

# **Experimental optimization**

## **Lecture 10: Contextual bandits**

**David Sweet**

# Review

## Response surface methodology

- Reduce experimentation cost by specializing
- Reduced cost means fewer measurements
- RSM specializes to continuous parameters
- RSM can optimize 1-5 parameters

# Review

## Predictor-in-controller

- Predictor: Estimates a target, ex.,  $P\{\text{click}\}$  on an ad
- Controller: Uses predictions to make a decision / choose an action
  - ex., “Of the 1000 ads available, show the one with the highest  $P\{\text{click}\}$ ”

# Industrial engineered systems

## Predictors in controllers

Controller	Prediction	Action	Reward
Ad server	$P\{\text{click}\}$	Show ad with highest $P\{\text{click}\}$	CPC revenue
Fraud detector	$P\{\text{fraudulent}\}$	Hold charges with high $P\{\text{fraudulent}\}$ until customer gives OK	Avoid losing money to fraud
Trading strategy	$E[\text{return}]$	Buy when $E[\text{return}] > 0$ , sell when $E[\text{return}] < 0$	Revenue ("PnL")
Social media feed	$P\{\text{like}\}$	Show posts with highest $P\{\text{like}\}$	Users spend more time on feed

# Contextual bandits

## Specialization

- Can optimize many — millions — of parameters by specializing
- ...to short-term business metrics, aka *rewards*
  - Ex: CTR, likes, fraudulent transaction
  - But **not**: DAU, daily pnl, purchase following ad, time spent per day
- ...and predictor-in-controller designs
  - The predictive model contains all of the CB-tunable parameters
  - Each reward corresponds to a single prediction

# Production logs

- Every time you show an ad, log
  - features of the ad
  - features of the user
  - whether the user clicked
- data =  $\{(x_i, y_i)\}$ , where
  - $x_i$  = all of the features
  - $y_i$  = 1 if clicked, else 0; “click indicator”

# Typical design

## Estimate $P\{\text{click}\}$

- Fit an SL model to the data, like
  - logistic regression
  - neural network
- Model may have many parameters
- Model estimates  $P\{\text{click}\} = \text{click-through-rate} = \text{CTR}$
- More specifically:  $P\{\text{click} \mid \text{ad \& user features}\} = \text{CTR for ad}$

# Typical design

## Periodic refitting

- Fit model every day
- Use data from trailing month's logs
- Production uses latest, refit model
- Refitting tracks changes in system over time

# Typical design

## Problem: Missing counterfactuals

- $P\{\text{click} \mid \text{Ad1}\} = .02$ ,  $P\{\text{click} \mid \text{Ad2}\} = .01$
- Controller chose to show Ad1
- Features and click indicator for Ad1 show up log
- No data collected for Ad2
- BUT: The model is wrong about  $P\{\text{click} \mid \text{Ad2}\}$ .
  - The CTR for Ad2 is actually much higher, .03.

# Typical design

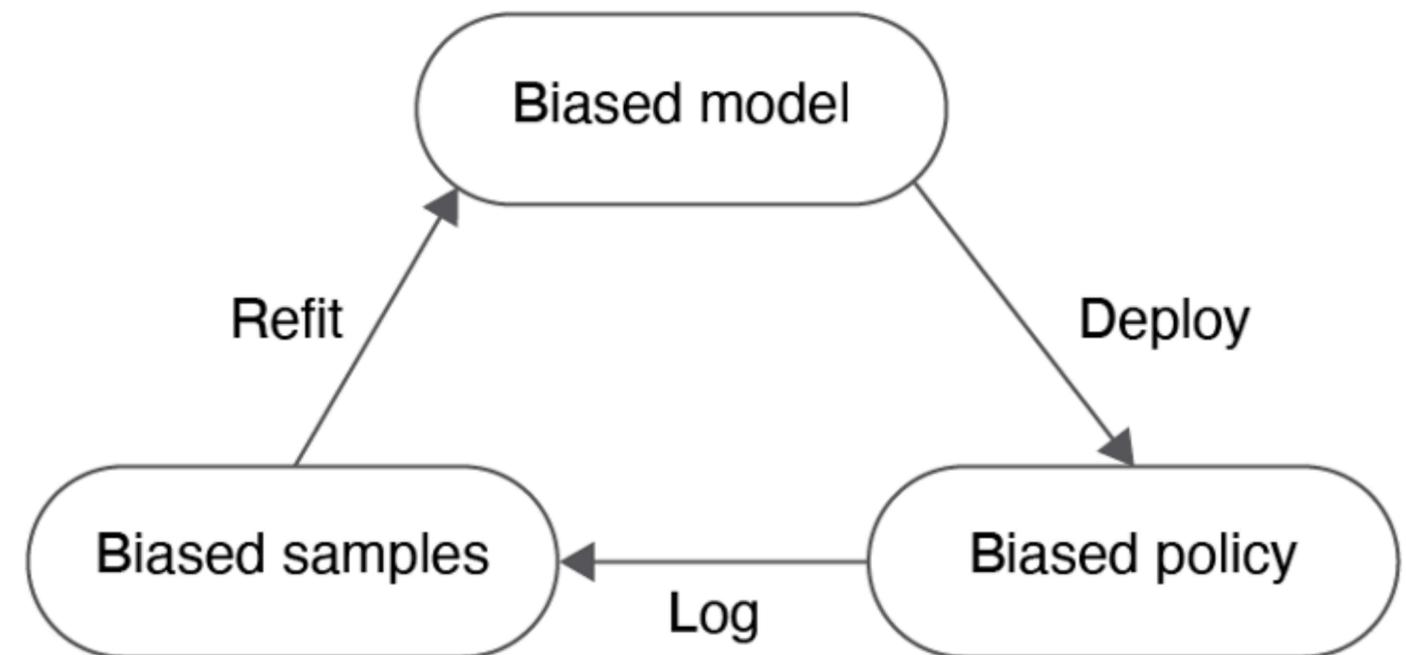
## Problem: Missing counterfactuals

- If we don't show Ad2, then
  - We won't log any data for Ad2
  - The model will fit from the same old data
  - The model will still estimate  $P\{\text{click} \mid \text{Ad2}\} = .02$
- Why is the model wrong about Ad2?
  - Small-sample bias
  - All models are biased. Only better data can fix that.

# Typical design

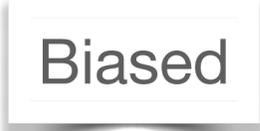
## Problem: Missing counterfactuals

- *counterfactual*: What would have happened if we had done something else?
  - I.e., What would have happened if we had shown Ad2 instead of Ad1?
- Without counterfactual data, the model can't make a better prediction about Ad2.
- Instead, endless, feedback loop



# Solution: Exploration

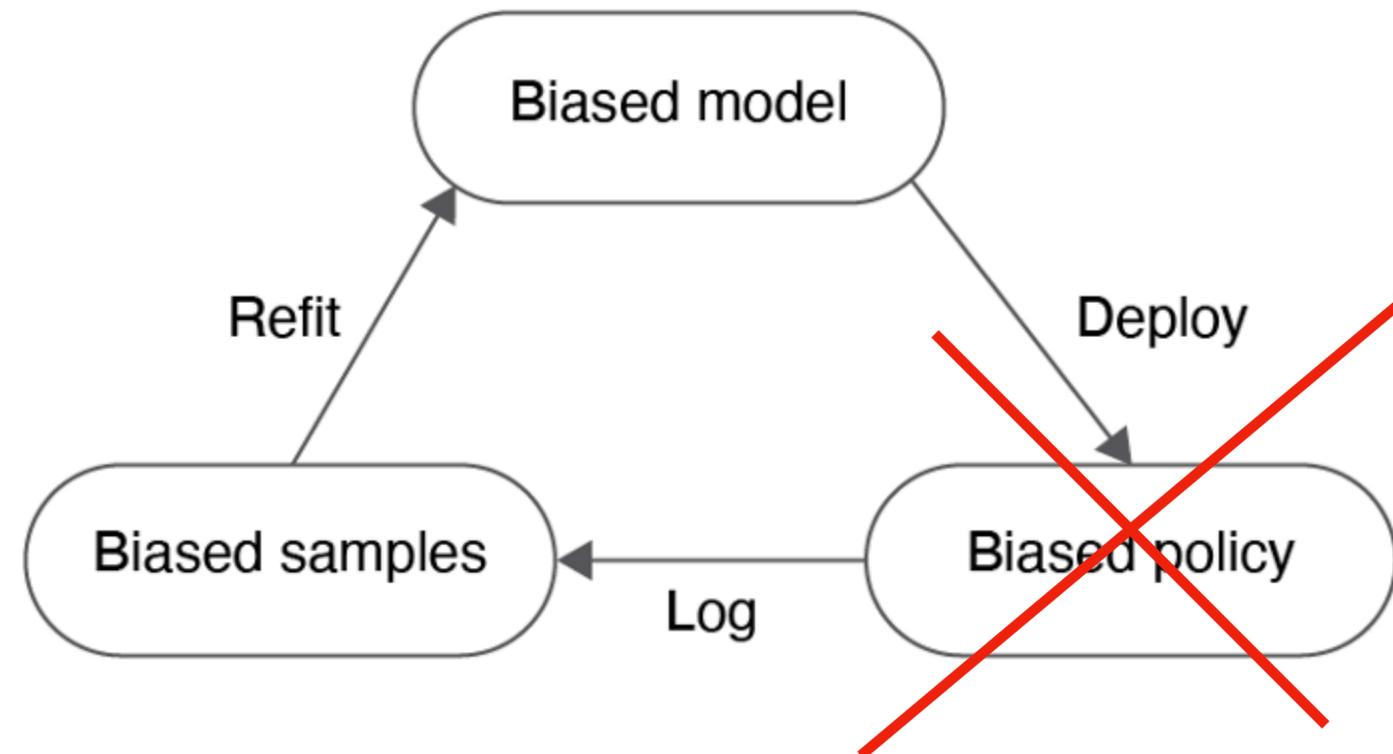
## Collect the counterfactuals

- Each time you show an ad:
  - exploit: w/ probability 0.90, use the model, as before 
  - explore: w/ probability 0.10, show an ad at random 
- Every day, exploration will collect some counterfactuals — ads that wouldn't have been shown by the model
- The counterfactuals show up in the logs, thus in the data
- This debiases the data which debiases the model

# Solution: Exploration

## Collect the counterfactuals

- Think of each ad as an “arm”, then this is epsilon-greedy
- Breaks the feedback loop
- You’re running an experiment to optimize the parameters of the predictor’s model



# Contextual bandit

## Generalization of multi-armed bandit

- Each ad is an arm
- MAB:  $\mu_{ad}$  = avg. CTR for ad = estimated CTR
- CB:  $P\{\text{click} \mid \text{features of ad and user}\}$  = estimated CTR
- “features of ad and user” called the *context*,
  - hence the name *contextual bandit*
- Set “features = 1”, i.e., just fit an intercept, then CB == MAB

# Policy classes

<p>Naive: Show ad with best CTR so far</p>	<p>MAB: <b>Explore</b> to get unbiased data</p>
<p>P-in-C: <b>Predict</b> CTR from ad id <i>and</i> other features</p>	<p>CB: <b>Explore &amp; predict</b></p>

# Solution 2: Explore models

- Model weights are found by fitting (regression)
- Model weights have errors:
  - ex., linear regression yields betas and standard errors of betas
- Alas, SGD yields NN weights, but no errors
- Given a fitting routine that returns weights, ex., Python function `fit(data)` how could you find the standard errors of the weights?

# Solution 2: Explore models

## Errors in weights

- Bootstrap!
  - Take  $B$  bootstrap samples of the data
  - Run `fit()` on each bootstrap sample to find  $B$  weight vectors
  - Then `std()` of the  $B$  weight vectors is the SE of the weight vector
- Works for any model type, linear regression, logistic regression, NN, random forest, SVM, etc.

# Solution 2: Explore models

## Thompson sampling

- Use bootstrap for Thompson sampling
- Each time you need to show an ad:
  - Take 1 bootstrap sample of data
  - Run a fit to find 1 weight vector
  - Calculate  $P\{\text{click}\}$  using that weight vector
- Each weight vector defines a different model
- $P\{\text{using a model}\} = P\{\text{that model is best}\}$ 
  - “best”: weight vector is closest to the “true” unobservable weight vector

Analogous to: “Run arm  $k$  if  $\tilde{\mu}_k = \max\{\tilde{\mu}_{k'}\}$ ”

# Solution 2: Explore models

## Thompson sampling, in practice

- `fit()` takes too long to run; can't run for every ad
- Instead, offline:
  - Generate  $B$  bootstrap samples (ex.,  $B=10$ )
  - Fit  $B$  models; an *ensemble* of models
- Online, for every ad:
  - Choose 1 model from ensemble, randomly
  - Use that model to choose which ad to show

# Thompson sampling

- Randomizes over / explores models instead of arms
- Optimal regret, unlike epsilon-greedy
- No meta-parameter to tune, unlike epsilon-greedy
  - epsilon-greedy has same decay parameter,  $c$ , when used in CB

# Short-term business metrics only

- If CB can optimize millions of parameters, why bother with, ex., RSM, which can optimize 2 or Bayesian optimization, which can optimize, maybe 10?
- Catch: CB only works with short-term business metrics (rewards)
  - Ex: CTR, likes, fraudulent transaction
  - But **not**: DAU, daily pnl, purchase following ad, time spent per day
- CB needs (features, target) pairings and many samples in data set
  - Ex: DAU is 1 number/day, even though many, many ads shown

# Summary

- Perspective 1: CBs fix feedback loops that bias predictor-in-controller designs by adding exploration
- Perspective 2: CBs improve MAB decisions by conditioning on context, i.e. adding a prediction model
- Contextual bandits use exploration to collect unbiased data
- Thompson sampling explores models (weights), epsilon-greedy explores arms
- Contextual bandits enable optimization of many parameters, but only for short-term business metrics