Modern Recommendation for Advanced Practitioners
Part I. Recommendation via Reward Modeling

Flavian Vasile, David Rohde, Amine Benhalloum, Martin Bompaire & Dmytro Mykhaylov

December 5, 2019

Speakers

Flavian Vasile

- **Flavian Vasile** is part of the Criteo AI Lab, where he works as the *ML Recommendations Solutions Architect*, with his main focus being on the development of Deep Learning-based Recommendation Systems and on introducing aspects of Causal Inference to Recommendation.
- Before joining Criteo, he worked as a Senior Researcher in the Twitter Advertising Science and as part of the Yahoo! Research Lab, where he mostly focused on Content Understanding problems.
- His current research interests include *Deep Sequential Models for Recommendation* and understanding *Causality in Recommendation*.
- Among his recent research publications, the work on *Causal Embeddings for Recommendation* received the *best paper award at RecSys 2018*, and he is the co-organizer of the *Workshop on Offline Evaluation for Recommender Systems* at *RecSys 2019*.

Speakers

David Rohde

- **David Rohde** David is a research scientist at Criteo. His research focuses on Bayesian statistics and causality especially applied to marketing problems.

Speakers

Amine Benhalloum

- **Amine Benhalloum** is a Senior Machine Learning Engineer at Criteo, working on building large scale representation learning and retrieval systems for recommendation, applying Deep learning to personalize billions of daily display ads, reaching billions of users and connecting them with millions of products.
- His areas of expertise are: large scale machine learning, natural language processing, information retrieval and data intensive systems.
- Before joining Criteo, Amine worked on a variety of topics ranging from Natural Language processing to fraud detection. He holds a master's degree in Applied Mathematics.

Speakers

Martin Bompaire

- **Martin Bompaire** has joined Criteo as a Machine Learning Engineer. He works on making recommendation more incremental, that is building a recommender engine fully aligned with the clients needs and not biased towards the attribution procedure.
- His research is mostly focused on convex optimization, causality and point processes.
- Martin holds Ph.D. in Applied Mathematics whose main topic is Hawkes processes: a point process model to study the cross causality that might occur between series of events and has many applications such as decoding the information (or fake news) propagation in a social network.

Speakers

Dmytro Mykhaylov

- **Dmytro Mykhaylov** is a Senior Software Engineer who works on a new Recommendation System in Criteo based on deep neural networks.
- His areas of expertise are: real-time systems and big-data.
- Before joining Criteo, Dmytro worked on the following domains: advertisement systems, investment banking, and internet browsers.
- Beyond his professional expertise, Dmytro is interested in Quantum Computing, Machine Learning as well as Logical Programming.
- Dmytro holds a Master's degree in Computer Science, the domain of Intellectual Systems and Nets.

Collaborators

Special thanks to our colleagues:

- **That co-authored this course:** Olivier Jeunen
- **That contributed to RecoGym and to some of the methods covered:** Stephen Bonner, Alexandros Karatzoglou, Travis Dunlop, Elvis Dohmatob, Louis Faury, Ugo Tanielian, Alexandre Gilotte

Course

- **Part I. Recommendation via Reward Modeling**
  - I.1. Classic vs. Modern: Recommendation as Autocomplete vs. Recommendation as Intervention Policy
  - I.2. Recommendation Reward Modeling via (Point Estimation) Maximum Likelihood Models
  - I.3. Shortcomings of Classical Value-based Recommendations
- **Part II. Recommendation as Policy Learning** Approaches
  - II.1. Policy Learning: Concepts and Notations
  - II.2. Fixing the issues of Value-based Models using Policy Learning
    - *Fixing Covariate Shift using Counterfactual Risk Minimization*
    - *Fixing Optimizer's Curse using Distributional Robust Optimization*
  - II.3. Recap and Conclusions

Getting Started with RecoGym and Google Colaboratory

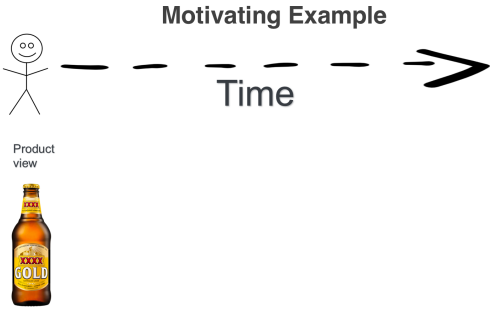
- Open your favorite browser and go to the course repository:
  `https://github.com/criteo-research/bandit-reco`
- You will find the notebooks and the slides for the course
- You can access the notebooks directly on Google Collab in the following manner:
  - `https://colab.research.google.com/github/criteo-research/bandit-reco/blob/master/notebooks/0.%20Getting%20Started.ipynb`
- Alternatively, clone the repository and run the notebooks locally.
- . . . you're all set!

Part I. Recommendation via Reward Modeling

I.1. Classic vs. Modern: Recommendation as Autocomplete vs. Recommendation as Intervention Policy

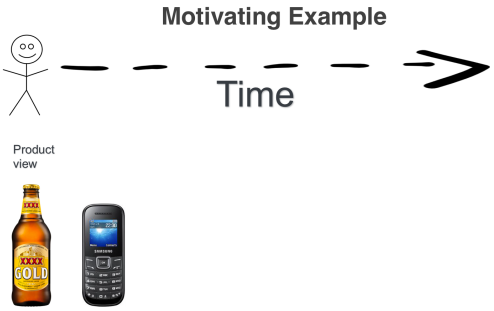
What Makes a Recommender System Modern?

Classic Reco



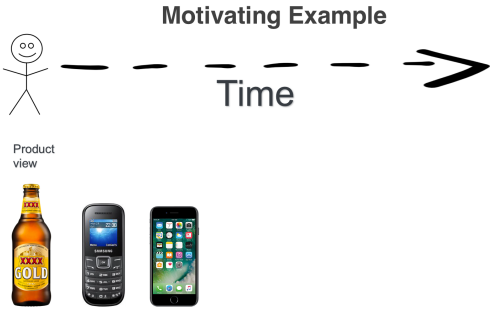
$$P(V_1 = \text{beer})$$

Classic Reco



$$P(V_2 = \text{phone A} | V_1 = \text{beer})$$

Classic Reco



$$P(V_3 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A})$$

Next Item Prediction: Hold out V_3

$$
\begin{aligned}
P(V_3 = \text{phone A} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.27 \\
P(V_3 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.24 \\
P(V_3 = \text{rice} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.10 \\
P(V_3 = \text{couscous} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.09 \\
P(V_3 = \text{beer} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.30
\end{aligned}
$$

What actually happened?

Next Item Prediction: Hold out V_3

$$
\begin{aligned}
P(V_3 = \text{phone A} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.27 \\
P(V_3 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.24 \\
P(V_3 = \text{rice} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.10 \\
P(V_3 = \text{couscous} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.09 \\
P(V_3 = \text{beer} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.30
\end{aligned}
$$

What actually happened? $V_3 = \text{phone B}$

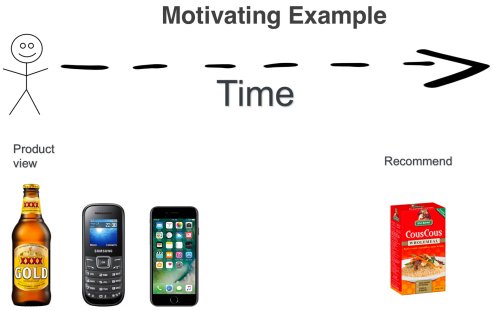
Next Item Prediction: Hold out V_3

$$
\begin{aligned}
P(V_3 = \text{phone A} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.27 \\
P(V_3 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.24 \\
P(V_3 = \text{rice} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.10 \\
P(V_3 = \text{couscous} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.09 \\
P(V_3 = \text{beer} | V_1 = \text{beer}, V_2 = \text{phone A}) &= 0.30
\end{aligned}
$$

What actually happened? $V_3 = \text{phone B}$

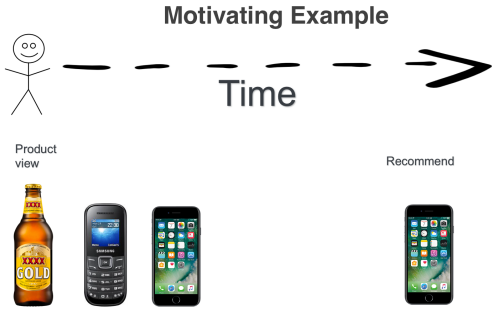
Our model assigned: $P(V_3 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}) = 0.24$

The Real Reco Problem



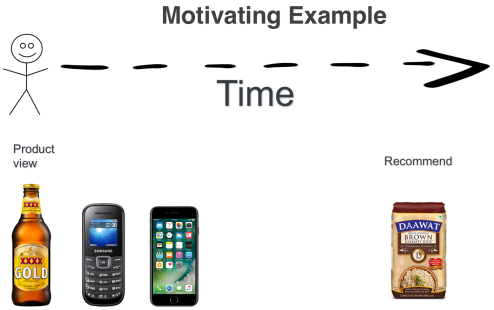
$$P(C_4 = 1 | A_4 = \text{couscous}, V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone B})$$

The Real Reco Problem



$$P(C_4 = 1 | A_4 = \text{phone B}, V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone B})$$

The Real Reco Problem



$$P(C_4 = 1 | A_4 = \text{rice}, V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone B})$$

What we have vs. what we want

Our model:

$$
\begin{aligned}
P(V_4 = \text{phone A} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.27 \\
P(V_4 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.28 \\
P(V_4 = \text{rice} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.12 \\
P(V_4 = \text{couscous} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.14 \\
P(V_4 = \text{beer} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.19
\end{aligned}
$$

Which is a vector of size P that sums to 1.

We want:

What we have vs. what we want

Our model:

$$
\begin{aligned}
P(V_4 = \text{phone A} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.27 \\
P(V_4 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.28 \\
P(V_4 = \text{rice} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.12 \\
P(V_4 = \text{couscous} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.14 \\
P(V_4 = \text{beer} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.19
\end{aligned}
$$

Which is a vector of size P that sums to 1.

We want:

$$
\begin{aligned}
P(C_4 = 1 | A_4 = \text{phone A}, V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= ? \\
P(C_4 = 1 | A_4 = \text{phone B}, V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= ? \\
P(C_4 = 1 | A_4 = \text{rice}, V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= ? \\
P(C_4 = 1 | A_4 = \text{couscous}, V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= ? \\
P(C_4 = 1 | A_4 = \text{beer}, V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= ?
\end{aligned}
$$

Which is a vector of size P where each entry is between 0 and 1.

Implicit Assumption

Our model predicts:

$$
\begin{aligned}
P(V_4 = \text{phone A} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.27 \\
P(V_4 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.28 \\
P(V_4 = \text{rice} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.12 \\
P(V_4 = \text{couscous} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.14 \\
P(V_4 = \text{beer} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone A}) &= 0.19
\end{aligned}
$$

So let's recommend phone B, it has the highest next item probability.

Implicit Assumption

If:

$$P(V_n = a | V_1 = v_1 .. V_{n-1} = v_{n-1}) > P(V_n = b | V_1 = v_1 .. V_{n-1} = v_{n-1})$$

Assume:

$$\begin{aligned} &P(C_n = 1 | A_n = a, V_1 = v_1 .. V_{n-1} = v_{n-1}) \\ &\qquad > P(C_n = 1 | A_n = b, V_1 = v_1 .. V_{n-1} = v_{n-1}) \end{aligned}$$

Vast amounts of academic Reco work assumes this implicitly!

Classical vs. Modern Recommendation

- **Classical Approach:** Recommendation as *Autocomplete*:
  - Typically leverage *organic* user behaviour information (e.g. item views, page visits)
  - Frame the problem either as *missing link prediction/MF* problem, either as a *next event prediction, sequence prediction* problem
- **Modern Approach:** Recommendation as *Intervention Policy*:
  - Typically leverage *bandit* user behaviour information (e.g. ad clicks)
  - Frame the problem either as *click likelihood* problem, either as a *contextual bandit/policy learning* problem

I.1.1. Advantages of the Classical Approach

Academia

- An already established framework
- Standard datasets and metrics
- Easier to publish and compare methods!

Real-World

- Data arises naturally from the use of the application (before any recommendation system in place)
- Methods have standard efficient and well-tested implementations
- Very good initial system!

I.1.2. Limitations of the Classical Approach

What is Wrong with the Classical Formulation?

We are operating under the assumption that the best recommendation policy is, in some sense, the **optimal auto-complete of natural user behavior**.

Solving the Wrong Problem

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.

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.

Solving the Wrong Problem: are the Classic Datasets Logs of RecSys?

- MovieLens: no, explicit feedback of movie ratings [1]
- Netflix Prize: no, explicit feedback of movie ratings [2]
- Yoochoose (RecSys competition '15): no, implicit session-based behavior [3]
- 30 Music: no, implicit session-based behavior [4]
- Yahoo News Feed Dataset: **yes!** [5]
- Criteo Dataset for counterfactual evaluation of RecSys algorithms: **yes!** [6]

Solving the Wrong Problem: are the Classic Datasets Logs of RecSys?

- MovieLens: no, explicit feedback of movie ratings [1]
- Netflix Prize: no, explicit feedback of movie ratings [2]
- Yoochoose (RecSys competition '15): no, implicit session-based behavior [3]
- 30 Music: no, implicit session-based behavior [4]
- Yahoo News Feed Dataset: **yes!** [5]
- Criteo Dataset for counterfactual evaluation of RecSys algorithms: **yes!** [6]

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

Solving the Wrong Problem: do the Classical Offline Metrics Evaluate the Quality of Recommendation?

Classical offline metrics:

- Recall@K, Precision@K, HR@K: How often is an item in the top k - no, evaluates next item prediction
- DCG: Are we assigning a high score to an item - no, evaluates next item prediction

Online metrics and their offline simulations:

- A/B Test: i.e. run a randomized control trial live — yes, but expensive + the academic literature has no access to this
- Inverse Propensity Score estimate of click-through rate: - to be explained later. — yes! (although it is often noisy) [7]

Solving the wrong problem: Do the classical offline metrics evaluate the quality of recommendation?

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

I.1.3. Modern Reco: Solving the Problems of Classical Approach

Aligning the Recommendation Objective with the Business Objectives

- 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, ...)

Better Offline Evaluation

Offline Evaluation that Predicts an A/B Test Result

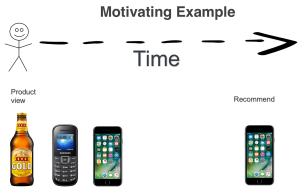
Let's follow the previous notation (A action/recommended item, V organic item view/interaction) and denote π the recommendation policy (that assign probabilities to items conditionally on user past).

Offline Evaluation that Predicts an A/B Test Result

Let's follow the previous notation (A action/recommended item, V organic item view/interaction) and denote π the recommendation policy (that assign probabilities to items conditionally on user past). Imagine we have a new recommendation policy $\pi_t(A_n = a_n | V_1 = v_1, ..., V_n = v_{n-1})$, can we predict how well it will perform if we deploy it? We collected logs from a different policy: $\pi_0(A_n = a_n | V_1 = v_1, ..., V_n = v_{n-1})$

Let's examine some hypothetical logs...

Offline Evaluation via IPS



Imagine the user clicks

$$\pi_t(A_4 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone B}) = 1$$
$$\pi_0(A_4 = \text{phone B} | V_1 = \text{beer}, V_2 = \text{phone A}, V_3 = \text{phone B}) = 0.8$$

The new policy will recommend "phone B" the $\frac{1}{0.8} = 1.25\times$ as often as the old policy.

Offline Evaluation via IPS

Imagine the user clicks

$$\pi_t(A_2 = \text{rice} | V_1 = \text{couscous}) = 1$$
$$\pi_0(A_2 = \text{rice} | V_1 = \text{couscous}) = 0.01$$

The new policy will recommend "rice" the $\frac{1}{0.01} = 100\times$ as often as the old policy.

Offline Evaluation IPS

Let's denote c_n the feedback on the n^{th} recommendation, i.e. 1 if the recommendation got clicked else 0.

Offline Evaluation IPS

Let's denote c_n the feedback on the n^{th} recommendation, i.e. 1 if the recommendation got clicked else 0.

Let's re-weight each click by *how much more likely the recommendation will appear under the new policy π_t compared the logging policy π_0)*

Offline Evaluation IPS

Let's denote c_n the feedback on the n^{th} recommendation, i.e. 1 if the recommendation got clicked else 0.
Let's re-weight each click by *how much more likely the recommendation will appear under the new policy π_t compared the logging policy π_0)*

We will assume that $\boldsymbol{X} = f(v_1, ..., v_n)$. Crafting $f(\cdot)$ is often called "feature engineering".

$$\begin{aligned}\text{CTR estimate} &= \mathbb{E}_{\pi_0}\left[\frac{c \cdot \pi_t(\boldsymbol{a}|\boldsymbol{X})}{\pi_0(\boldsymbol{a}|\boldsymbol{X})}\right] \\ &\approx \frac{1}{N}\sum_{n}^{N}\frac{c_n \pi_t\left(\boldsymbol{a}|\boldsymbol{X}\right)}{\pi_0\left(\boldsymbol{a}_n|\boldsymbol{X}\right)}\end{aligned}$$

Offline Evaluation via IPS

This estimator is unbiased

$$\text{CTR estimate} = \mathbb{E}_{\pi_0}\left[\frac{c \cdot \pi_t(\boldsymbol{a}|\boldsymbol{X})}{\pi_0(\boldsymbol{a}|\boldsymbol{X})}\right] = \mathbb{E}_{\pi_t}[c]$$

Offline Evaluation via IPS

Shortcomings:

- We only look at the clicks i.e. $c_n = 1$ (otherwise the contribution is 0).
- When the new policy differs markedly from the old, the weights become very high (to compensate for the fact that these are rare examples in your sample). These rare high values contribute a lot to the variance of the estimator.

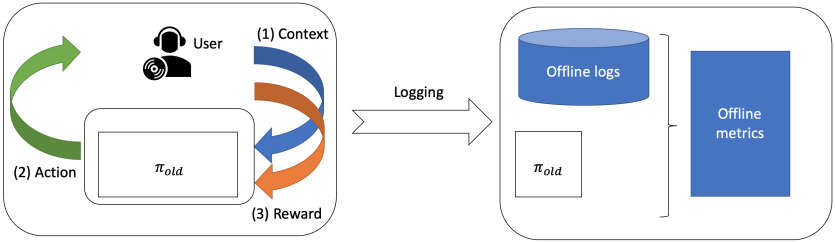
Offline Evaluation via IPS

Overall, IPS actually attempts to answer a counterfactual question (what would have happened if instead of π_0 we would have used π_t ?), but it often has less than spectacular results because of its variance issues.

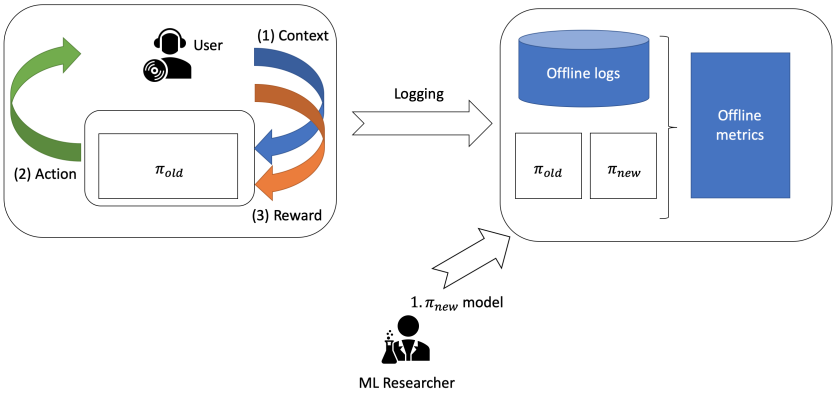
A lot of the shortcomings of the naive IPS estimator are handled in [8].

The Experimentation Life-Cycle of Real-World Recommendation Systems

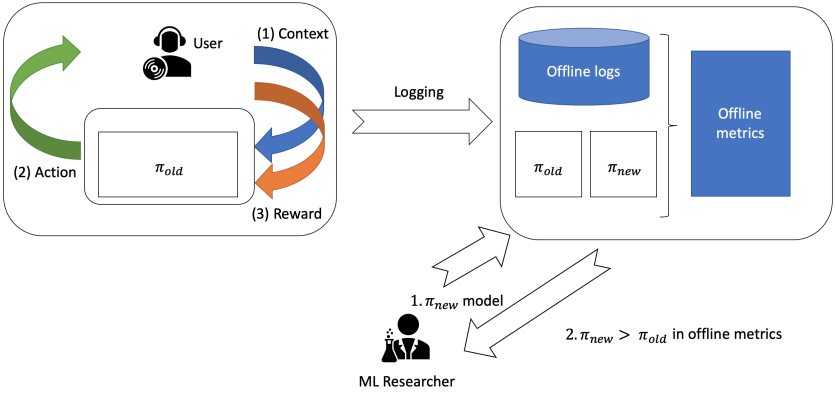
Recommendation as Supervised Learning and A/B Testing



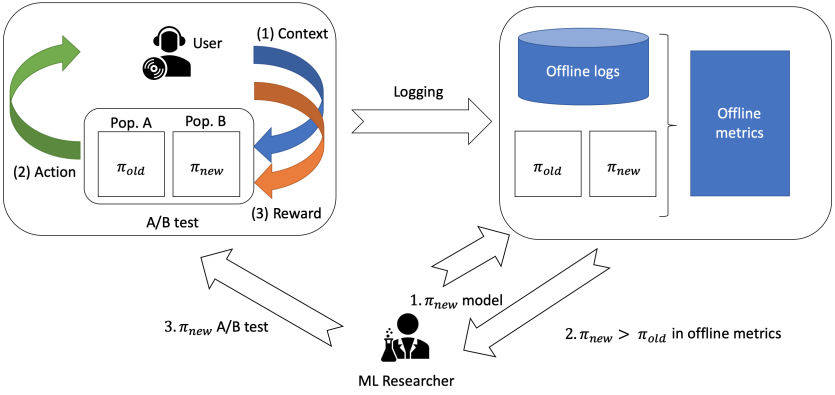
Recommendation as Supervised Learning and A/B Testing



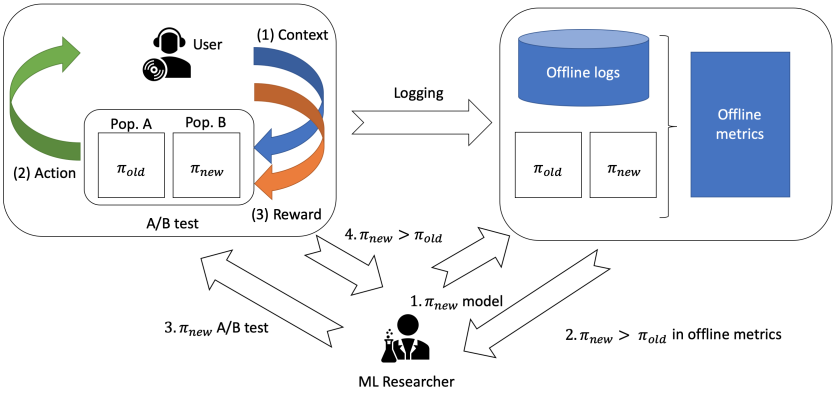
Recommendation as Supervised Learning and A/B Testing



Recommendation as Supervised Learning and A/B Testing



Recommendation as Supervised Learning and A/B Testing



Let's Take a Step Back...

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

- Learn a supervised model from past user activity
- Evaluate offline and decide whether 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

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 offline a reinforcement learning algorithm?

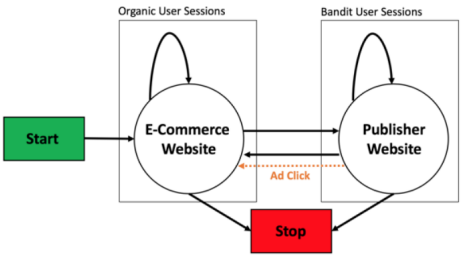
1.4. RecoGym: an Offline Simulator for Recommendation

OpenAI Gym

- **Problem:** The reinforcement learning community lacked a common set of tools to allow comparison of reinforcement learning algorithms.
- **OpenAI Gym:** Software standard (Python API) that allows comparions of the performance of reinforcement learning algorithms
- **Core idea:** Environments define the problem and the agents (the RL algorithms) try to solve it by repeated interactions.

Introducing RecoGym

RecoGym: An OpenAI compatible environment that simulates user behavior (both organic and bandit, e.g. its reaction to recommendation agents)



Introducing RecoGym

- Allows online evaluation of new recommendation policies in a simulated environment
- Holistic view of user (organic and bandit), it provides a framework for categorizing recommendation algorithms
- Key feature: contains an adjustable parameters that allows varying the effectiveness of *pure organic*, e.g. next item prediction algorithms

The Simulation Model for Organic and Bandit Feedback

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 $\beta \in \mathcal{R}^{P\times K}$ bandit embeddings, μ_β non-personalised ctr contribution $f(\cdot)$ montonic increasing function accounting for ad fatigue 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
  $\boldsymbol{\omega}_{u,0} \sim \mathcal{N}(0_{K\times1}, I_K)$
  $v_{u,0} \sim$ Categorical(softmax($\Gamma\boldsymbol{\omega}_{u,0}$))
  while $z_{u,t} \neq$ *stop* **do**
    $t \leftarrow t + 1$
    $\boldsymbol{\omega}_{u,t} \sim \mathcal{N}(\boldsymbol{\omega}_{u,t-1}, \sigma_\omega^2 I_K)$
    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($\Gamma\boldsymbol{\omega}_{u,t} + \mu_\Gamma$))
      $r_{u,t} \leftarrow$ undef
      $c_{u,t} \leftarrow$ undef
    end
    if $z_{u,t}$ = *bandit* **then**
      $r_{u,t}$ is geneated from the policy
      $c_{u,t} \sim$ Bernoulli($[f(\beta\boldsymbol{\omega}_{u,t} + \mu_\beta)]_{r_{u,t}}$)
    end
  end
end

Let's Get Started

An introductory notebook to RecoGym can be found: **click here!**
We will be using Google Colab, so you have nothing to install!

Exercise no. I.1: Organic Best Of vs. Bandit Best Of

Exercise - The best of the best

Go to the notebook *1. Organic vs Bandit Best Ofs* **(click here)**

- Examine the logs; to start with, we will look at non-personalized behavior only.
- Plot a histogram of organic product popularity. What is the most popular product?
- Plot the (non-personalized) 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 proxy popularity for click-through rate?
- Simulate an A/B 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.

A quick recap of the differences between classic and modern reco

Classic vs. Modern Reco

	Classic Reco	**Modern Reco**
Approach	Autocomplete	Intervention
Feedback	Organic	Bandit
Target	Product id	Click / no click
Loss	Softmax	Sigmoid
Metrics	Offline ranking and regression	Online A/B Test / Offline simulation of A/B Test
Offline evaluation setup	Datasets	Propensity datasets / Simulators
Literature	RecSys	ComputationalAdvertising/ContextualBandits/RL
RealWorld deployment	Initial Solution	Advanced Solution

I.2. Recommendation Reward Modeling via (Point Estimation) Maximum Likelihood Models

Maximum Likelihood-based Agent

How can we build an agent/recommender from historical recommendation data i.e. $(\boldsymbol{X}_n, \boldsymbol{a}_n, c_n)$?

Maximum Likelihood-based Agent

How can we build an agent/recommender from historical recommendation data i.e. $(\boldsymbol{X}_n, \boldsymbol{a}_n, c_n)$?

The most straight-forward way is to first frame the task as a reward/click prediction task.

Maximum Likelihood-based Agent

How can we build an agent/recommender from historical recommendation data i.e. $(\boldsymbol{X}_n, \boldsymbol{a}_n, c_n)$?

The most straight-forward way is to first frame the task as a reward/click prediction task.

The classical solution:

Maximum Likelihood Linear Model = **Logistic Regression**

Maximum Likelihood-based Agent

How can we build an agent/recommender from historical recommendation data i.e. $(\boldsymbol{X}_n, \boldsymbol{a}_n, c_n)$?

The most straight-forward way is to first frame the task as a reward/click prediction task.

The classical solution:

Maximum Likelihood Linear Model = **Logistic Regression**

Point estimation = Estimates the best fit solution only = Non-Bayesian Model

Reward(Click) Modeling via Logistic Regression

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

where

- $\boldsymbol{X}$ represent context or user features
- $\boldsymbol{a}$ represent action or product features
- $\boldsymbol{\beta}$ 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 $\boldsymbol{a}_n$.

Reward(Click) Modeling via Logistic Regression

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

where

- $\boldsymbol{X}$ represent context or user features
- $\boldsymbol{a}$ represent action or product features
- $\boldsymbol{\beta}$ 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?*

Modeling interaction between user/context and product/action

- $\Phi\left([\boldsymbol{X}_n\ \boldsymbol{a}_n]\right) = \boldsymbol{X}_n \otimes \boldsymbol{a}_n$ Kronecker product, this is somewhat what we will be doing in the example notebook, these will model pairwise interactions between user and product features.
- $\Phi\left([\boldsymbol{X}_n\ \boldsymbol{a}_n]\right) = \nu(\boldsymbol{X}_n) \cdot \mu(\boldsymbol{a}_n)$ Another more "modern" approach, is to build an embedded representation of the user and product features and consider the dot product between both (in this case the parameters of the model are carried in the embedding functions ν and μ and we have no parameter vector $\boldsymbol{\beta}$)

Reward(Click) Modeling via Logistic Regression

Then it's just a classification problem, right? We want to maximize the log-likelihood:

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

Other Possible Tasks

Imagine we recommend a group of items (a banner with products, for instance), and one of them gets clicked, an appropriate task would be to rank the clicked items higher than non clicked items.

Example: Pairwise Ranking Loss

$$\sum_n \log \sigma \left(\Phi \left([\boldsymbol{X}_n \ \boldsymbol{a}_n] \right)^T \beta - \Phi \left(\left[\boldsymbol{X}_n \ \boldsymbol{a'}_n \right] \right)^T \beta \right)$$

Where $\boldsymbol{a}_n$ represent a clicked item from group n and $\boldsymbol{a'}_n$ a non clicked item.
Multitude of possible other tasks (list-wise ranking, top-1, ...) e.g. BPR [9]

I.2.1. Using Reward Models to Act

Value-based Models for Recommendation

To choose which action to take, we need to evaluate all possible actions and take the one maximizing the predicted reward.

This is a *value-based model* approach to Recommendation, where in order to take the optimal action we:

Value-based Models for Recommendation

To choose which action to take, we need to evaluate all possible actions and take the one maximizing the predicted reward.

This is a *value-based model* approach to Recommendation, where in order to take the optimal action we:

- **Step1:** Build a value model that can estimate the reward of using any action in the context of any user

Value-based Models for Recommendation

To choose which action to take, we need to evaluate all possible actions and take the one maximizing the predicted reward.

This is a *value-based model* approach to Recommendation, where in order to take the optimal action we:

- **Step1:** Build a value model that can estimate the reward of using any action in the context of any user
- **Step2:** At decision time, evaluate all possible actions given the user context

Max.Likelihood Model $\rightarrow$ Recommendation Policy

In the LR case, our max likelihood click predictor can be converted into a (deterministic) policy by just taking the action with the highest predicted click probability.

Sidepoint: Scaling *argmax* in the Case of Large Action Spaces

- Imagine we mapped the context/user features $\boldsymbol{X}$ into a dense representation (embedding) $\boldsymbol{u}$
- Same thing for the action/product features $\boldsymbol{a}$ into $\boldsymbol{v}$
- Such that $P(c = 1|\boldsymbol{X}, \boldsymbol{a}) = \sigma(\boldsymbol{u} \cdot \boldsymbol{v})$
- Choosing for a given context the reward maximizing action is a *Maximum InnerProduct Search (MIPS)* problem, which is a very well studies problem for which approximate solutions have a high quality

Sidepoint: Scaling *argmax* in the Case of Large Action Spaces

- Several method families
  - Hashing based: Locality Sensitive Hashing [10]
  - Tree based: Random Projection Trees [11]
  - Graph based: Navigable Small World Graphs [12]
- Bottom line: use and abuse them !

Exercise no. I.2: Pure Bandit Model

Pure Bandit Feedback

- Build a model using feature engineering using logistic regression (or a deep model)

Pure Bandit Feedback

- Build a model using feature engineering using logistic regression (or a deep model)
- Finally you evaluate your performance against production.

Pure Bandit Feedback

- Build a model using feature engineering using logistic regression (or a deep model)
- Finally you evaluate your performance against production.

Please look at notebook "2. Likelihood Agent.ipynb" Click **here!**

I.3. Shortcomings of Classical Value-based Recommendations

I.3.1. Covariate Shift Problem

Issue #1: The Covariate Shift Problem

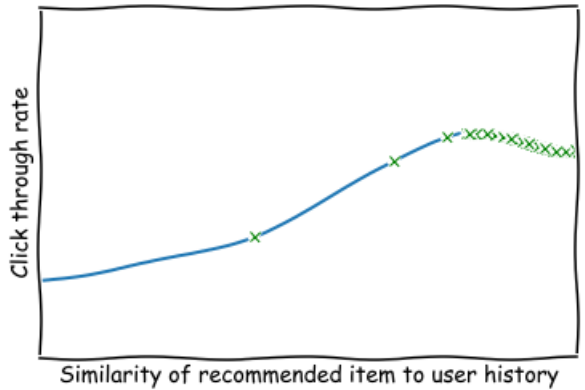
The problem: We train our value model on one distribution of (reward, context,action) tuples, but we test on another!

- Train on: $P(c = 1|\boldsymbol{X}, \boldsymbol{a}) \cdot \pi_0(\boldsymbol{a}|\boldsymbol{X}) \cdot P(\boldsymbol{X})$
- (Online) test on: $P(c = 1|\boldsymbol{X}, \boldsymbol{a}) \cdot \pi_{\text{unif}}(\boldsymbol{a}|\boldsymbol{X}) \cdot P(\boldsymbol{X})$

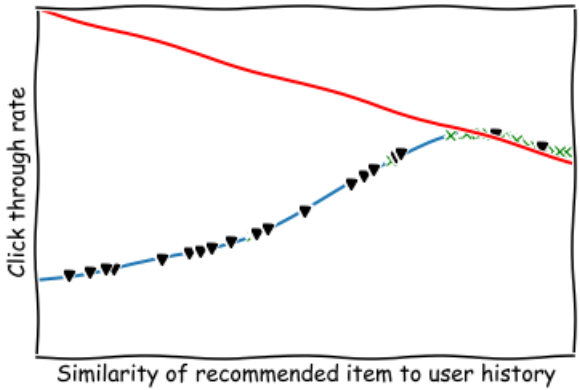
In the case of value based recommendations we are biased at training time (because of the logging policy) but we are always evaluating on the uniform distribution (because of the argmax).

Good recommendations are similar to history, but not too similar.

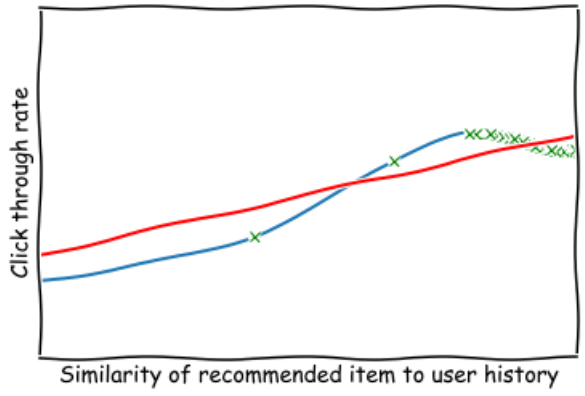
Historical reco tries to be good, it is mostly very similar to history.



When we evaluate new actions we do this uniformly. The model underfits badly where most of the data is.



A re-weighting scheme makes the underfit uniform when we evaluate.



Covariate Shift and Model Underfitting

- If we have a perfect predictor for $P(c = 1|\boldsymbol{X}, \boldsymbol{a})$, why should we care if $\pi(\boldsymbol{a}|\boldsymbol{X})$ changes?
- Of course, $P(c = 1|\boldsymbol{X}, \boldsymbol{a})$ is not perfect and it is usually designed to underfit (to insure generalization through capacity constraints - limited degrees of freedom)
- In this case, we should allocate the model capacity on the portions of the $\pi(\boldsymbol{a}|\boldsymbol{X})$ where the model will be tested

Covariate Shift and Recommendation

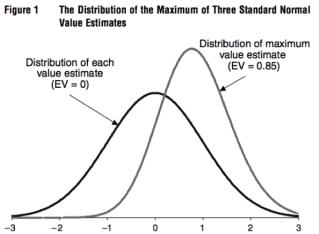
- **Note:** In the case of using value-based models for decision making (and in our case, Recommendation) we will evaluate all actions for all contexts so we will need to be good everywhere!

In Part 2, we will see how we are going to solve this, but first let's talk about the second issue!

I.3.2. The Optimizer's Curse

Issue #2: Optimizer's Curse: a Story

The **actual value** of choosing the action with the best predicted reward out of k alternatives **will usually be *worse* than its predicted value**, even if your estimate was unbiased!



The distribution of rewards of argmax of 3 identical actions

From: « The Optimizers Curse: Skepticism and Postdecision Surprise in Decision Analysis »

Issue #2: Optimizer's Curse: a Story

Day1: Offline Evaluation Time (Predicted Reward)

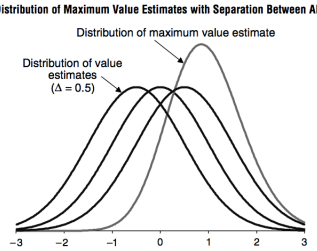
- Imagine a decision problem, handed to us by our CEO, where we need to choose between **3 actions** and where, for each one of them, we have samples of historical rewards.
- In order to choose, we build empirical reward estimators for each of the 3 actions, we find the one that has the best historical performance and we go back to our CEO with the recommendation! After work, we congratulate ourselves and go to sleep happy!

Issue #2: Optimizer's Curse: a Story

Day2: Online Evaluation Time (True Reward)

- However, the next day we meet with the CEO only to find out this was just a test of our decision-making capabilities, and that all of the sampled rewards for the 3 actions were **all drawn independently from the same distribution**.
- Due to the finite nature of our samples, each one of our 3 reward estimators made errors, some of them more negative (pessimistic), and some of them more positive (optimistic).
- Because when we decided, we selected the action with the highest estimate, we obviously favored the most optimistic of the 3, making us believe we optimized our decision, when we were just overfitting the historical returns. This effect is known at the **Optimizer's Curse**.

Issue #2: The Optimizer's Curse. A Story



Δ	Expected disappointment	Probability of correct choice
0.0	0.85	0.33
0.2	0.66	0.42
0.4	0.51	0.50
0.6	0.39	0.59
0.8	0.30	0.66
1.0	0.22	0.73
1.2	0.17	0.78
1.4	0.12	0.83
1.6	0.10	0.87
1.8	0.07	0.90
2.0	0.05	0.92
2.2	0.03	0.94
2.4	0.02	0.95
2.6	0.01	0.97
2.8	0.01	0.98
3.0	0.00	0.98

Figure 1: The distribution of rewards of *argmax* of 3 actions with Δ separation

Optimizer's Curse in Recommendation

- The *Optimizer's Curse* mainly appears when the **reward estimates are based on small sample sizes** and **the deltas between the true mean rewards** of the different actions **are relatively small** such that the variance of the empirical estimators can lead to overoptimistic expectations or even erroneous ranking of actions.
- **This is not a corner case!** In real recommendation applications, the space of possible context-action pairs is extremely large, so there are always areas of the distribution where the current system is deciding based on very low sample sizes.

Fixing Covariate Shift and Optimizer's Curse with Policy Learning

We saw that value-based approaches for decision-making suffer from two issues **when using ERM-based models**.

In Part II. we will show how to solve both issues !

Example 3. Showcasing the Optimizer's Curse effect in our likelihood agent

Demo: Expected CTR vs. Observed CTR

We will be using our likelihood model to showcase optimizer's curse, where we overstimate the number of clicks we will be getting from showing particular products. The notebook is "3 Optimizer's curse"

Wrapping-up Part I. of our course

What have we learnt so far?

- How the classical recommendation setting differs from real world recommendation
- Using a simulator aka RecoGym to build and evaluate recommender agents
- Shortcomings of value-based models with maximum likelihood estimators (e.g. LR) (covariate shift and optimizer's curse)

About Part II.

- Frame the problem in the contextual bandit vocabulary
- We will introduce an alternative approach to value-based models to solve the covariate shift issue
- We will define a theoretical framework to help reason about Optimizer's Curse and solve the issue

Thank You!

Questions?

References

[1] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

[2] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA., 2007.

[3] David Ben-Shimon, Alexander Tsikinovsky, Michael Friedmann, Bracha Shapira, Lior Rokach, and Johannes Hoerle. Recsys challenge 2015 and the yoochoose dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 357–358. ACM, 2015.

References ii

[4] Roberto Turrin, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 30music listening and playlists dataset. In *RecSys 2015 Poster Proceedings*, 2015. Dataset: `http://recsys.deib.polimi.it/datasets/`.

[5] Yahoo. Yahoo news feed dataset, 2016.

[6] Damien Lefortier, Adith Swaminathan, Xiaotao Gu, Thorsten Joachims, and Maarten de Rijke. Large-scale validation of counterfactual learning methods: A test-bed. *arXiv preprint arXiv:1612.00367*, 2016.

[7] Daniel G Horvitz and Donovan J Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685, 1952.

References iii

[8] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. Offline a/b testing for recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 198–206. ACM, 2018.

[9] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[10] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[11] erikbern. Approximate nearest neighbors in c++/python optimized for memory usage and loading/saving to disk, 2019.

[12] Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.