# Part 3. Bandit Feedback and Likelihood Models for Recommendation

David Rohde, Flavian Vasile

Criteo Research

*June 19, 2019*

criteo

# Collaborators:

Special thanks to:

- Olivier Jeunen
- Dmytro Mykhaylov
- Stephen Bonner

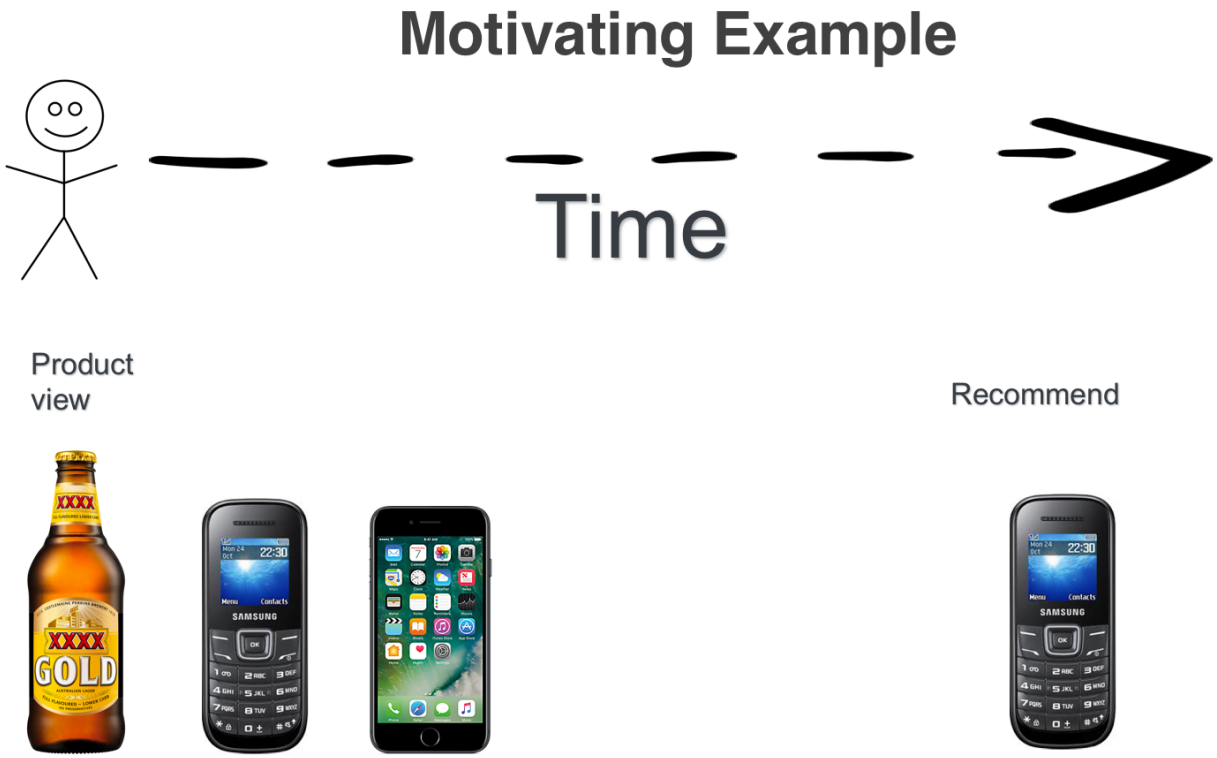
criteo

# Structure of the talk

- Recommendation Revisited: Acadmic vs Industry Reco

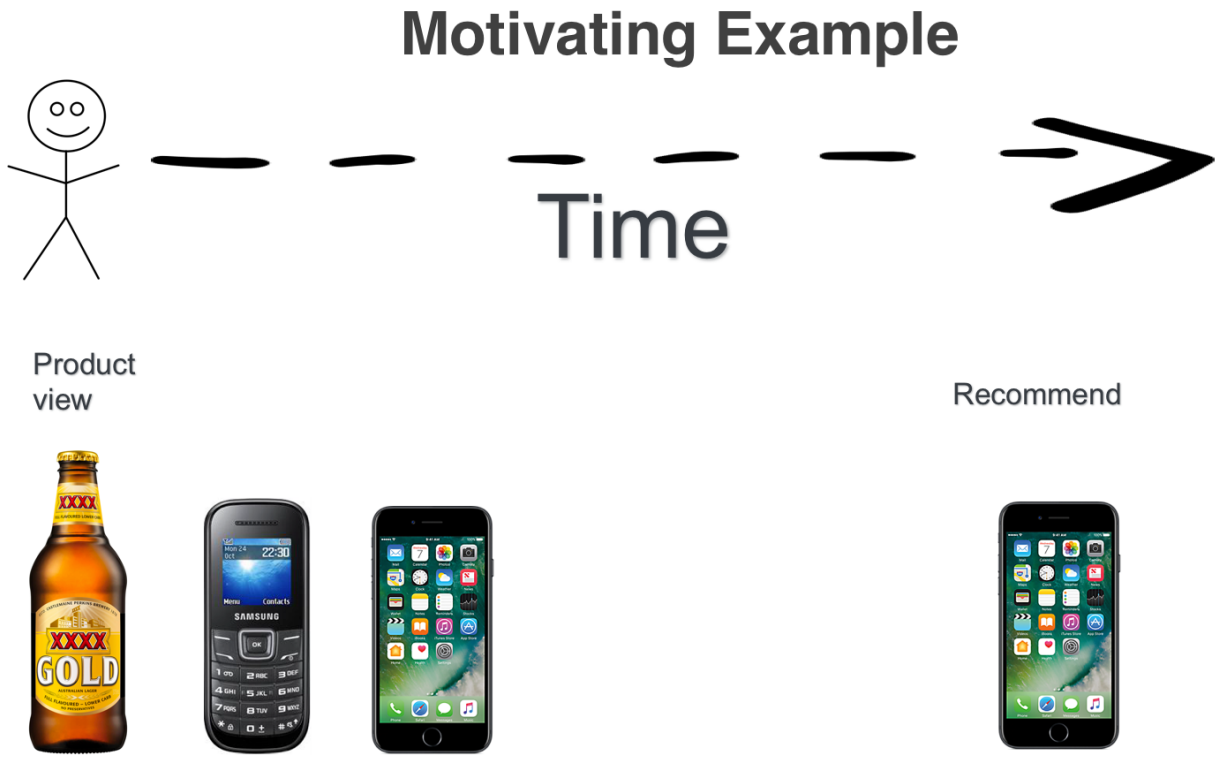
criteo

# The Reco Problem Revisited



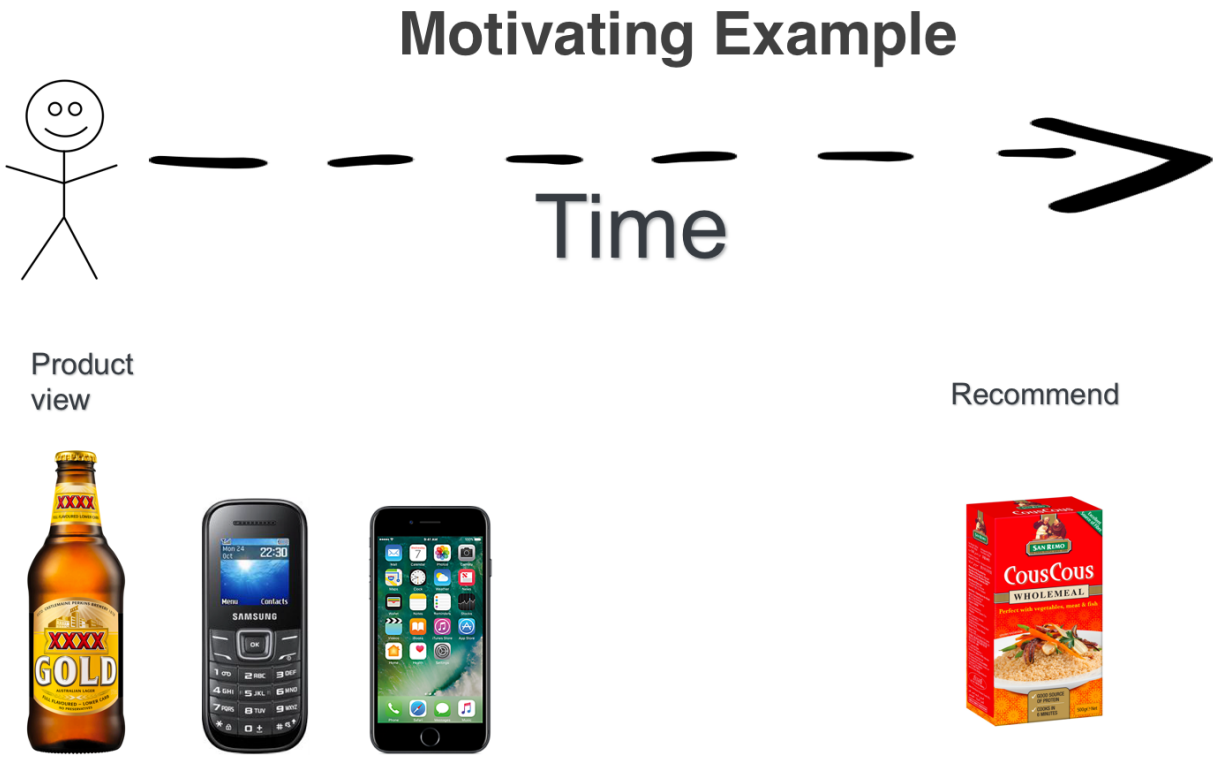

criteo

# The Reco Problem Revisited



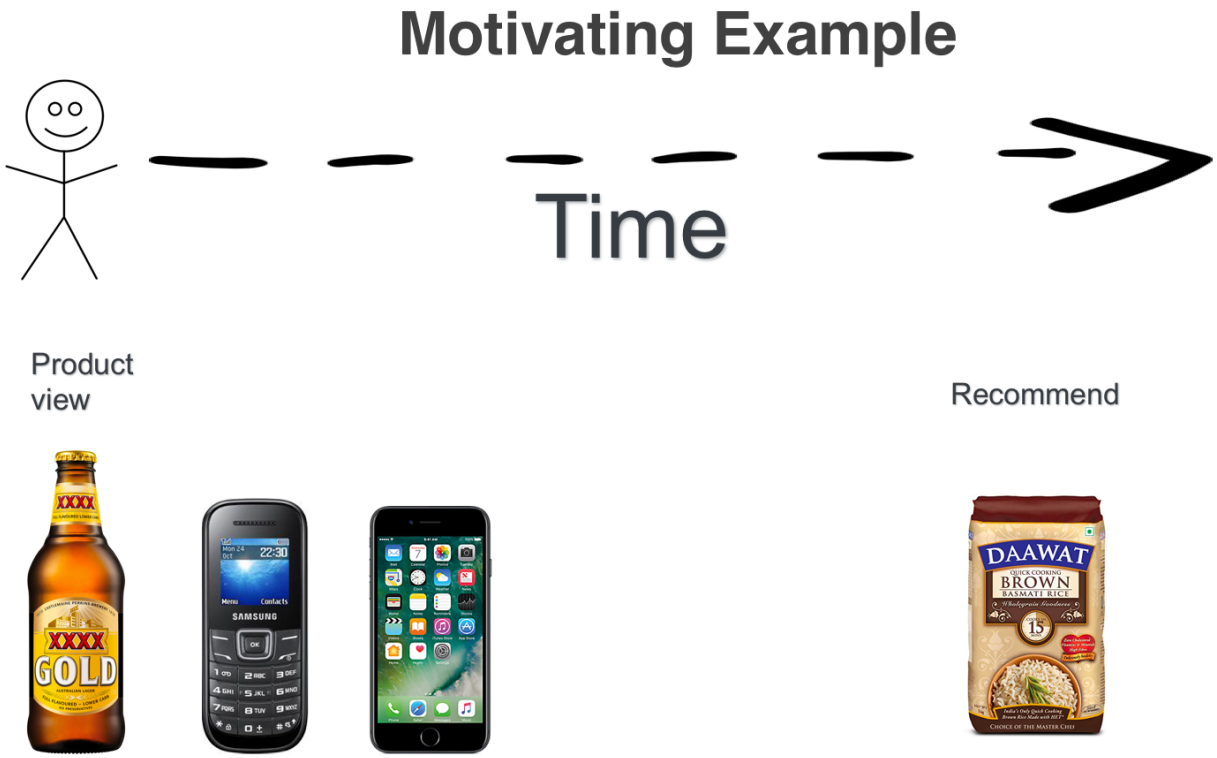

criteo

# The Reco Problem Revisited



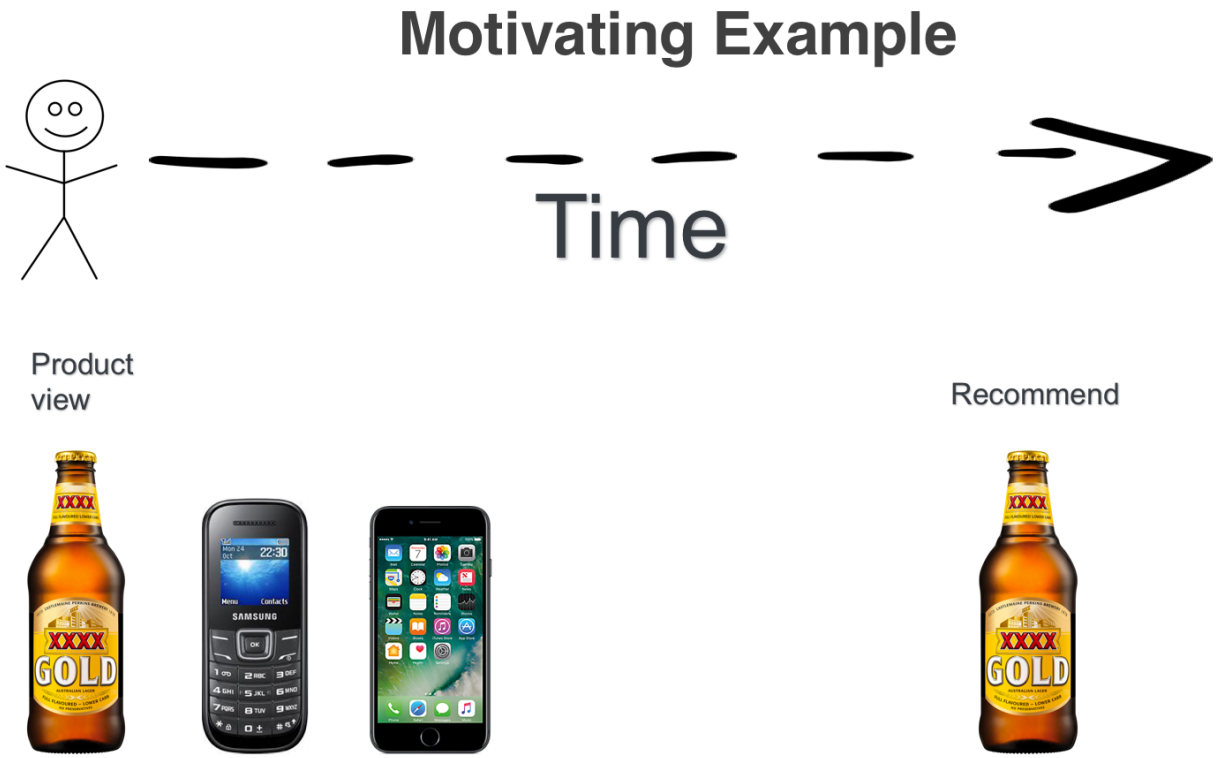

criteo

# The Reco Problem Revisited




criteo

# The Reco Problem Revisited




criteo

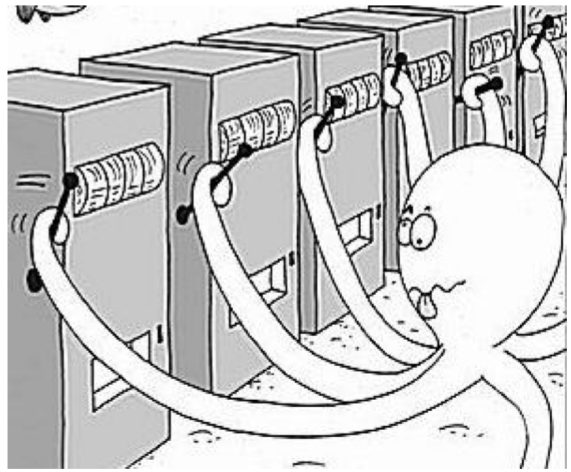
# History: Split History of Recommender Systems

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.





criteo

# 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

criteo

# 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

criteo

# 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

criteo

# 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

# 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!

criteo

# 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.

criteo

# 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

criteo

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

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

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

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

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

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

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)

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

# 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

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

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

criteo

# Supervised learning with the wrong objective

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

- Missing link prediction
- Next user event prediction

criteo

# 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**

criteo

# 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.

criteo

# 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

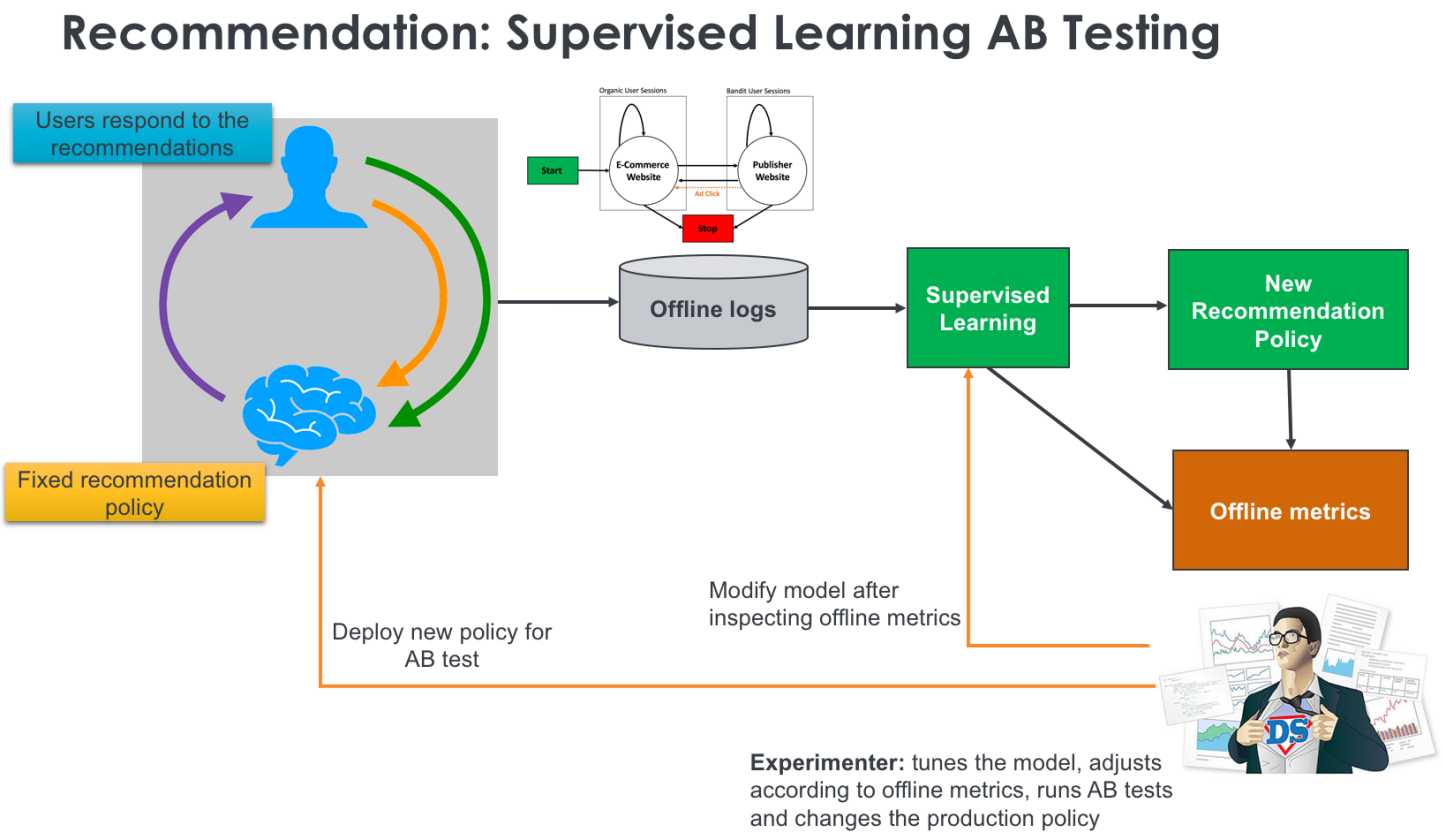
- Of course, we could start incorporating user feedback that we collected while running the initial recommendation policies
- We should be able to continue bringing improvements using feedback that is now aligned with our business metrics (ad ctr, post click sales, dwell time, number of videos watched,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

# Aligning the Recommendation objective with the business

However, this does not come without caveats...

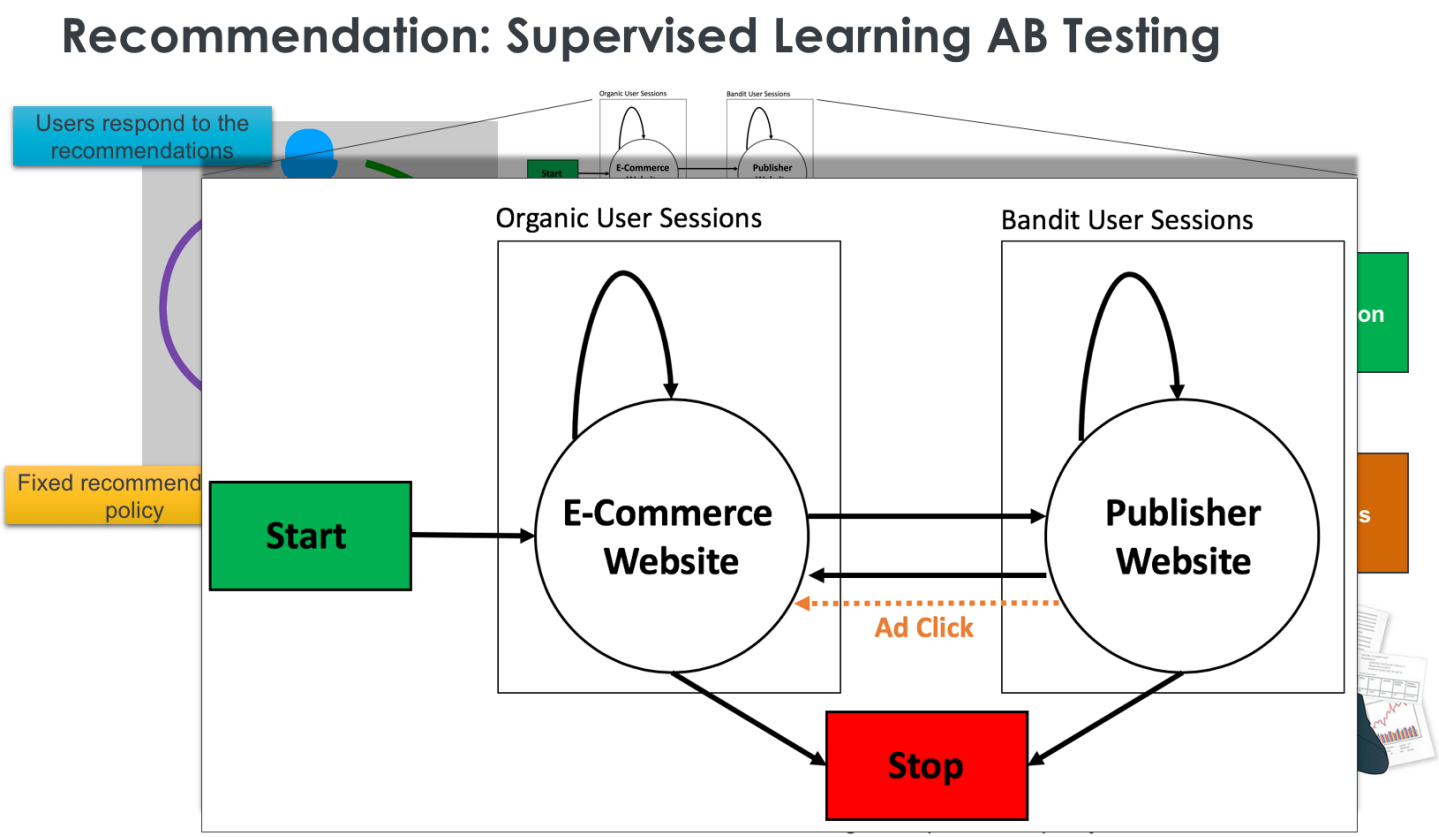
criteo

# Recommendation as Supervised Learning and AB Testing




criteo

# 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/`

criteo

# The relationship between Reinforcement Learning and Recommendation

- We are doing Reinforcement Learning by hand!

criteo

# 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

criteo

# 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

criteo

# 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?

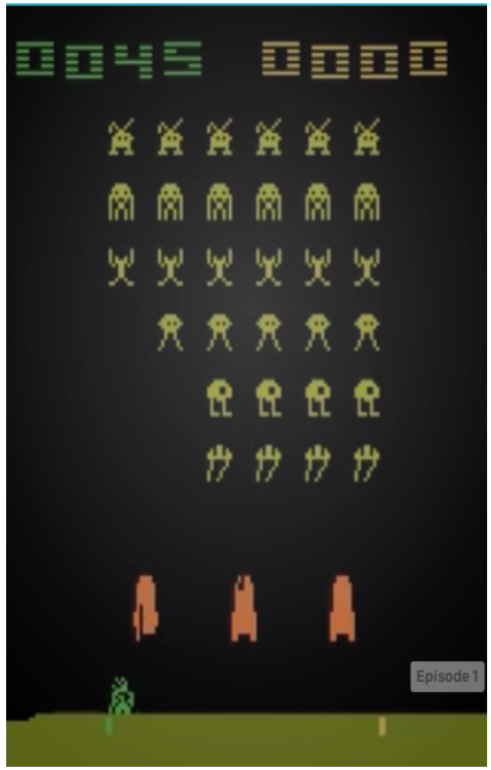
criteo

# 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.

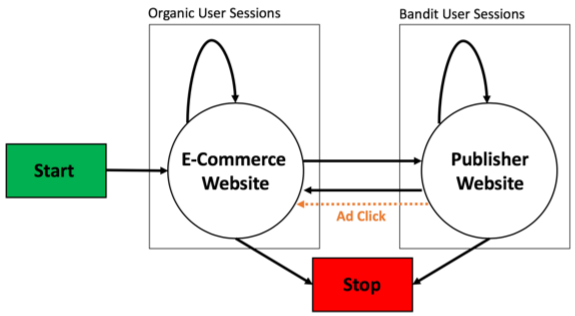





criteo

# 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, $\mu_\Gamma$ organic popularity $\boldsymbol{\beta} \in \mathcal{R}^{P\times K}$ bandit embeddings, $\mu_\beta$ 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
  $\omega_{u,0} \sim \mathcal{N}(0_{K\times1}, I_K)$
  $v_{u,0} \sim$ Categorical(softmax($\Gamma\omega_{u,0}$))
  **while** $z_{u,t} \neq$ *stop* **do**
    $t \leftarrow t+1$
    $\omega_{u,t} \sim \mathcal{N}(\omega_{u,t-1}, \sigma_\omega^2 I_K)$
    **if** $c_{u,t-1} \neq 1$ **then**
      $z_{u,t} \sim$ 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**
      $v_{u,t} \sim$ Categorical(softmax($\Gamma\omega_{u,t} + \mu_\Gamma$))
      $r_{u,t} \leftarrow$ undef
      $c_{u,t} \leftarrow$ undef
    **end**
    **if** $z_{u,t} =$ *bandit* **then**
      $r_{u,t}$ is geneated from the policy
      $c_{u,t} \sim$ Bernoulli($[f(\boldsymbol{\beta}\omega_{u,t} + \mu_\beta)]_{r_{u,t}}$)
    **end**
  **end**
**end**



criteo

$f$ is an increasing function

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

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

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

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

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

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

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

time



criteo

# 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]: []
```

criteo

# 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]: []
```

criteo

# 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}]
```

criteo

# 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 |

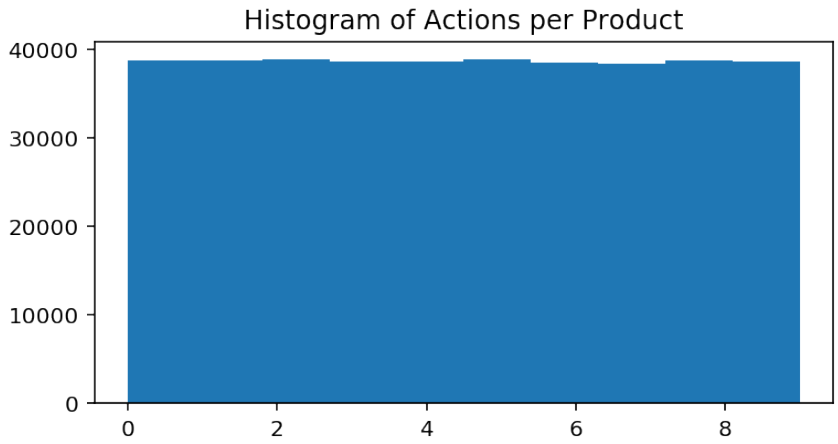
criteo

# Organic Best Of vs Bandit Best Of

criteo

# Organic Best Of vs Bandit Best Of

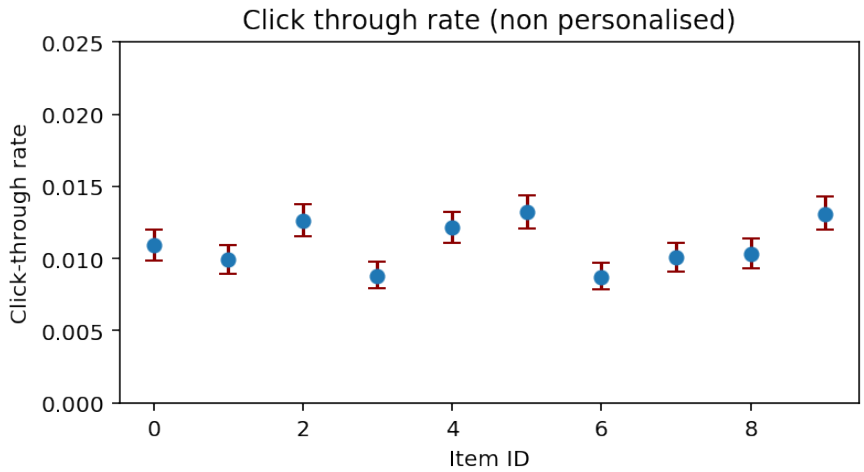
We can exampine the click through rate of each action under this policy, we see that action 5, 2 and 9 have high click through rates.




criteo

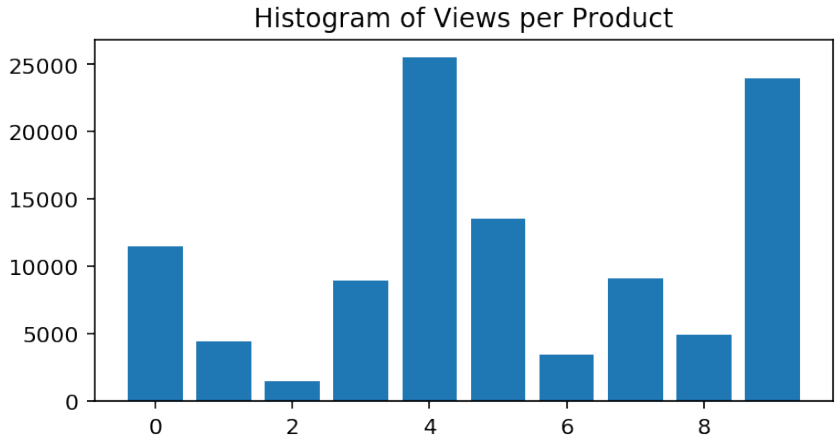
# Organic Best Of vs Bandit Best Of

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.



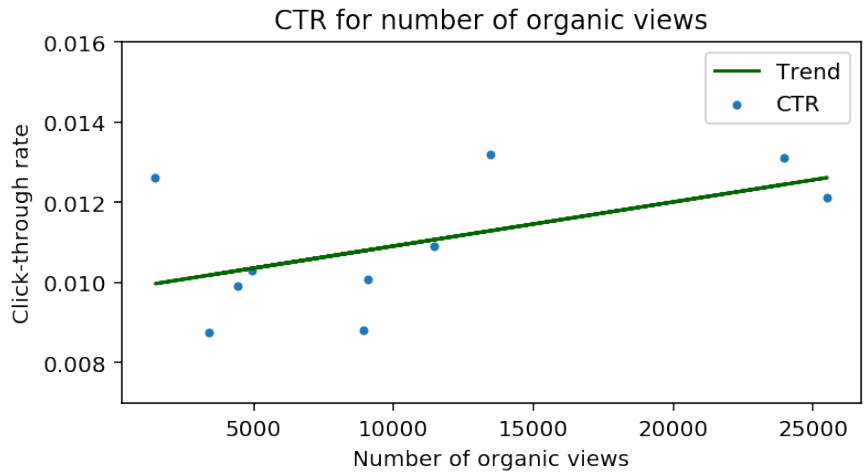
# Organic Best Of vs Bandit Best Of

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.

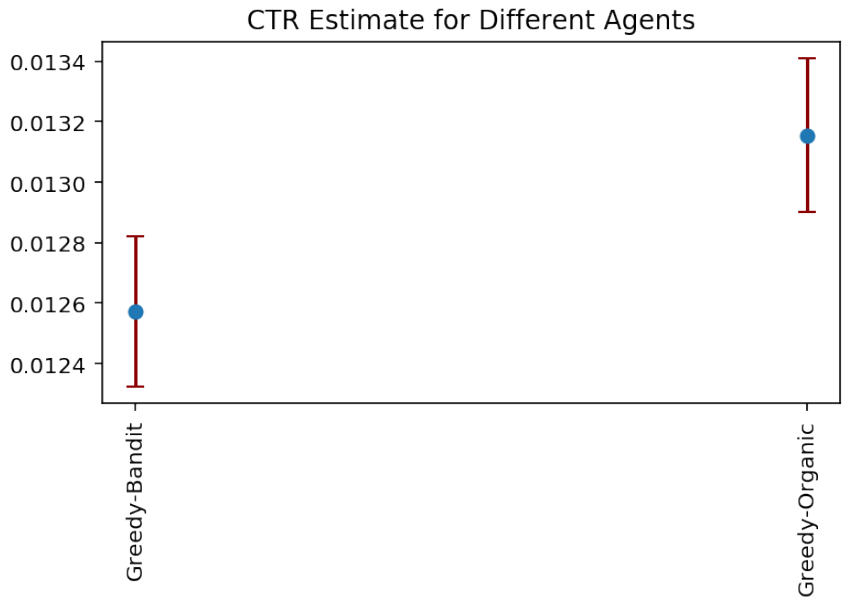



criteo

# Organic Best Of vs Bandit Best Of

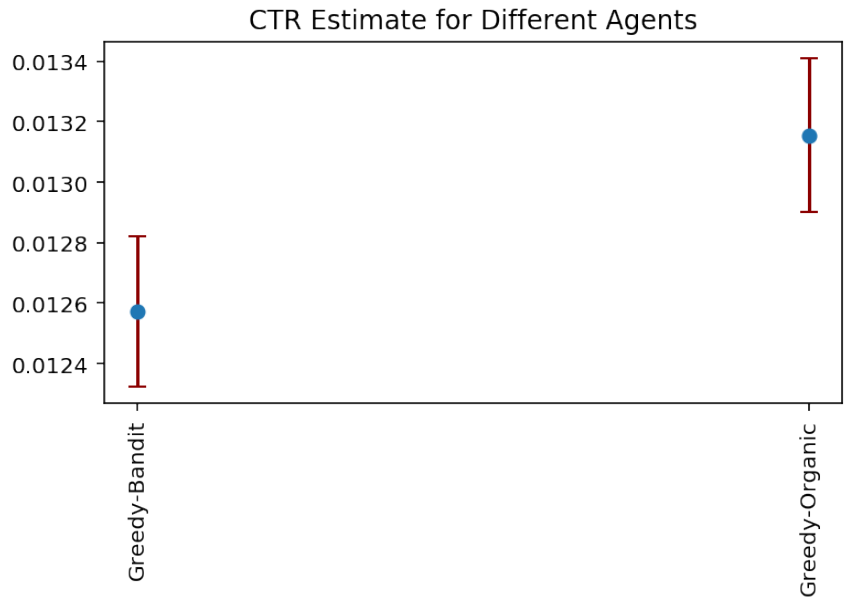
A bandit best of agent always recommends the highst ctr product (product 5). An organic best of always recommends the most frequently viewed product (product 4). A simmulated AB test shows using the bandit best of yields a better result.

# Organic Best Of vs Bandit Best Of




criteo

# Organic Best Of vs Bandit Best Of




Personalisation of course will result in a better result still.

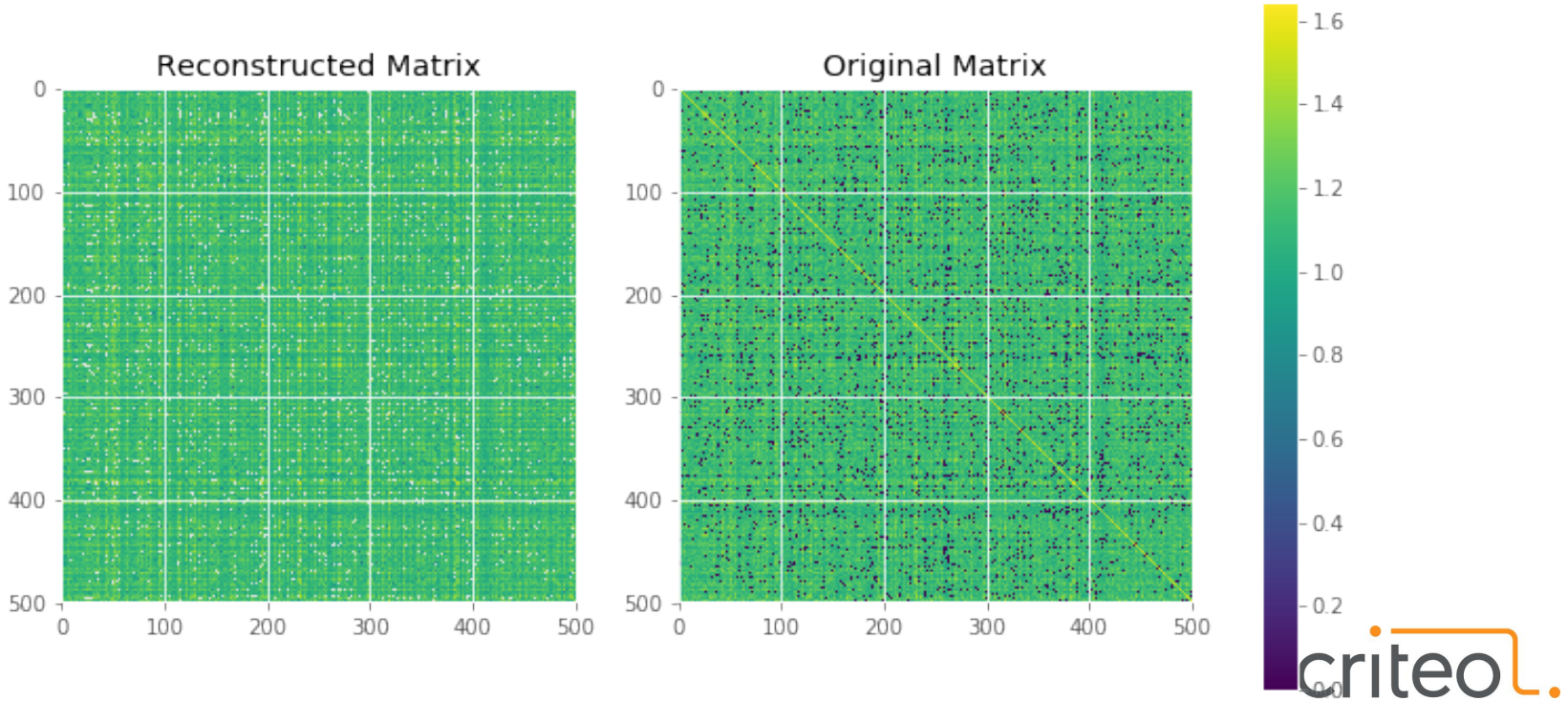
criteo

# Evaluate an organic agent using the Bandit Signal

criteo

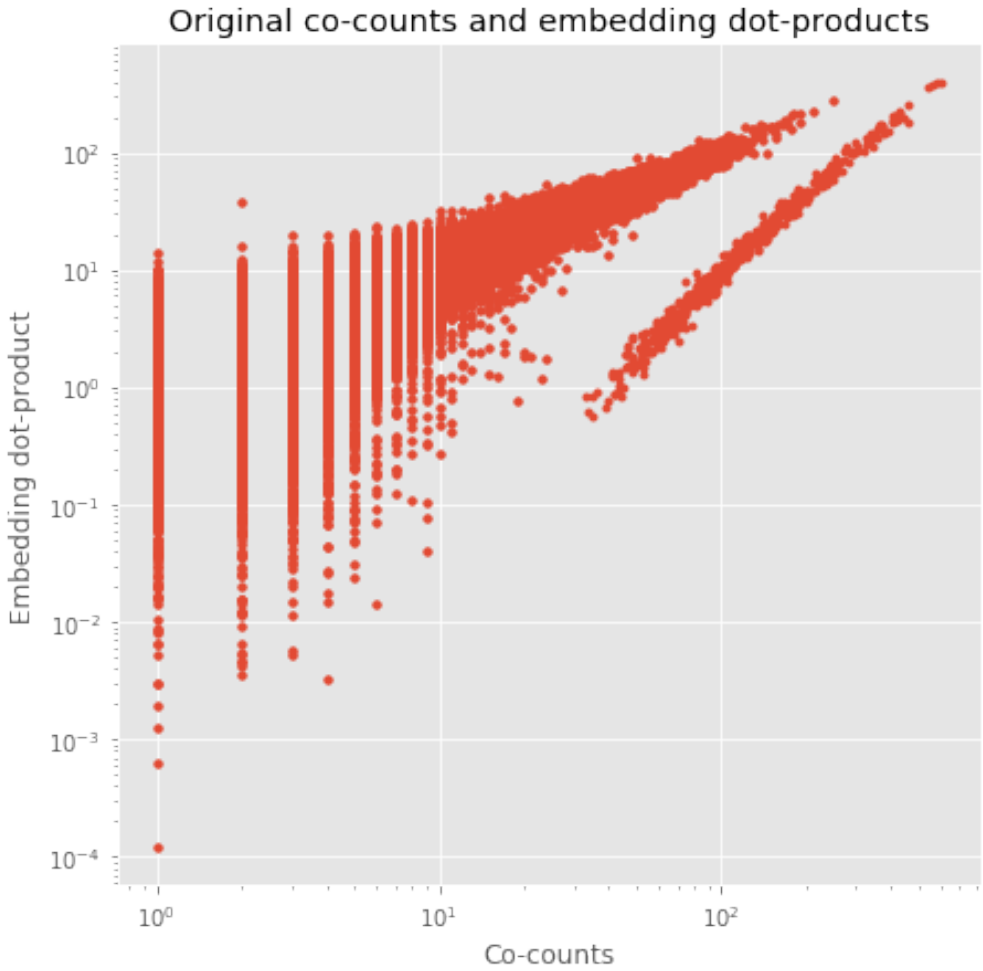
# Evaluate an organic model using Bandit

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.




criteo

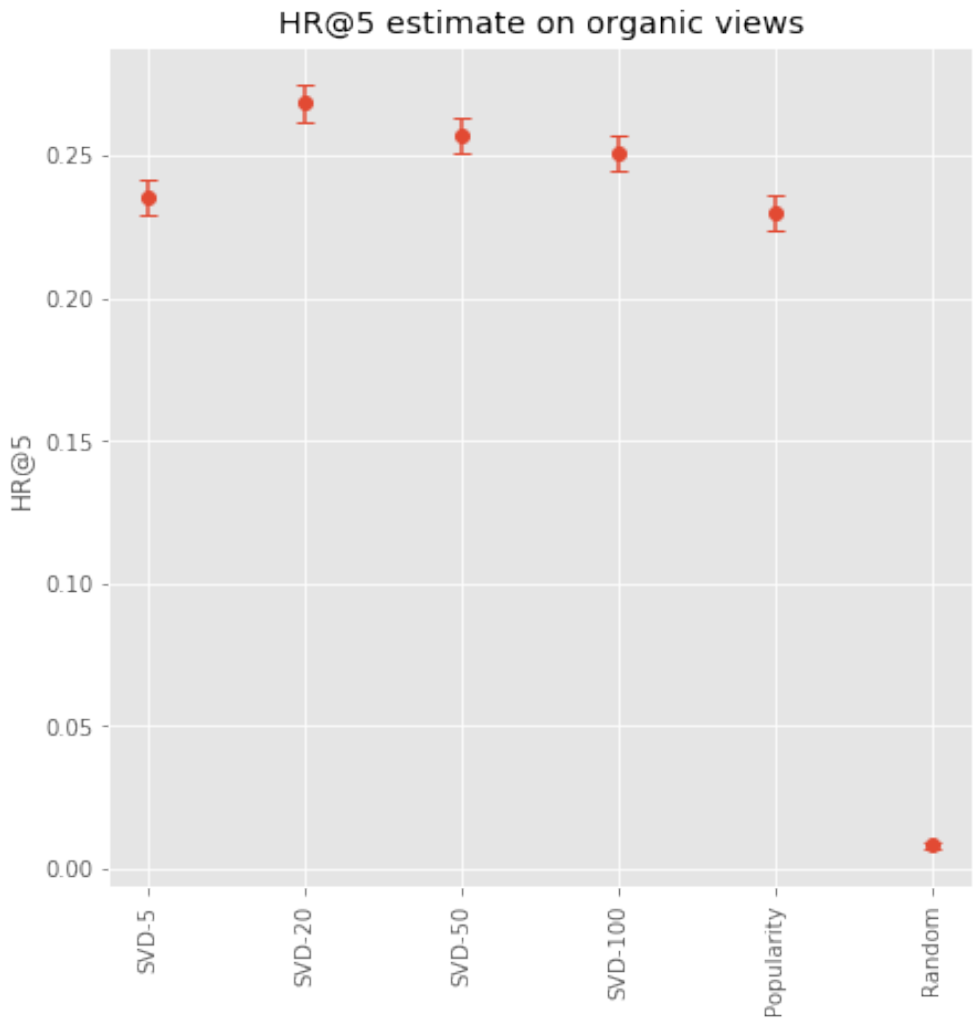
# Evaluate an organic model using Bandit




criteo

# 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.




criteo

# 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.

criteo

# 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..

criteo

# 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?

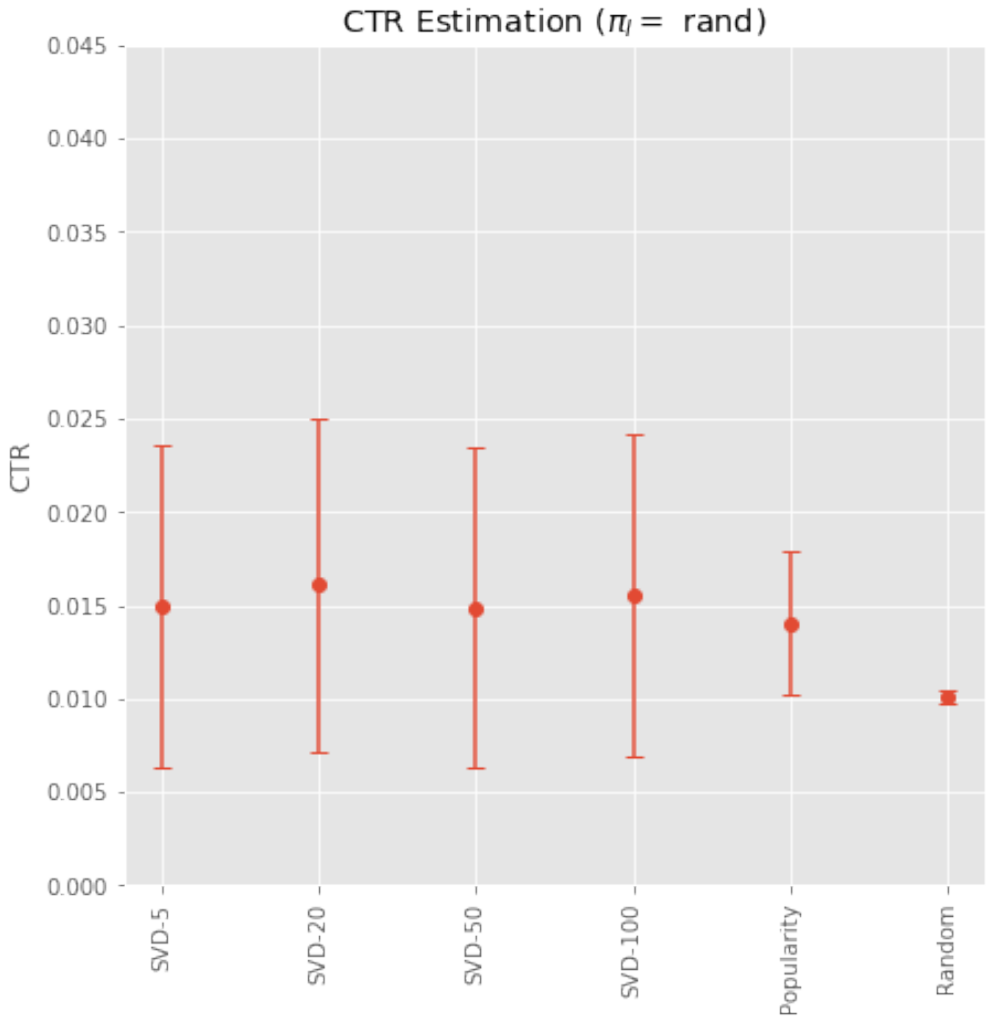
Imagine the past policy is random i.e. with some probability it will choose any action for a given user. Random isn't the same as uniform and the policy typically will not be random. *why?* .. still a random logging policy is an interesting thought experiment or reference.

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

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

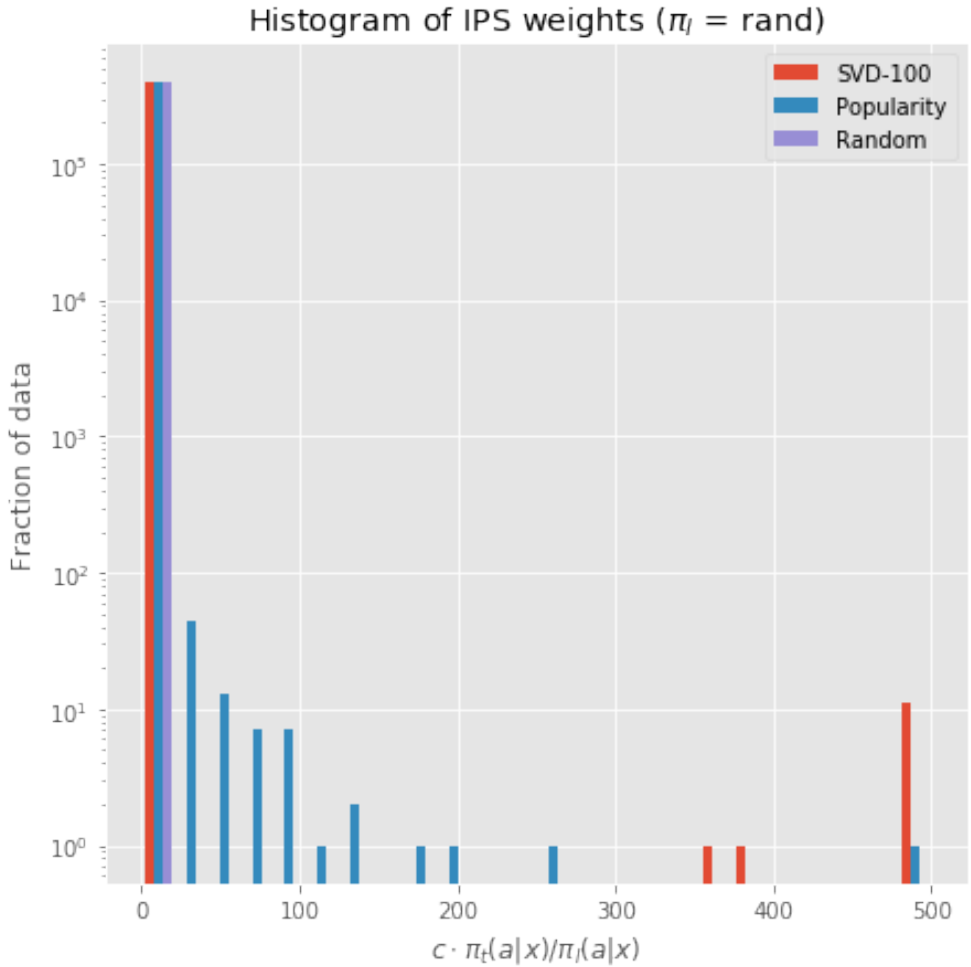
criteo

# Evaluate an organic model using Bandit



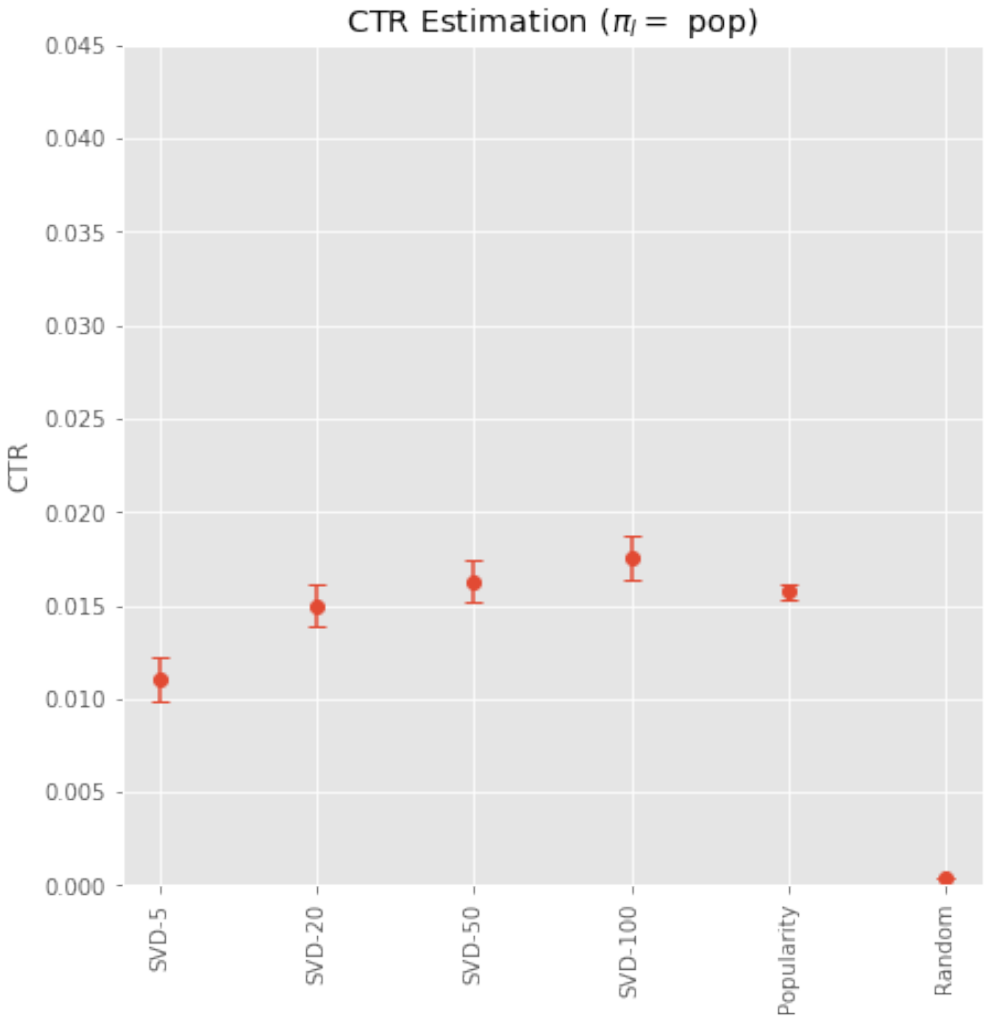

criteo

# Evaluate an organic model using Bandit



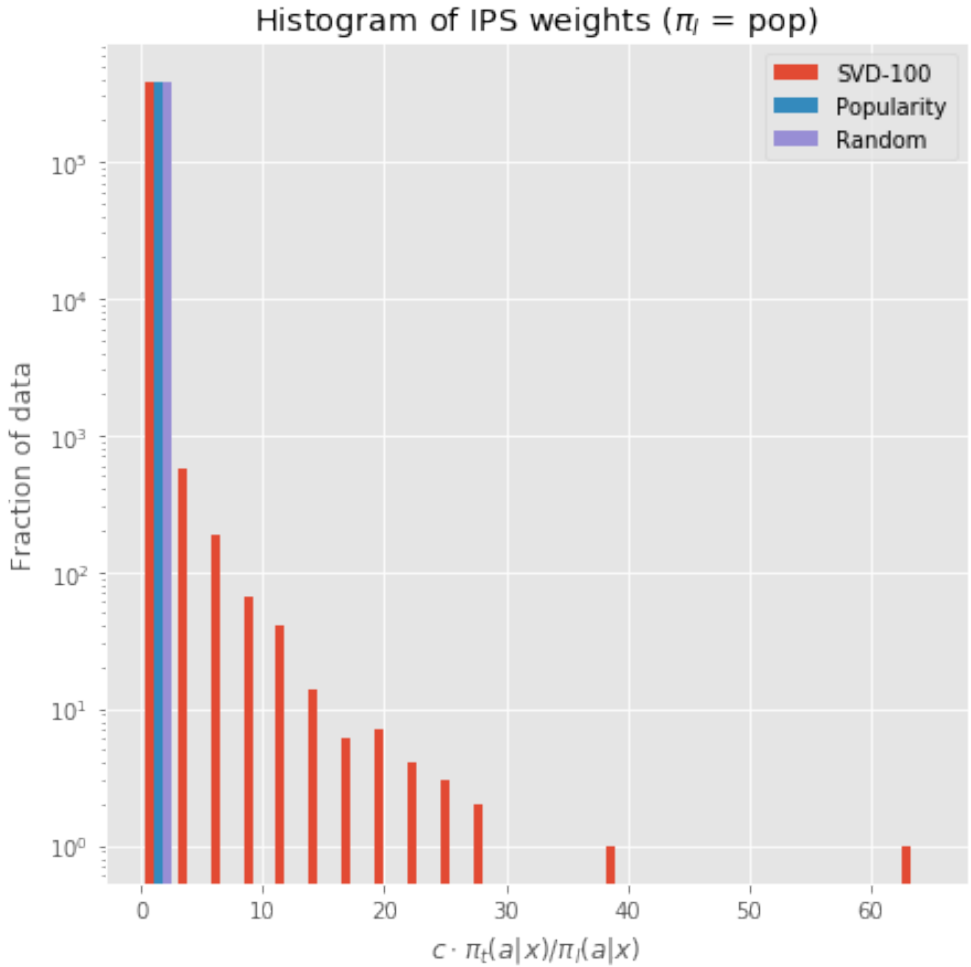

criteo

# Evaluate an organic model using Bandit



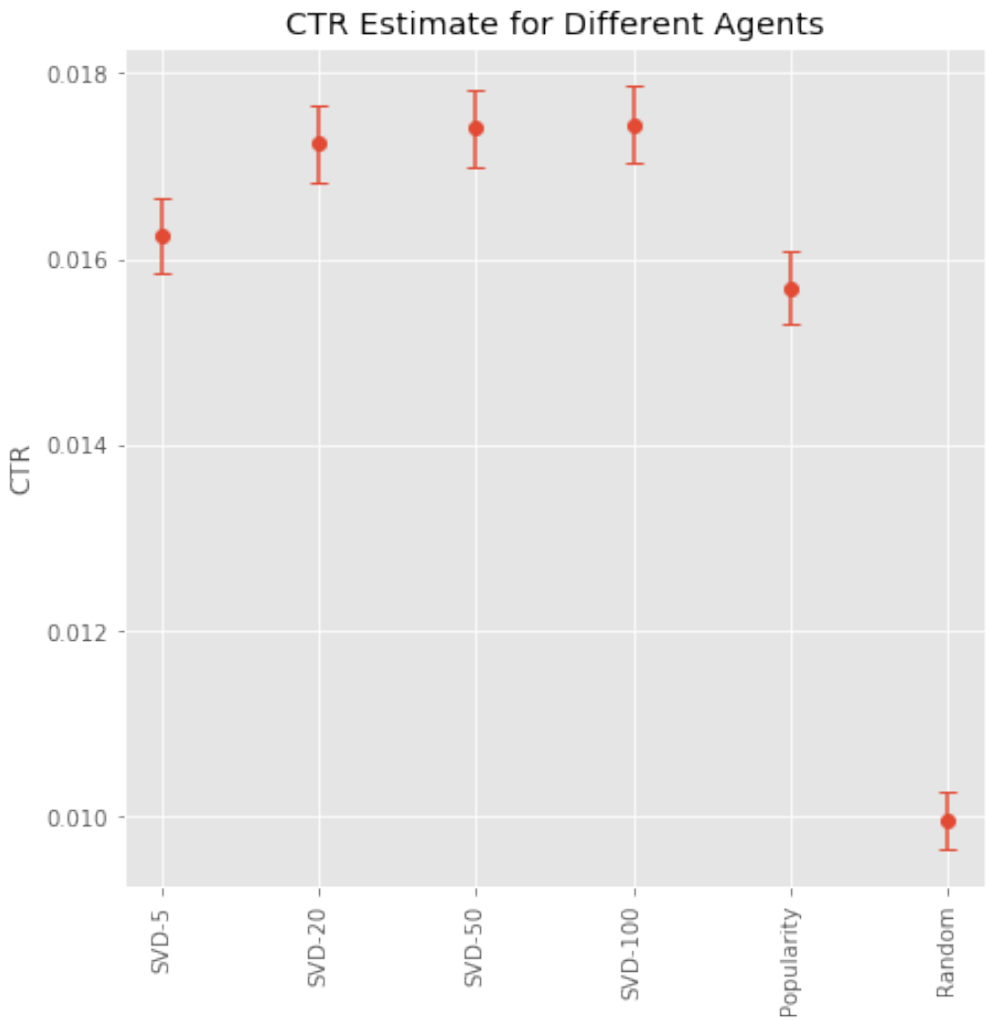

criteo

# Evaluate an organic model using Bandit




criteo

# Evaluate an organic model using Bandit




criteo

# LIkelihood Based Agents

criteo

# Likelihood Based Agent

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

where

- $\beta$ 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 $\boldsymbol{a}_n$

criteo

# Likelihood Based Agent

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

where

- $\beta$ 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 $\boldsymbol{a}_n$ *Why?*

criteo

# Likelihood Based Agent

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

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

criteo

# 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)]
```

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?

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?

criteo

# 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?

criteo

# 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.

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

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

criteo

# 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.

criteo

# 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:

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

# Other ways to train on bandit feedback - counterfactual risk

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

criteo

# Other ways to train on bandit feedback - counterfactual risk

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

Much more in the next model.

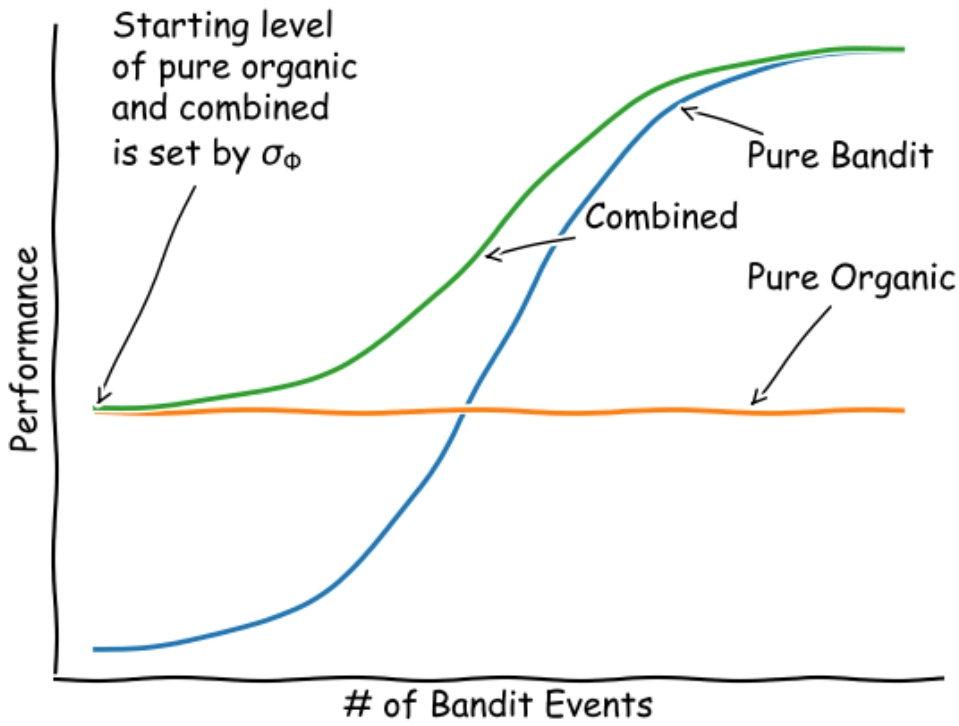
For a short summary see our paper: "Three methods for training on Bandit feedback"

criteo

# Combining Organic Signal with Bandit Signal

criteo

# Pure Organic vs Pure Bandit




criteo

# Pure Organic vs Pure Bandit

A simple algorithm:

# Pure Organic vs Pure Bandit

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.

# Pure Organic vs Pure Bandit

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

# Pure Organic vs Pure Bandit

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.

# Pure Organic vs Pure Bandit

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.

*What advantages or challenges do you have in this approach?*

# Product Recommendation for Ad Optimization

criteo

# Objective: Ad CTR Optimization

Toy World: An online advertising ecosystem with **one e-commerce website** (where the user can buy) and **one publisher website** (where we can show the ads).

**Simplifying assumption:** constant probability of buying once on-site.

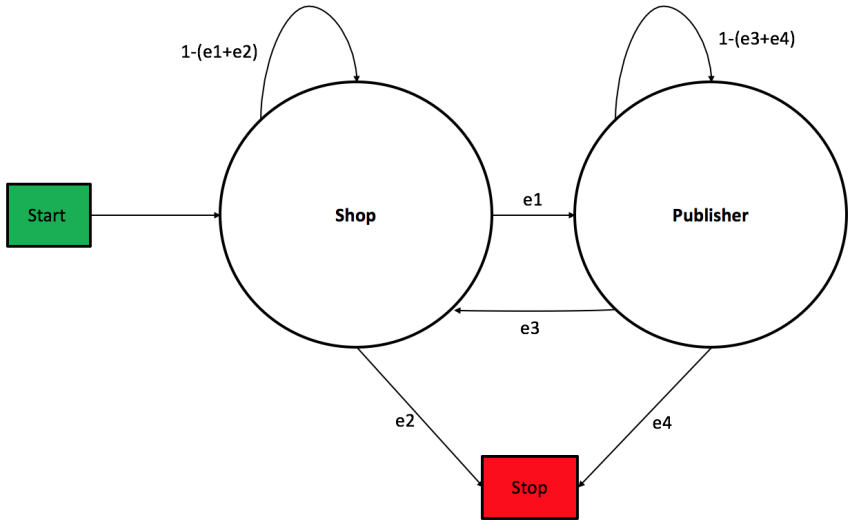



Figure 1: FSM of organic user behavior

criteo

# Objective: Ad CTR Optimization

**The goal of Recommendation: Show the product ad that maximizes CTR**

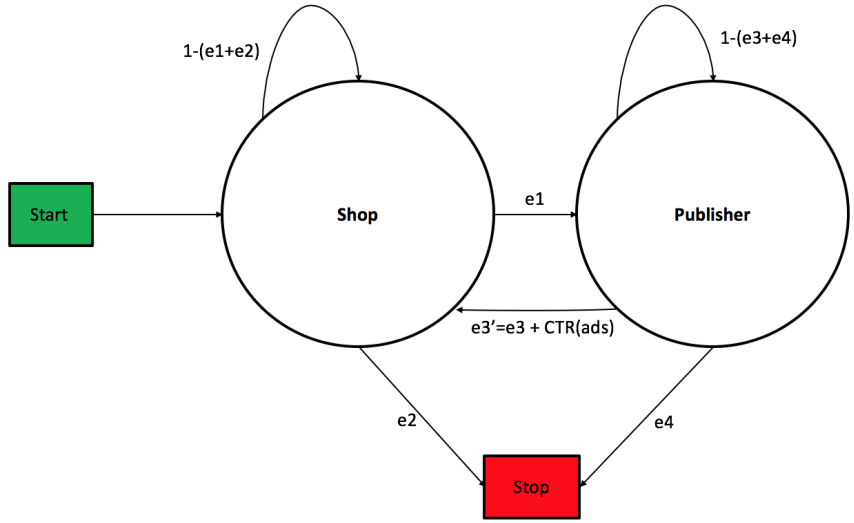
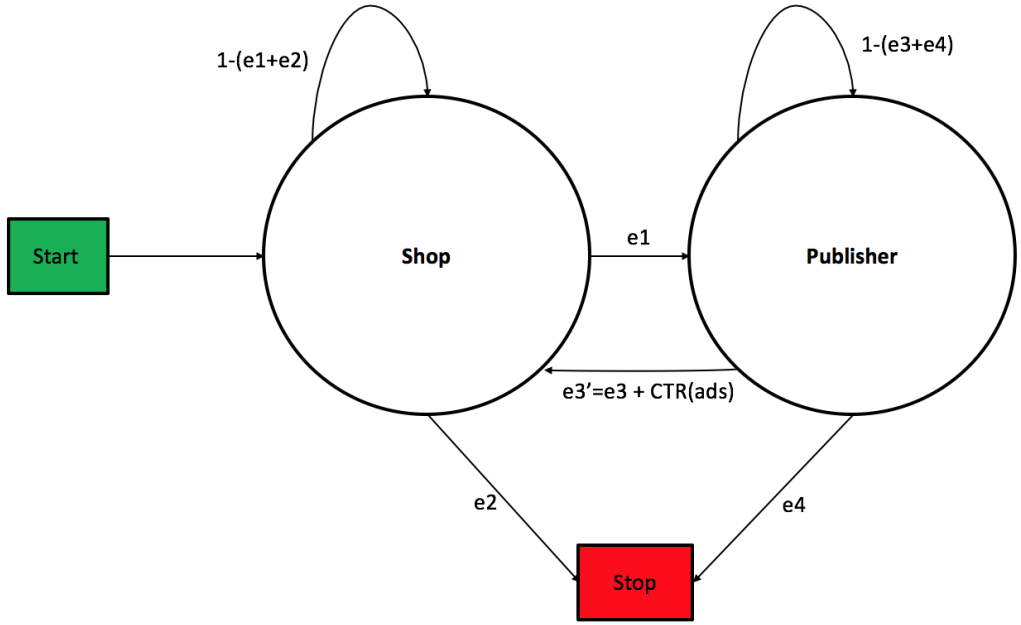



Figure 2: FSM of ad-influenced user behavior

criteo

# Two alternative sources for the Implicit Feedback Matrix




- **Standard recommendation literature:** Look at the user shopping behavior matrix (left hand side matrix)
- **Computational advertising/Bandits literature:** Look at the user CTR matrix (right hand side matrix)

criteo

# Factorization of Implicit Feedback User-Product Matrix

- **Implicit Feedback Matrix:** NbUsers x NbProducts, Binary response, Very Sparse
- Many factorization objectives: MLE, Ranking ...
- **Very similar online performance!**

criteo

# The Supervised Objective

Choose the best personalized ad $ad_i^*$ for a user $u_i$, by framing the problem of CTR prediction as a *Maximum Likelihood Estimation (MLE)* problem which leads to the following formula:

$$ad_i^* = argmax_j\{P^{\pi_0}(ad_j^{click}|u_i, ad_j^{view})\} \quad (1)$$

# The Supervised Objective

$$ad_i^* = argmax_j\{P^{\pi_0}(ad_j^{click}|u_i, ad_j^{view})\}$$

where:

- $\pi_0$ is the current recommendation policy: $ad_j^{view} \sim \pi_0(.|u_i)$
- $ad_j^{view}$ is the event of displaying an ad for product $j$
- $ad_j^{click}$ is the event of clicking an ad for product $j$
- $u_i$ is the profile of the user $i$

# Shortcomings of the supervised objective

Three issues:

- **First**, in the worst case, simply fitting supervised models to your historical data (collected with an older logging policy) will suffer from selection bias and could give you with high confidence a bad recommendation policy - *Simpsons Paradox*
- **Second**, for models with limited capacity, you might spend too much fitting power on the wrong area of logging policy
- **Third**, even if the models have high capacity (DNNs) and the resulting estimators are somehow unbiased, the variance will be huge outside of the support of the logging policy!

# Shortcomings - degree of severity

Three issues:

- **First** - Low (under some assumptions on how the causal graph looks like)
- **Second** - Medium (there are classes of models that are supposed to be able to handle changes in domain - see Gaussian Processes)
- **Third** - Very high! (counterfactual inference and exploration)

# Shortcomings of the supervised approach

Let's look at the 3 problems in more detail

criteo

# Simpsons paradox.

# Simpsons paradox. An Example

| | Overall CTR | Male | Female |
|---|---|---|---|
| Sport | 0.5 | 0.4 | 0.8 |
| Movie | 0.6 | 0.3 | 0.7 |

Figure 3: Simpson's paradox example - Image source: Lihong Li WSDM Tutorial
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tutorial.pdf

criteo

# Simpsons paradox. An Example

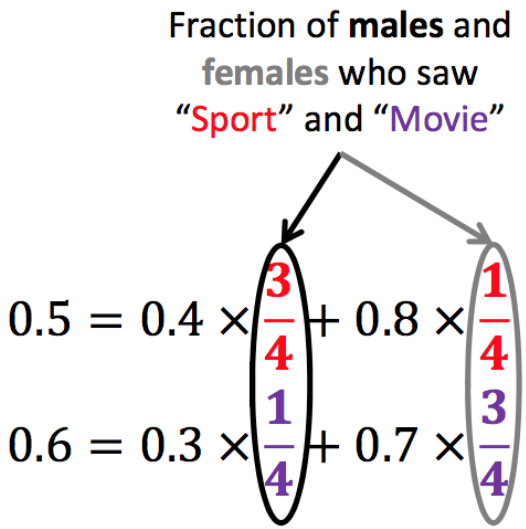



Figure 4: Mixing weights effect - Image source: Lihong Li WSDM Tutorial
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tutorial.pdf

criteo

# Simpsons paradox. An Example

- CTR without knowing gender:
  $P(click|action = sport) = 0.5$
  $P(click|action = movie) = 0.6$
- CTR knowing gender:
  $P(click|action = sport, user = m) = 0.4$
  $P(click|action = movie, user = m) = 0.3$
  $P(click|action = sport, user = f) = 0.8$
  $P(click|action = movie, user = f) = 0.7$

# Simpsons paradox. An Example

Comparing the decisions before and after observing gender

Online traffic: 4 f, 4 m

**Before:** Decision: show movie recommendation to everyone

Offline Expected Payoff $= (4 * 0.6 + 4 * 0.6)/8 = 0.6$

Online Observed Payoff $= (4 * 0.3 + 4 * 0.7)/8 = 0.5$

The offline estimator is too optimistic, the ABTest will go bad!

**After:** Decision: show best arm (news sports/movie) for each gender

Offline Expected Payoff $= (4 * 0.4 + 4 * 0.8)/8 = 0.6$

Online Observed Payoff $= (4 * 0.4 + 4 * 0.8)/8 = 0.6$

The offline estimator is unbiased, the ABTest will go as expected!

criteo

# Simpsons paradox in Recommendation Systems

**Q: Can we check?**

**A:** Yes, using Pearls *Backdoor Criterion* we can check that under the following causal graph representing a generic Recommender System there is no Simpsons paradox:

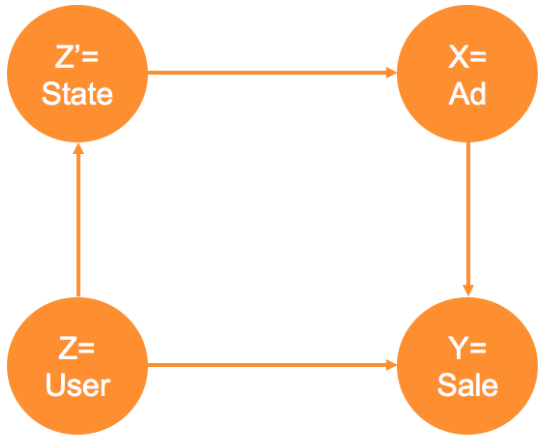



Figure 5: Causal Graph for Ad Recommendations

criteo

# Simpsons paradox in Recommendation Systems

- The principle is simple: The paths connecting X and Y are of two kinds, causal and spurious.
- Causative associations are carried by the causal paths, namely, those tracing arrows directed from X to Y .
- The other paths carry spurious associations and need to be blocked by conditioning on an appropriate set of covariates. All paths containing an arrow into X are spurious paths, and need to be intercepted by the chosen set of covariates.

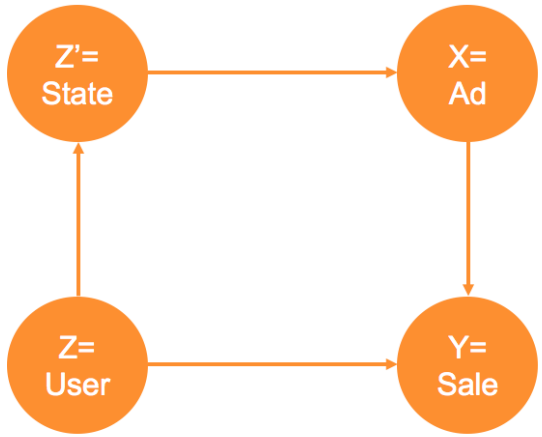



Figure 6: Causal Graph for Ad Recommendations

criteo

# Simpsons paradox in Recommendation Systems

When dealing with a singleton covariate Z, as in the Simpsons paradox, we need to merely ensure that:

- Z is not a descendant of X - True
- Z blocks every path that ends with an arrow into X - True

**Lesson: Always condition on all features used for acting!**

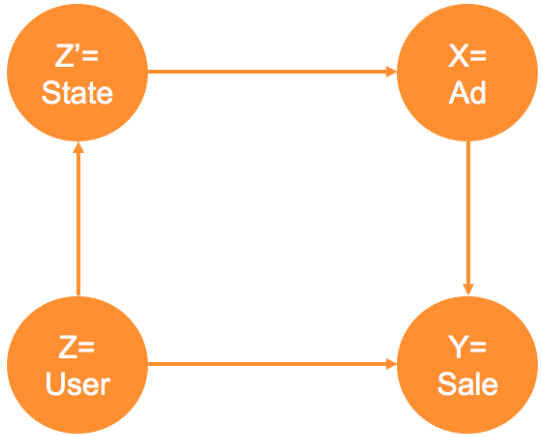



Figure 7: Causal Graph for Ad Recommendations

criteo

# Simpsons paradox in Recommendation Systems

So what is different in Reco vs. the cases where Simpson's paradox is present?
Real World cases that are facing Simpson's paradox:

- Observational study data: The criteria used for past actions/decisions are not fully logged/observable.
- Outcome: first formal policy

# Fitting the wrong problem

criteo

# Fitting the wrong problem

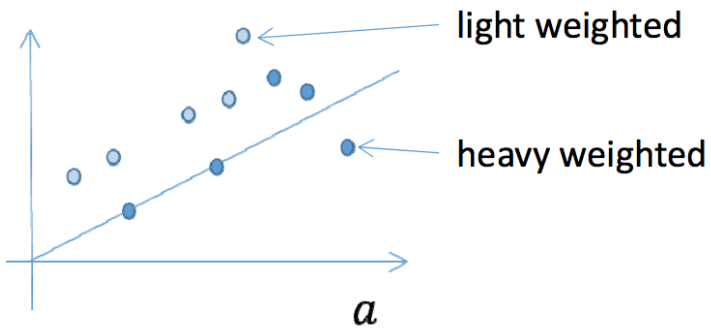



Figure 8: Fitting a linear model of reward on past data

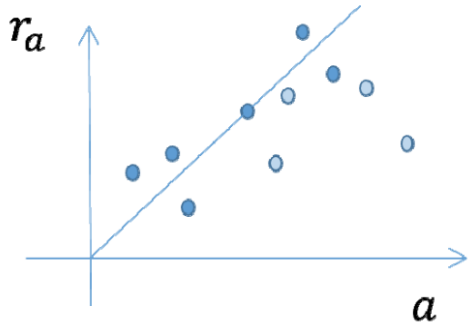



Figure 9: The true best linear predictor

criteo

# Domain Shift affects misspecified models

**Fitting $P(a|x)$ vs fitting $P(a, x)$**

- If we can fit an unbiased model for $P(a|x)$ we should not care about the change of $P(x)$
- Of couse we won't be good at predicting everywhere but only in the places we historically observed $x \in X$.
- What we need is a model that can predict well and have a model of their own uncertainty everywhere.

# Domain Shift affects misspecified models

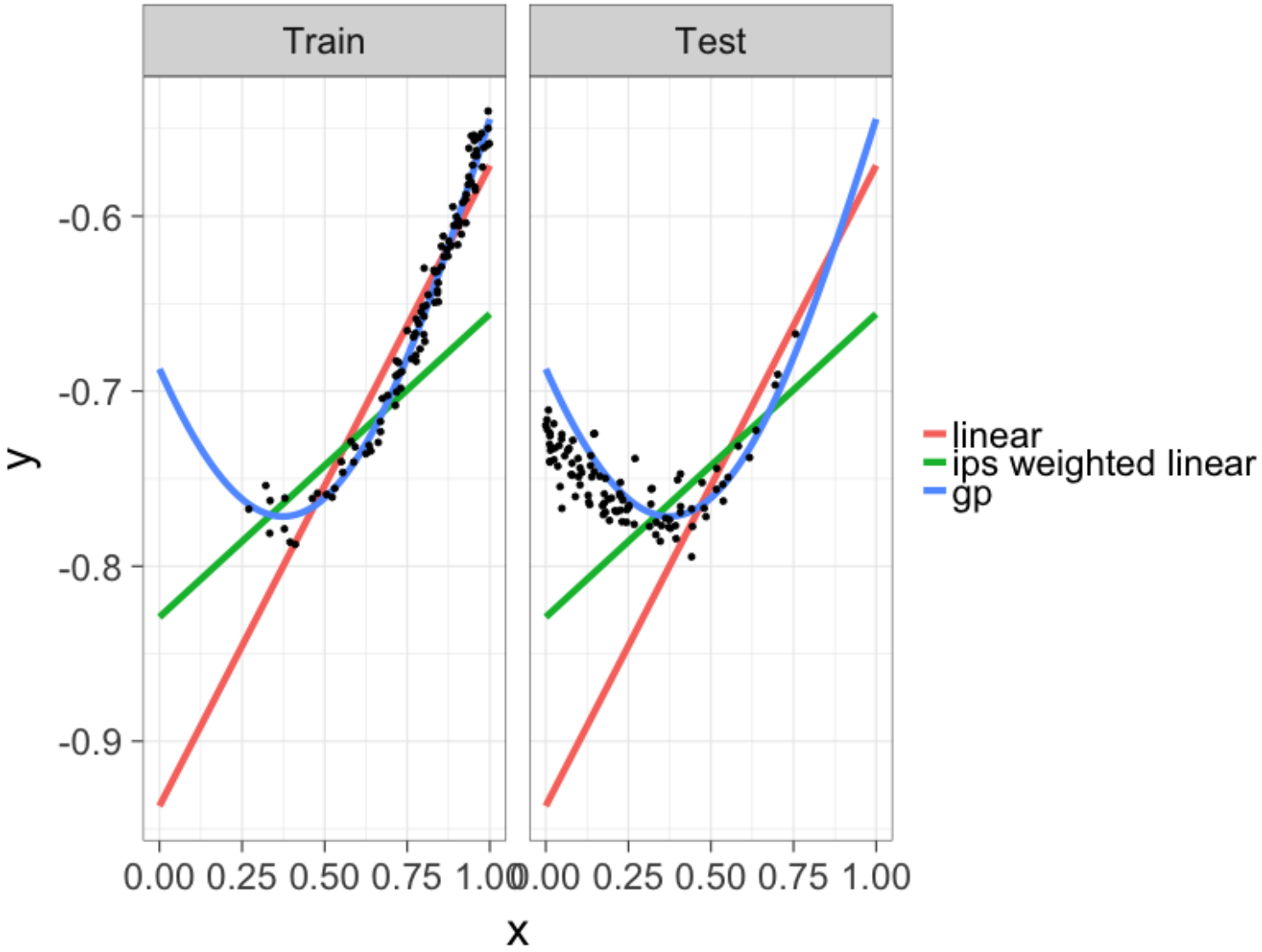



Figure 10: Gaussian Processes vs. Linear Models

criteo

# Fitting the wrong problem

**Example:** Gaussian Processes (GPs)

- GPs are non-parametric models that can model any continuous function and can be arbitrarilly complex.
- DNNs + Dropout = Fast GPs!

criteo

# Fitting the wrong problem

**Conclusion:** This is definitely not a fully solved problem but non-linear models suffer from it a lot less than linear models shown in toy examples.

# Variance

# Variance

The main real problem remains variance outside of the current support of $P(x)$.

**The only answers: Exploration and Counterfactual Reasoning.**

We will introduce the Counterfactual/Causal framework in the next section.

criteo

# Causal Inference Framework

criteo

# Causal Vocabulary

- **Classical setup:** One single action - Treatment: The real action, Control: The placebo action (do vs. not do)
- **Stochastic setup:** Multiple actions - Treatment and Control are both distributions (aka stochastic policies) over all actions (classical setup is a special case)
- In the case of Criteo: **Control policy = Existing Reco System**

# Causal Vocabulary

- **Average Treatment Effect (ATE)**:

$$p(outcome|treatment, pop_B) - p(outcome|control, pop_A)$$

  where $pop_B$ and $pop_A$ are populations of randomly split users

- **Individual Treatment Effect (ITE)**:

$$p(outcome|treatment, user_x) - p(outcome|control, user_x)$$

  where both probabilities are computed for the same $user_x$

- **Notes:**
  - ATE is what we measure with an A/B test
  - In Recommendation, we are interested to optimize the ITE (personalization!)

criteo

# Recommendation Policy

- We assume a *stochastic policy* $\pi_x$ that associates to each user $u_i$ and product $p_j$ a probability for the user $u_i$ to be exposed to the recommendation of product $p_j$:

$$p_j \sim \pi_x(.|u_i)$$

- For simplicity we assume showing no products is also a valid intervention in $\mathcal{P}$.

# Policy Rewards

- Reward $r_{ij}$ is distributed according to an unknown conditional distribution $r$ depending on $u_i$ and $p_j$:

$$r_{ij} \sim r(.|u_i, p_j)$$

- The reward $R^{\pi_x}$ associated with a policy $\pi_x$ is equal to the sum of the rewards collected across all incoming users by using the associated personalized product exposure probability:

$$R^{\pi_x} = \sum_{ij} r_{ij} \pi_x(p_j|u_i) p(u_i) = \sum_{ij} R_{ij}^{\pi_x}$$

# Individual Treatment Effect

- The *Individual Treatment Effect (ITE)* value of a policy $\pi_x$ for a given user $i$ and a product $j$ is defined as the difference between its reward and the control policy reward:

$$ITE_{ij}^{\pi_x} = R_{ij}^{\pi_x} - R_{ij}^{\pi_c}$$

- We are interested in finding the policy $\pi^*$ with the *highest sum of ITEs*:

$$\pi^* = arg \max_{\pi_x} \{ITE^{\pi_x}\}$$

where: $ITE^{\pi_x} = \sum_{ij} ITE_{ij}^{\pi_x}$

# Optimal ITE Policy

- It is easy to show that, starting from any control policy $\pi_c$, the best incremental policy $\pi^*$ is the policy that shows deterministically to each user $u_i$ the product $p_i^*$ with the highest personalized reward $r_i^*$:

$$\pi^* = \pi_{det} = \begin{cases} 1, & \text{if } p_j = p_i^* \\ 0, & \text{otherwise} \end{cases}$$

- Note: This assumes non-fatigability, e.g. non-diminishing returns of recommending to the user the same product over and over (no user state/ repeated action effects at play)

# State of art in Estimating ITE from Observational Data

criteo

# 3 Main Areas of Research

- **Causal literature:** Counterfactual estimators (IPS, Doubly Robust, Self-Normalized IPS)
- **Offpolicy in RL:** Robust/Safe, worst-case estimators (Sample Variance Penalization, Natural Gradient, Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Adversarial Attacks)
- **Causality as Domain Adaptation:** Optimal embeddings for Counterfactual Reasoning

criteo

# Importance Weigthing Approaches

criteo

# Counterfactual estimators

This work is around resampling the logged data to simulate another distribution and the main work is on limiting variance for the estimator

criteo

# Counterfactual estimators

- Direct Method (aka Regression Estimator)
- IPS
- Doubly Robust
- Self Normalized
- MAGIC

criteo

# Direct Method (aka Regression Estimator)

Solution: Fit the MLE

- Pros: Easy
- Cons: Sampling/selection bias + Model bias (put the capacity in the wrong area)

# Counterfactual Method: IPS

Solution: Randomize and use IPS

- Pros: Unbiased if the new actions have support in the old policy / Bounds see page 37 in tutorial
- Cons: High variance

$$\hat{r}_{ij} \approx \frac{y_{ij}}{\pi_c(p_j|u_i)}$$

criteo

# Doubly Robust

- A combination between Direct Estimator and IPS
- **Doubly** Robust = It fails only if both IPS and Direct Estimator fail

criteo

# OffPolicy Estimators

criteo

# SVP: Sample Variance Penalizations

$$SVP_\lambda = argmin_{f \in F} P_n(f, X) + \lambda \sqrt{\frac{V_n(f, X)}{n}}$$

where:

- $P_n(f, X)$ is the empirical risk on the function $f$ and data $X$
- $n$ is the number of samples
- $V_n$ is the empirical variance

`https://arxiv.org/pdf/0907.3740.pdf`

criteo

# POEM: Policy Optimizer for Exponential Models

$$CRM_\lambda = argmin_{f \in F} P_n^{CRM}(f, X) + \lambda \sqrt{\frac{V_n^{CRM}(f, X)}{n}}$$

where:

- $P_n^{CRM}(f, X) = \frac{1}{n} \sum \delta_i \frac{f(y_i|x_i}{p_i}$ is the counterfactual risk on the function $f$ and data $X$ with propensities $p$
- $n$ is the number of samples
- $V_n$ is the empirical variance

`https://arxiv.org/pdf/0907.3740.pdf`

criteo

# Natural Gradient Descent

- Gradient depends on choice of coordinate system.
- Newtons method is invariant to affine coordinate transformations, but not to general coordinate transformations.
- The covariant gradient is also called natural gradient
- Kullback-Leibler divergence between distributions over paths induced by the policies. Second-order Taylor approximation. Fisher information matrix, independent of the choice of parameterization of the class of distributions.

$$NGD_\lambda = L_{\pi_{old}} + \lambda KL(\pi_{old}(.|s)|\pi(.|s))$$

# Truncated Natural Policy Gradient Algorithm

- for iteration=1, 2, . . . do
- Run policy for T timesteps or N trajectories
- Estimate advantage function at all timesteps
- Compute policy gradient g
- Use CG (with Hessian-vector products) to compute $F^{-1}g$
- Update policy parameter $\theta = \theta_{old} + \alpha F^{-1}g$
- end for

# Causal Embeddings

criteo

# Idea: ITE as Extreme Domain Adaptation

criteo

# Distance between treated and control distributions

One approach to the problem of estimating the function $f(x)$ is by learning the two functions $m_0(x)$ and $m_1(x)$ using samples from $p(Y_t|x,t)$.

This is similar to a standard machine learning problem of learning from finite samples.

However, there is an additional source of variance at work here:

For example, if mostly rich patients received treatment $t = 1$, and mostly poor patients received treatment $t = 0$, we might have an unreliable estimation of $m_1(x)$ for poor patients.

# Distance between treated and control distributions

In this paper we upper bound this additional source of variance using an Integral Probability Metric (IPM) measure of distance between two distributions $p(x|t=0)$, and $p(x|t=1)$, also known as the control and treated distributions.

criteo

# Distance between treated and control distributions

In practice we use two specific IPMs:

- the Maximum Mean Discrepancy
- and the Wasserstein distance. We show that the expected error in learning the individual treatment effect function $f(x)$ is upper bounded by the error of learning $Y_1$ and $Y_0$, plus the IPM term. In the randomized controlled trial setting, where $t = x$, the IPM term is 0, and our bound naturally reduces to a standard learning problem of learning two functions.

criteo

# The bound

The bound we derive points the way to a family of algorithms based on the idea of representation learning: Jointly learn hypotheses for both treated and control on top of a representation which minimizes a weighted sum of the factual loss (the standard supervised machine learning objective), and the IPM distance between the control and treated distributions induced by the representation.

criteo

# Distance between treated and control distributions

This can be viewed as learning the functions $m_0$ and $m_1$ under a constraint that encourages better general- ization across the treated and control populations. In the Experiments section we apply algorithms based on multi-layer neural nets as representations and hypotheses, along with MMD or Wasserstein distributional distances over the representation layer; see Figure 1 for the basic architecture.

# CausE: Causal Embeddings For Recommendation

criteo

# IPS Solution For $\pi^*$

- In order to find the optimal policy $\pi^*$ we need to find for each user $u_i$ the product with the highest personalized reward $r_i^*$.
- In practice we do not observe directly $r_{ij}$, but $y_{ij} \sim r_{ij}\pi_c(p_j|u_i)$.
- **Quick parenthesis:** Normal MF for Reco would just decompose the matrix of $y_{ij}$!
- **Current approach to estimate $r_{ij}$:** *Inverse Propensity Scoring (IPS)*-based methods to predict the unobserved reward $r_{ij}$:

$$\hat{r}_{ij} \approx \frac{y_{ij}}{\pi_c(p_j|u_i)}$$

- Note: This assumes we have incorporated randomization in the current policy $\pi_c$

# Addressing The Variance Issues Of IPS

- Main shortcoming: IPS-based estimators do not handle well big shifts in exposure probability between treatment and control policies (products with low probability under the logging policy $\pi_c$ will tend to have higher predicted rewards).
- **Minimum variance attained when:** $\pi_c = \pi_{rand}$**.**
- $\pi_{rand}$ **= Recommend all products uniformly.** This means zero recommendation performance!
- **Our question:** Could we learn from $\pi_c$ a predictor for performance under $\pi_{rand}$ and use it to compute the optimal product recommendations $p_i^*$?

criteo

# Our Approach: Causal Embeddings (CausE)

- We are interested in building a good predictor for recommendation outcomes under random exposure for all the user-product pairs, which we denote as $\hat{y}_{ij}^{rand}$.
- We assume that we have access to a large sample $S_c$ from the logging policy $\pi_c$ and a small sample $S_t$ from the randomized treatment policy $\pi_{t=rand}$ (e.g. the logging policy $\pi_c$ uses *e-greedy* randomization).
- To this end, we propose a multi-task objective that jointly factorizes the matrix of observations $y_{ij}^c \in S_c$ and the matrix of observations $y_{ij}^t \in S_t$.

criteo

# Causality as Domain Adaptation - Related Work


- Johansson, Fredrik, Uri Shalit, and David Sontag. *Learning representations for counterfactual inference.* ICML 2016.
- Louizos, Christos, et al. *Causal effect inference with deep latent-variable models.* NIPS 2017.
- Rosenfeld, Nir, Yishay Mansour, and Elad Yom-Tov. *Predicting Counterfactuals from Large Historical Data and Small Randomized Trials.* WWW 2017.


criteo

# Predicting Rewards Via Matrix Factorization

- Using the MF model, we assume that both the expected factual control and treatment rewards can be approximated as linear predictors over the **shared** user representations $u_i$:

$$y_{ij}^c \approx < p_j^c, u_i >$$

$$y_{ij}^t \approx < p_j^t, u_i >$$

- As a result, we can approximate the ITE of a user-product pair $i, j$ as the difference between the two:

$$\widehat{ITE}_{ij} = < p_j^t, u_i > - < p_j^c, u_i > = < w_j^\Delta, u_i >$$

# Proposed joint optimization solution

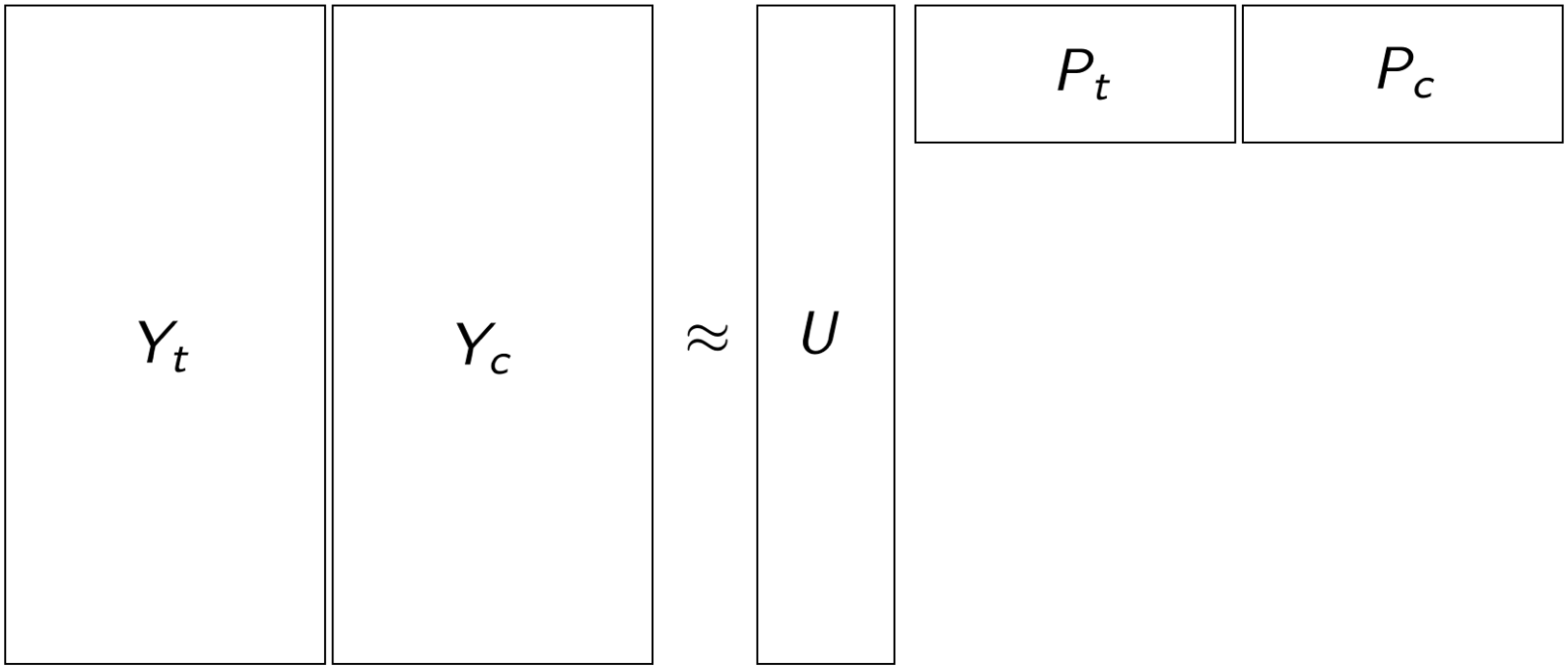



Figure 11: The joint MF problem.

# Sub-objective #1: Treatment Loss Term $L_t$

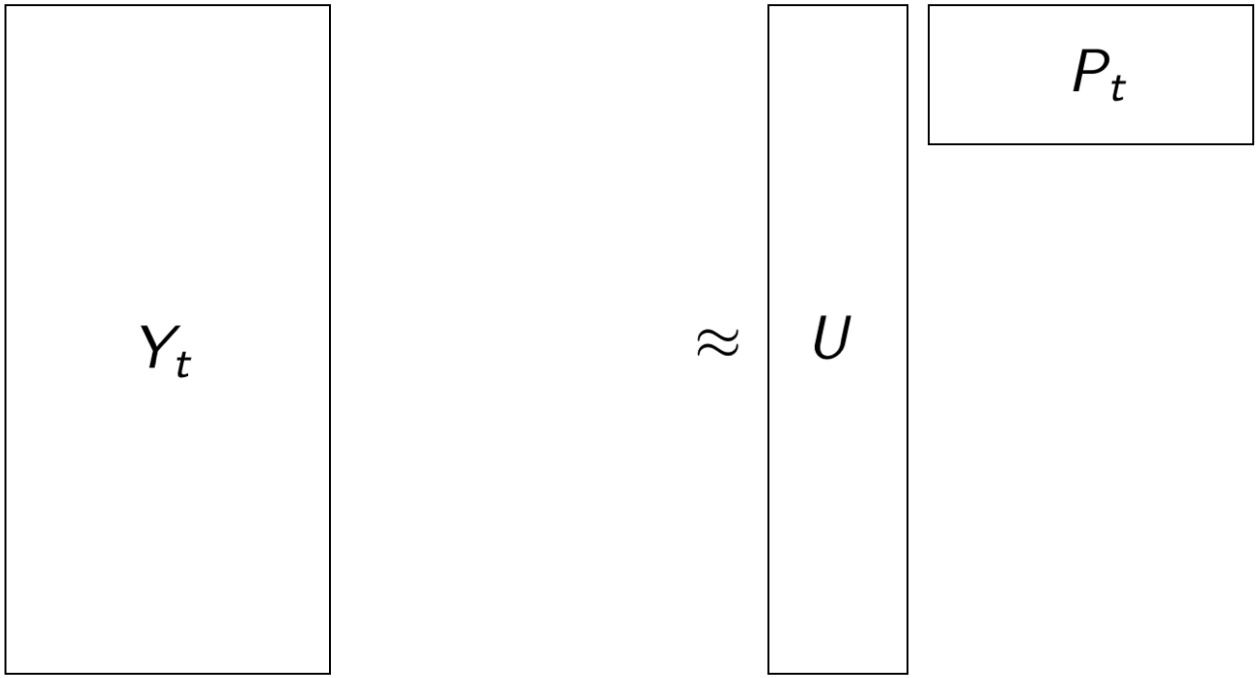



Figure 12: MF of the $Y_t$ matrix

# Sub-objective #1: Treatment Loss Term $L_t$

- We define the first part of our joint prediction objective as the supervised predictor for $y_{ij}^t$, trained on the limited sample $S_t$:

$$L_t(P_t) = \sum_{(i,j,y_{ij}) \in S_t} l_{ij}^t = L(UP_t, Y_t) + \Omega(P_t)$$

where:

- $P_t$ is the parameter matrix of treatment product representations.
- $U$ is the fixed matrix of the user representations.
- $Y_t$ is the observed rewards matrix.
- $L$ is an arbitrary loss function.
- $\Omega(.)$ is a regularization term over the weights of the model.

# Linking the control and treatment effects

- We can use the translation factor in order to be able to use the model built from the treatment data $S_t$ to predict outcomes from the control distribution $S_c$:

$$< p_j^c, u_i > = < p_j^t - w_j^{\Delta}, u_i >$$

# Sub-objective #2: Control Loss Term $L_c$

- Now we want to leverage our ample control data $S_c$ and we can use our treatment product representations through a translation:

$$L_c(P_t, W^{\Delta}) = \sum_{(i,j,y_{ij}) \in S_c} l_{ij}^c = L(U(P_t - W^{\Delta}), Y_c) + \Omega(P_t, W^{\Delta})$$

  which can be written equivalently as:

$$L_c(P_t, P_c) = \sum_{(i,j,y_{ij}) \in S_c} l_{ij}^c = L(UP_c, Y_c) + \Omega(P_c, P_t - P_c)$$

- Where we regularize the control $P_c$ against the treatment embeddings $P_t$ ($W^{\Delta} = P_t - P_c$).

# Sub-objective #2: Control Loss Term $L_c$

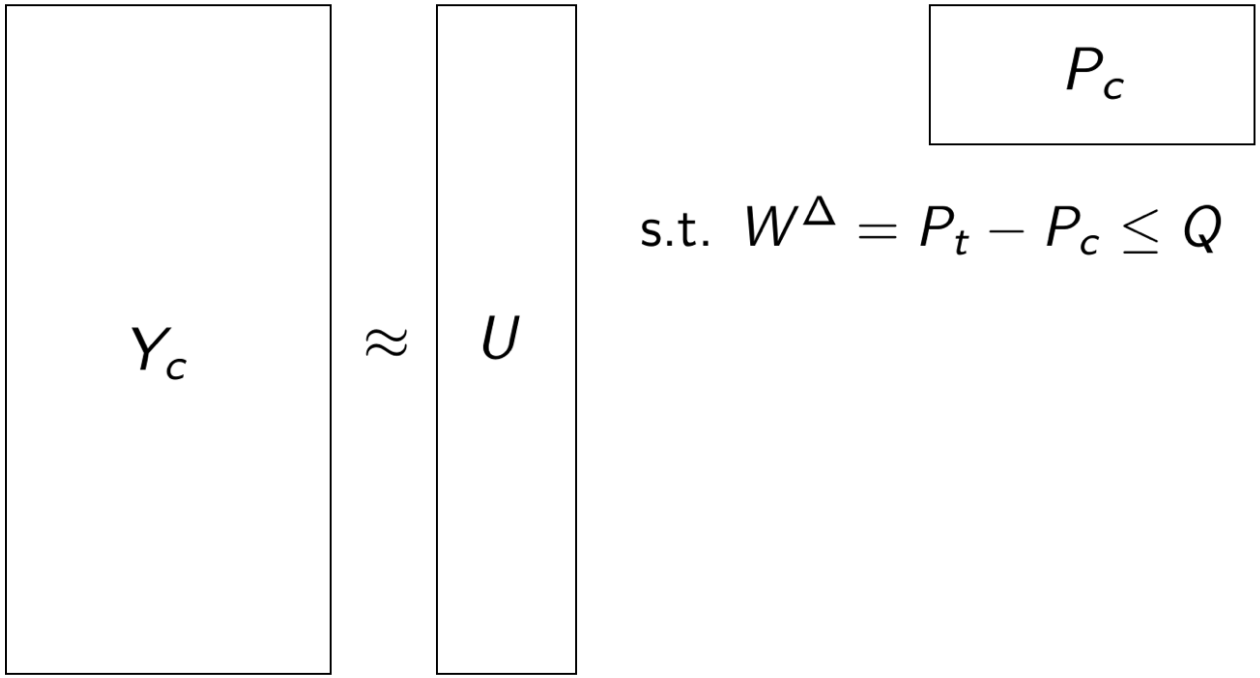



Figure 13: MF of the $Y_c$ matrix

# Proposed joint optimization solution - updated

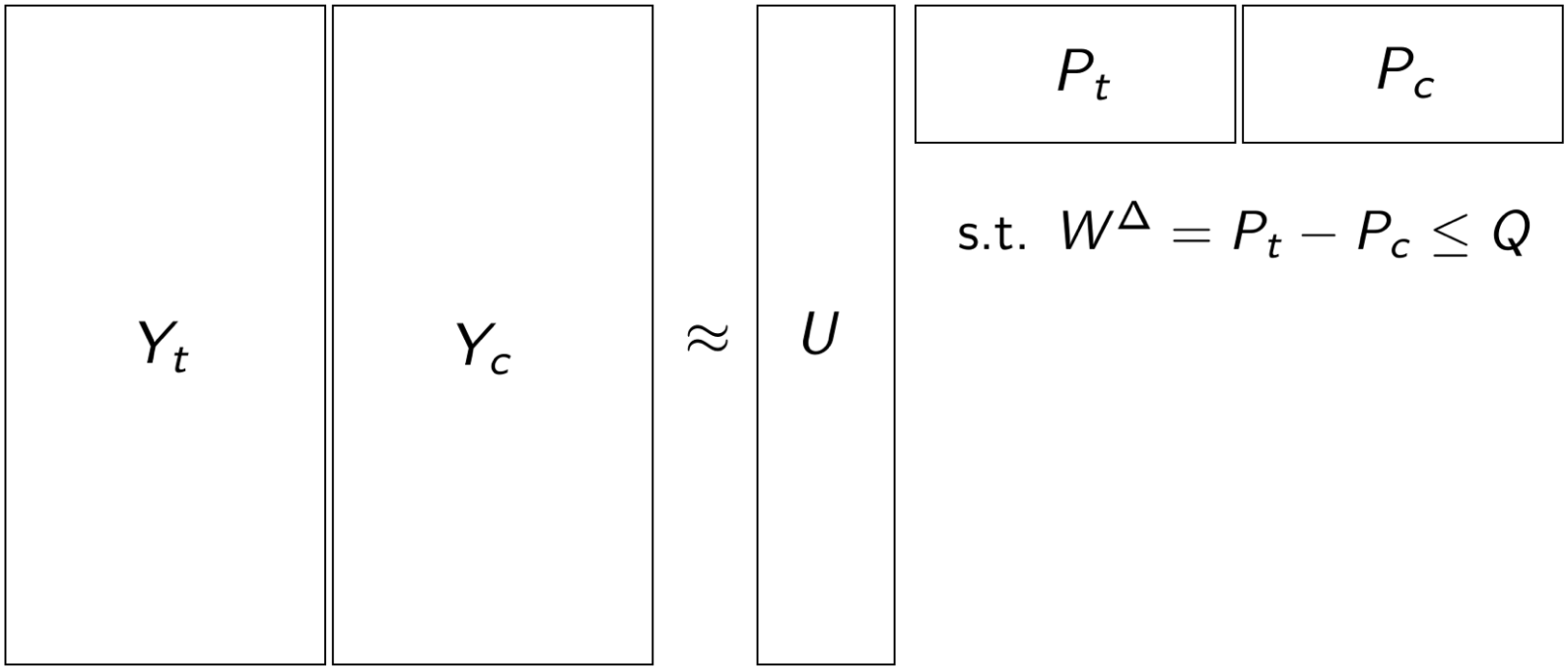



Figure 14: The joint MF problem.

# Overall Joint Objective

- By putting the two tasks together ($L_t$ and $L_c$) and regrouping the loss functions and the regularizer terms, we have that:

$$L^{prod}_{CausE}(P_t, P_c) = L(P_t, P_c) + \Omega_{disc}(P_t - P_c) + \Omega_{embed}(P_t, P_c)$$

Where:

- $L(.)$ is the reconstruction loss function for the concatenation matrix of $P_t$ and $P_c$.
- $\Omega_{disc}(.)$ is a regularization function that weights the discrepancy between the treatment and control product representations.
- $\Omega_{embed}(.)$ is a regularization function that weights the representation vectors.

# Relationship Between IPS And $W^{\Delta}$

$$IPS_{ij} = \frac{\pi_t(p_j|u_i)}{\pi_c(p_j|u_i)} = \frac{< u_i, p_j^t >}{< u_i, p_j^c >} = 1 + \frac{< u_i, w_j^{\Delta} >}{< u_i, p_j^c >}$$

- We can see that $IPS$ is a function of $W^{\Delta}$.
- Regularizing $W^{\Delta} \approx$ regularizing $IPS$

criteo

# Question: How about User Shift?

- **Example of user shift:** The current recommendation solution is targeting a subset of users, lets say active buyers on a website and the new recommendation targets mainly newly signed users (modulo randomization which should give non-zero probabilities for all user product pairs).

criteo

# Generalization Of The Objective To User Shift

- Our objective function can be altered to allow for the user representations to change, we obtain the equation below:

$$L^{user}_{CausE}(U_t, U_c) = L(U_t, U_c) + \Omega_{disc}(U_t - U_c) + \Omega_{embed}(U_t, U_c)$$

- Putting the loss functions associated with the user and product dimension together ($L^{prod}_{CausE}$, $L^{user}_{CausE}$), we reach the final loss function for our method $L_{CausE}(P_t, P_c, U_t, U_c) =$

$$L(P_t, P_c, U_t, U_c) + \Omega_{disc}(P_t - P_c, U_t - Uc) + \Omega_{embed}(P_t, P_c, U_t, U_c)$$

# Algorithm Overview

**Input** : Mini-batches of $S_c = \{(u_i, p_j^c, \delta_{ij}^c)\}_{i=1}^{M_c}$ and $S_t = \{(u_i, p_j^t, \delta_{ij}^t)\}_{i=1}^{M_t}$ , regularization parameters $\lambda_{embed}$ and $\lambda_{dist}$, learning rate $\eta$

**Output:** $P_t, P_c, U_t, U_c$ - Product and User Control and Treatment Matrices

Random initialization of $P_t, P_c, U_t, U_c$ ;

**while** *not converged* **do**

  read batch of training samples;

  **for each** *product* $p_j$ *in* $P_c, P_t$ **do**

    Update product vector: $p_j \leftarrow p_j - \eta \nabla L_{CausE}^{prod}(p, \lambda_{embed}, \lambda_{dist}))$

  **end**

  **for each** *user* $u_i$ *in* $U_c, U_t$ **do**

    Update user vector: $u_i \leftarrow u_i - \eta \nabla L_{CausE}^{user}(u, \lambda_{embed}, \lambda_{dist}))$

  **end**

**end**

**return** $P_t, P_c, U_t, U_c$

**Algorithm 1:** CausE Algorithm: Causal Embeddings For Recommendations

criteo

# Experimental Results

# Experimental Setup

- The task is predicting the outcomes $y_{ij}^{rand}$ under treatment policy $\pi_{rand}$, where all of the methods have available at training time a large sample of observed recommendations outcomes from $\pi_c$ and a small sample from $\pi_{rand}$.
- Essentially it's a classical conversion-rate prediction problem so we measure *Mean-Squared Error (MSE)* and *Negative Log-Likelihood (NLL)*.
- We report lift over average conversation rate from the test dataset.

$$lift_x^{metric} = \frac{metric_x - metric_{AvgCR}}{metric_{AvgCR}}$$

criteo

# Experimental Setup: Matrix Factorization Baselines

We compare our method with the following matrix factorization baselines:

- **Bayesian Personalized Ranking (BRR)** - Directly learn user and product representations via user/item matrix factorization, optimized using the BPR-OPT criteria and trained using LearnBPR.
- **Supervised-Prod2Vec (SP2V)** - Logistic MF method that approximates $y_{ij}$ as a sigmoid over a linear transform of the inner-product between the user and product representations: $\hat{y}_{ij} = \sigma(\alpha < p_j, u_i > + b_i + b_j + b)$

criteo

# Experimental Setup: Causal Inference Baselines

We compare our method with the following causal inference baselines:

- **Weighted-SupervisedP2V (WSP2V)** - SP2V algorithm on propensity-weighted data.
- **BanditNet (BN)-like** - MF with Self-Normalized IPS optimization objective

criteo

# Experimental Setup: Datasets

- We use the *MovieLens10M* and *Netflix* explicit rating datasets (1-5). We process it as follows:
- We binarize the ratings $y_{ij}$ by setting 5-star ratings to 1 (click) and everything else to zero (view only).
- We then create two datasets: regular (REG) and skewed (SKEW), each one with 70/10/20 train/validation/test event splits.

criteo

# Experimental Setup: SKEW Dataset

- **Goal: Generate a test dataset that simulates rewards uniform expose $\pi_t^{rand}$.**
- Method:
  - Step 1: Simulate uniform exposure on 30% of users by rejection sampling.
  - Step 2: Split the rest of 70% of users in 60% train 10% validation
  - Step 3: Add to train a fraction of the test data (e.g. $S_t$) to simulate a small sample from $\pi_t^{rand}$.
- NB: In our experiments, we varied the size of $S_t$ between 1% and 15%.

criteo

# Experimental Setup: Exploration Sample $S_t$

We define 5 possible setups of incorporating the exploration data:

- **No adaptation** *(no)* - trained only on $S_c$.
- **Blended adaptation** *(blend)* - trained on the blend of the $S_c$ and $S_t$ samples.
- **Test adaptation** *(test)* - trained only on the $S_t$ samples.
- **Product adaptation** *(prod)* - separate treatment embedding for each product based on the $S_t$ sample.
- **Average adaptation** *(avg)* - average treatment product by pooling all the $S_t$ sample into a single vector.

criteo

# Results- MovieLens10M

| Method | MovieLens10M (SKEW) | | |
|---|---|---|---|
| | **MSE lift** | **NLL lift** | **AUC** |
| *BPR-no* | – | – | 0.693(±0.001) |
| *BPR-blend* | – | – | 0.711(±0.001) |
| *SP2V-no* | +3.94%(±0.04) | +4.50%(±0.04) | 0.757(±0.001) |
| *SP2V-blend* | +4.37%(±0.04) | +5.01%(±0.05) | 0.768(±0.001) |
| *SP2V-test* | +2.45%(±0.02) | +3.56%(±0.02) | 0.741(±0.001) |
| *WSP2V-no* | +5.66%(±0.03) | +7.44%(±0.03) | 0.786(±0.001) |
| *WSP2V-blend* | +6.14%(±0.03) | +8.05%(±0.03) | 0.792(±0.001) |
| *BN-blend* | – | – | 0.794(±0.001) |
| *CausE-avg* | +12.67%(±0.09) | +15.15%(±0.08) | 0.804(±0.001) |
| *CausE-prod-T* | +07.46%(±0.08) | +10.44%(±0.09) | 0.779(±0.001) |
| **CausE-prod-C** | **+15.48%(±0.09)** | **+19.12%(±0.08)** | **0.814(±0.001)** |

Table 1: Results for MovieLens10M on the Skewed (SKEW) test datasets. All three versions of the *CausE* algorithm outperform both the standard and the IPS-weighted causal factorization methods, with *CausE-avg* and *CausE-prod-C* also out-performing BanditNet.

criteo

# Results - Netflix

| Method | Netflix (SKEW) | | |
|---|---|---|---|
| | **MSE lift** | **NLL lift** | **AUC** |
| *BPR-no* | – | – | 0.665(±0.001) |
| *BPR-blend* | – | – | 0.671(±0.001) |
| *SP2V-no* | +10.82%(±0.02) | +10.19%(±0.01) | 0.752(±0.002) |
| *SP2V-blend* | +12.82%(±0.02) | +11.54%(±0.02) | 0.764(±0.003) |
| *SP2V-test* | +05.67%(±0.02) | +06.23%(±0.02) | 0.739(±0.004) |
| *WSP2V-no* | +13.52%(±0.01) | +13.11%(±0.01) | 0.779(±0.001) |
| *WSP2V-blend* | +14.72%(±0.02) | +14.23%(±0.02) | 0.782(±0.002) |
| *BN-blend* | – | – | 0.785(±0.001) |
| *CausE-avg* | +15.62%(±0.02) | +15.21%(±0.02) | 0.799(±0.002) |
| *CausE-prod-T* | +13.97%(±0.02) | +13.52%(±0.02) | 0.789(±0.003) |
| **CausE-prod-C** | **+17.82%(±0.02)** | **+17.19%(±0.02)** | **0.821(±0.003)** |

Table 2: Results for Netflix on the Skewed (SKEW) test datasets.

criteo

# Results


Figure 15: Change in MSE lift as more test set is injected into the blend training dataset.

# Conclusions

- We have introduced a novel method for factorizing implicit user-item matrices that optimizes for *incremental recommendation outcomes.*
- We learn to predict user-item similarities under the uniform exposure distribution.
- *CausE* is an extension of matrix factorization algorithms that adds a regularizer term on the discrepancy between the product embeddings that fit the training distribution and their counter-part embeddings that fit the uniform exposure distribution.

criteo

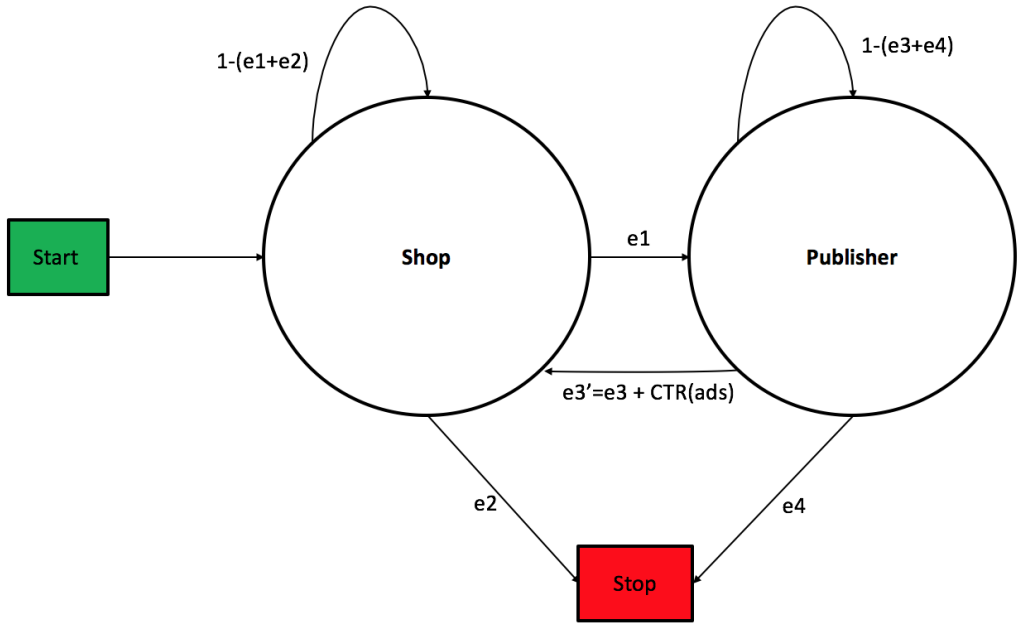
# TL;DR

- **Simple algo for causal MF:** Word2Vec-like learning algo over two matrices with a regularised difference between product vectors.
- **What if we dont have e-greedy but a different randomization scheme?** You can use IPS trick to simulate data from $\pi_{rand}$.

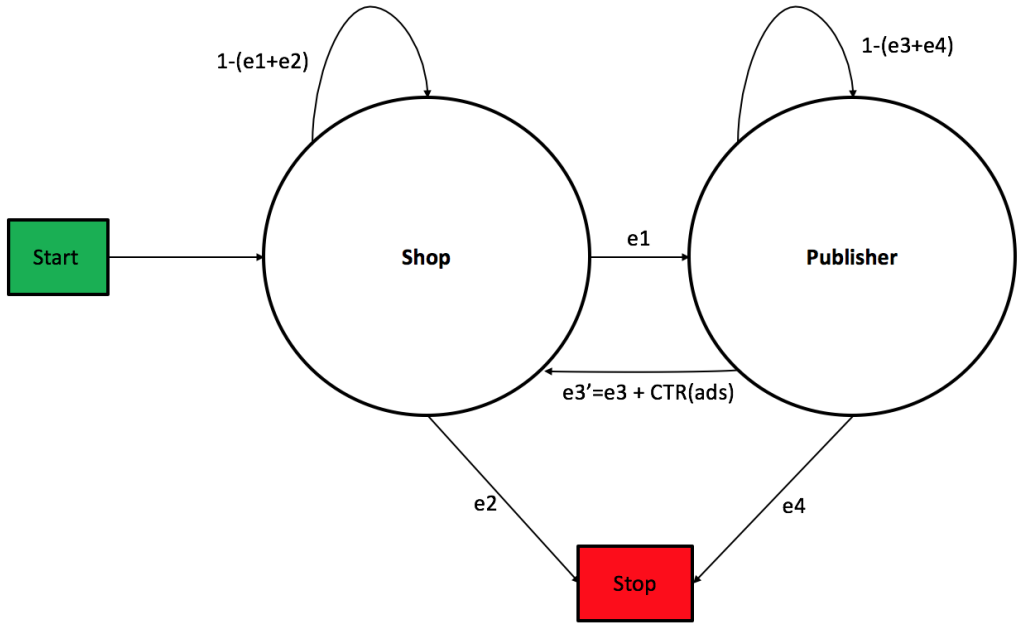
criteo

# Next Steps

In our current work we did not leverage at all the user shopping activity:



Second, by making it a MF task, we removed time from our models



In our upcoming work, we are incorporating both the organic user behavior and taking the time into account.

**Challenges:**

- Sparse rewards
- Delayed rewars
- Extremely large action space

We believe that with reasonable additional assumptions, we can build a realistic user sequence model, optimized for causal recommendations.