(\Phi\left([\boldsymbol{X}_n \ \boldsymbol{a}_n]\right)^T \beta\right)$$

criteo

Other ways to train on bandit feedback - counterfactual risk

$$\hat{\beta}_{\mathrm{CB}} = \mathrm{argmax}_{\boldsymbol{\beta}} \sum_{n} w_n c_n (\boldsymbol{X}_n \otimes \boldsymbol{a}_n)^T \ \boldsymbol{\beta} - w_n c_n \log \sum_{\boldsymbol{a}_n'} e^{\left(\boldsymbol{X}_n \otimes \boldsymbol{a}_n'\right)^T \boldsymbol{\beta}}$$

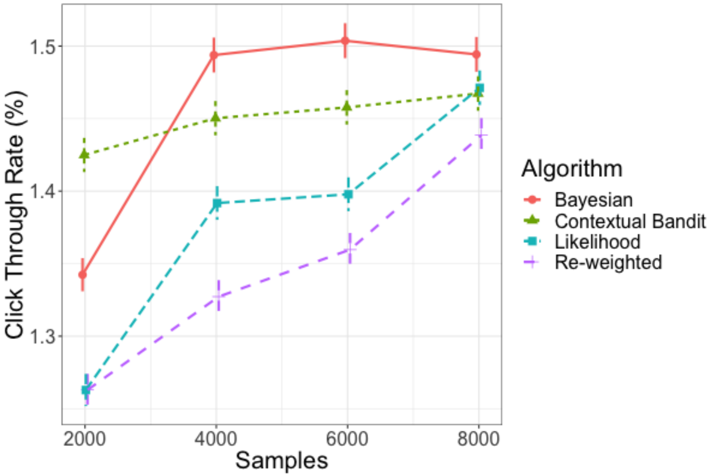
criteo

Other ways to train on bandit feedback - counterfactual risk

$$\hat{\beta}_{\mathrm{CB}} = \operatorname{argmax}_{\boldsymbol{\beta}} \sum_n w_n c_n (\boldsymbol{X}_n \otimes \boldsymbol{a}_n)^T \; \boldsymbol{\beta}$$
$$- w_n c_n \log \sum_{\boldsymbol{a}_n'} e^{\left(\boldsymbol{X}_n \otimes \boldsymbol{a}_n'\right)^T \boldsymbol{\beta}}$$

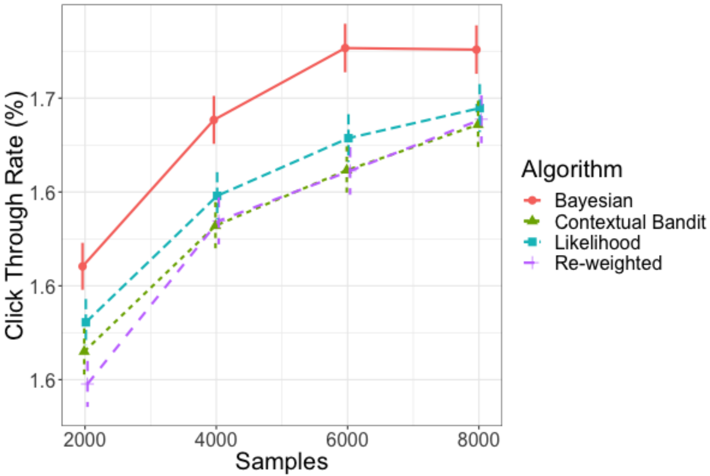
criteo

Three methods



criteo

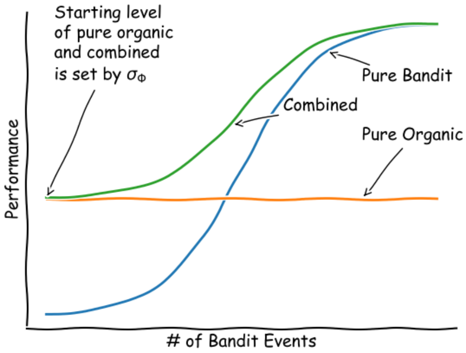
Three methods



Much more in the next moudule. For a short summary see our paper: "Three methods for training on Bandit feedback"

Combining Organic Signal with Bandit Signal

Pure Organic vs Pure Bandit



criteo

Can we get the best of both worlds?

A simple algorithm:

Can we get the best of both worlds?

A simple algorithm:

Use one of the methods from the first part of the course on the organic data to prouduce product emebeddings. Classic methods are SVD or word2vec.

Can we get the best of both worlds?

A simple algorithm:

Use one of the methods from the first part of the course on the organic data to prouduce product emebeddings. Classic methods are SVD or word2vec.

A user's history is then a list of embeddings rather than non-descript identifiers

Can we get the best of both worlds?

A simple algorithm:

Use one of the methods from the first part of the course on the organic data to prouduce product emebeddings. Classic methods are SVD or word2vec.

A user's history is then a list of embeddings rather than non-descript identifiers

The action space is also in the same embedding space.

Can we get the best of both worlds?

A simple algorithm:

Use one of the methods from the first part of the course on the organic data to prouduce product emebeddings. Classic methods are SVD or word2vec.

A user's history is then a list of embeddings rather than non-descript identifiers

The action space is also in the same embedding space.

Exercise: implement this approach. See "INSERT NAME"

Can we get the best of both worlds?

A simple algorithm:

Use one of the methods from the first part of the course on the organic data to prouduce product emebeddings. Classic methods are SVD or word2vec.

A user's history is then a list of embeddings rather than non-descript identifiers

The action space is also in the same embedding space.

Exercise: implement this approach. See "INSERT NAME"

What advantages or challenges do you have in this approach?

Thank You!