Multi-Armed Bandits (MAB)

MKTG 6279

Week 5

Today

- What is a multi-armed bandit?
- Epsilon-greedy
- Thompson sampling
- Contextual bandits

Multi-Armed Bandits

What is a multi-armed bandit?

- A situation or problem where you make sequential decisions with multiple options ("arms") with unknown rewards
- The key challenge is determining whether to either...
  - **Explore**: try different arms to learn rewards
  - **Exploit**: collect rewards from the arm you think is best
- MAB algorithms aim to maximize cumulative rewards

Simple example: slot machines

The terminology comes from slot machines in casinos

You can gamble at any slot machine, but the payouts (i.e., the house advantages of each machine) are not known

You start playing at one machine, and get a sense of the payout rate

Do you switch machines or keep playing?

Everyday examples...

You typically solve these every day:

- Try a new restaurant or stick with the usual?
- Watch a different show on Netflix or see how the current series pans out?
- Any relationship

There is a trade-off between sticking with what you know versus trying something new

In marketing: whenever the user experience can be dynamically optimized

Google/Meta Ads Optimization

- Unlike static A/B testing, where traffic is split equally across versions, MAB gradually shifts traffic toward the best-performing ad in real time
- Minimizes lost revenue by automatically optimizing ad exposure instead of waiting until an A/B test ends
- The algorithm tracks click-through rates (CTR) of different ads and adjusts the distribution dynamically

Recommendation engines

- Amazon uses MAB to personalize product recommendations in real-time
- Netflix and Spotify use MAB to suggest content dynamically based on what a user has watched or listened to
- MAB dynamically adjusts recommendations as a user interacts with the platform, optimizing engagement

Recommend items the user is **most likely to like** (based on past behavior) or try **new items** to learn more about the user's preferences?

Three popular MAB algorithms

Method	When to Use
Epsilon-Greedy	Simple to implement, occasional exploration
Thompson Sampling	Uses probability distributions to represent uncertainty
Contextual Bandit	Integrates X when choosing options

Epsilon-Greedy

What is epsilon-greedy?

A simple and widely used MAB algorithm

Epsilon: with probability ϵ (e.g., 10%), *explore* a random action

Greedy: with probability $1 - \epsilon$, *exploit* the best option

A **greedy strategy** always chooses what seems **best now**, without worrying about long-term consequences

Using epsilon-greedy

Why? The "hello world" of MABs:

- Simple and effective
- Explore a little to avoid being stuck
- Dynamic, reduces wasted impressions

When?

- Low user data
- Early-stage testing for benchmark
- Stable best option over time

Example with A/B testing

You work at Instagram and want to figure out which digital ad gets a higher click-through rate (clicks/impressions or CTR)

- Version "A" or version "B"

Run the A/B test for a fixed period, then pick the winner and use for future web traffic

Note: since the MAB alogorithms are dynamic, viewing and simulating the data will be a little different this week

Ground truth data setup

```
set.seed(3)
n = 1000
#mobile and male used later for contextual bandit
mobile   = sample(0:1,n,TRUE)
male     = sample(0:1,n,TRUE)
true_ctr_A = 0.10 - .04*mobile - .02*male
true_ctr_B = 0.04 + .04*mobile + .06*male
clicks_a = rbinom(n, 1, true_ctr_A)
clicks_b = rbinom(n, 1, true_ctr_B)
```

`mobile` and `male` will be ignored until we get to contextual bandits

Notice how $E[A] = .10 - .04 \times .5 - .02 \times .5 = .07$ and $E[B] = .09$

These are the unconditional CTRs from both ads

Plain vanilla A/B test

This is very inefficient code, but easier to see how it compares with MAB algorithms

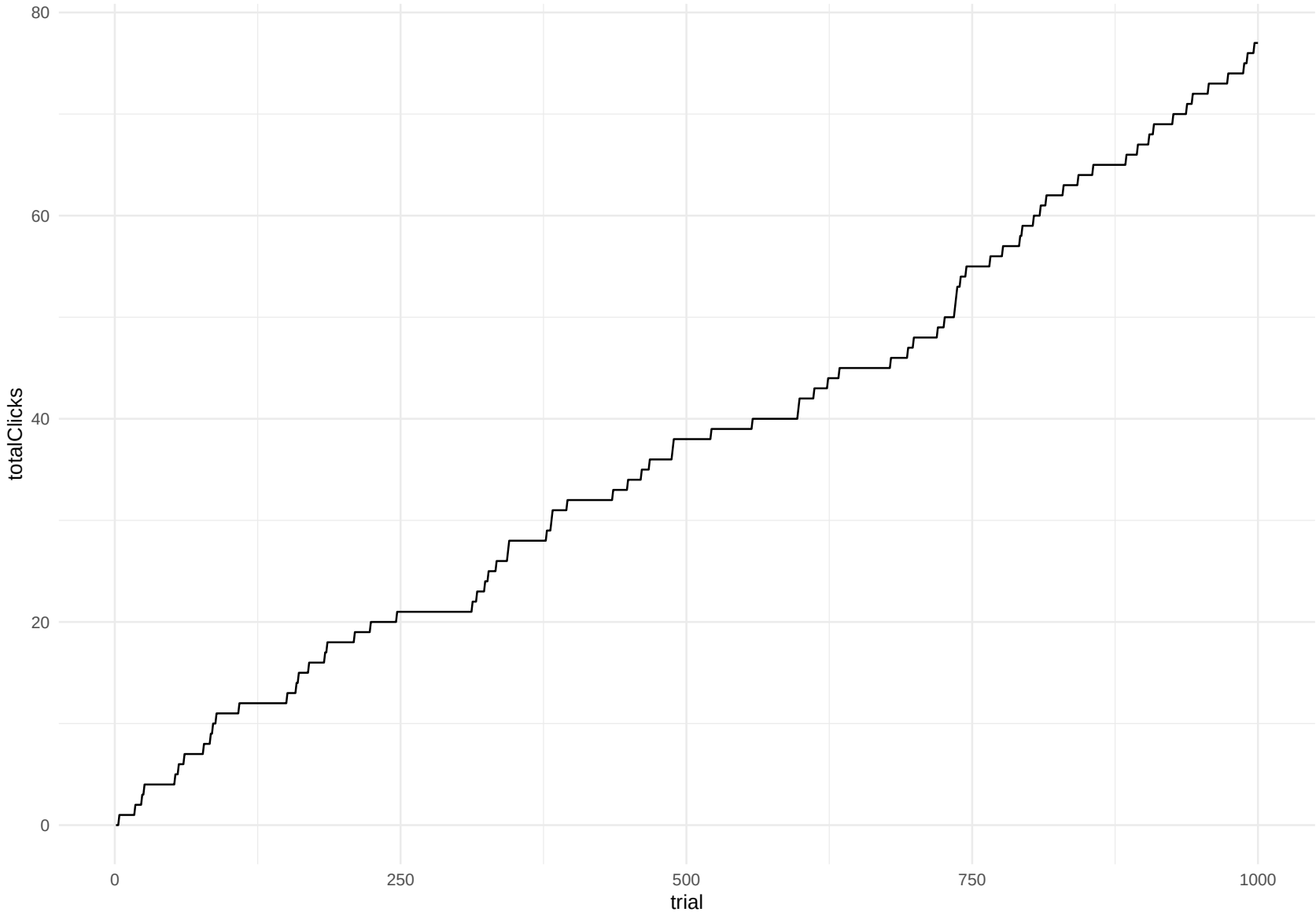
```
ig_ab = data.frame(mab='A/B',trial=1:n,group=NA,
                   click=NA,totalClicks=0)
for(t in 1:n){
  arm_t   = sample(c('A','B'), 1)
  click_t = ifelse(arm_t == 'A',clicks_a[t],clicks_b[t])
  ig_ab$group[t] = arm_t
  ig_ab$click[t] = click_t
  if(t == 1) {
    ig_ab$totalClicks[t] = click_t
  } else {
    ig_ab$totalClicks[t] = ig_ab$totalClicks[t - 1] +
      click_t
  }
}
```

A/B results

```
ig_ab %>%
  group_by(group) %>%
  summarise(click_bar = mean(click))
```

```
## # A tibble: 2 x 2
##   group click_bar
##   <chr>     <dbl>
## 1 A        0.0714
## 2 B        0.0821
```

A/B cumulative clicks



Epsilon-Greedy code

Key modification: exploit with $\Pr(1-\epsilon)$

See `R` for full code

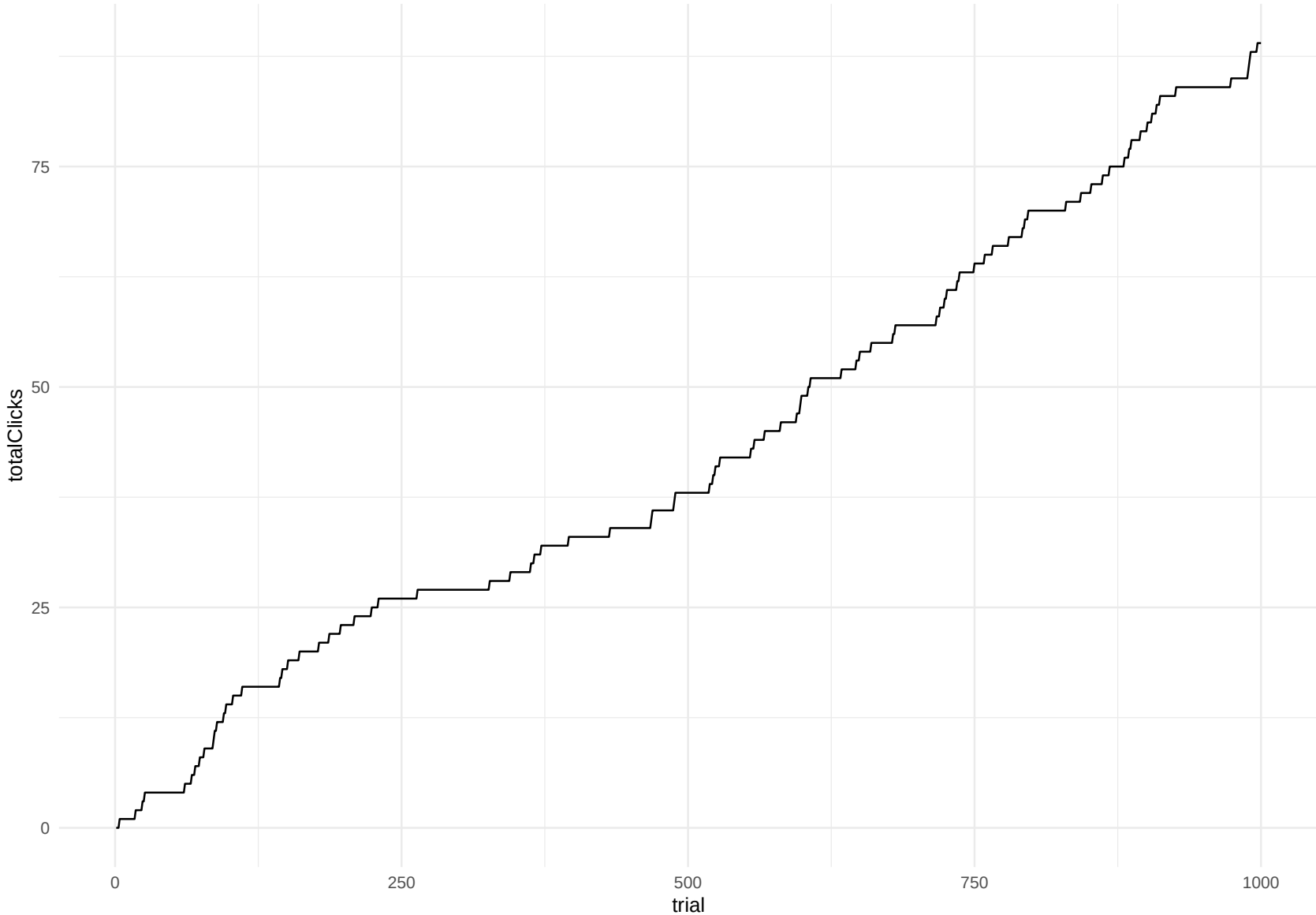
```
if(runif(1) > epsilon) {
  ctr_a = mean(subset(ig_eg,group=='A' & trial < t)$click)
  ctr_b = mean(subset(ig_eg,group=='B' & trial < t)$click)
  arm_t = ifelse(ctr_b > ctr_a, 'B','A')
  }
```

Epsilon-Greedy results

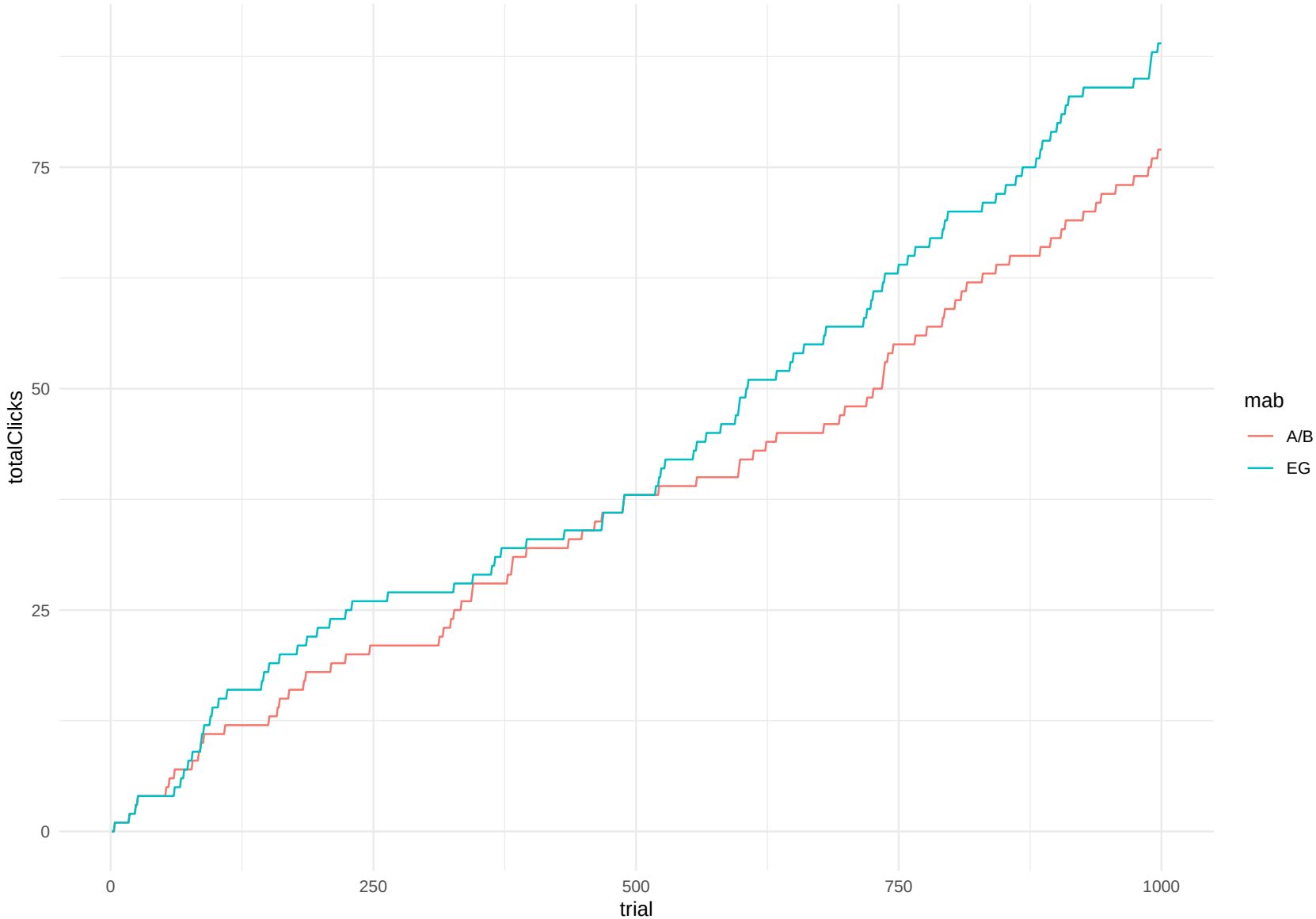
```
ig_eg %>%
  group_by(group) %>%
  summarise(click_bar = mean(click))
```

```
## # A tibble: 2 x 2
##   group click_bar
##   <chr>     <dbl>
## 1 A        0.0779
## 2 B        0.103
```

Epsilon-Greedy cumulative clicks



Epsilon-Greedy vs. A/B



Epsilon-Greedy vs A/B Testing

Benefits of dynamic optimization

- Faster Adaptation: Learns from user behavior and quickly favors better options
- Smarter Traffic Allocation: Adjusts in real time, unlike 50/50 A/B splits
- Higher Reward: Achieves more cumulative clicks $\rightarrow$ more value from same traffic

Thompson Sampling

Thompson sampling

In the epsilon-greedy algorithm, we simply derived the average CTR up through that point to decide which arm to exploit

Thompson sampling is more sophisticated because it puts a **distribution** over each action's reward

This way there is some sense of the uncertainty

We sample from these distributions to distributions to decide which arm to pursue

Beta distributions

In most A/B testing, the outcome is binary

There is either a "success" or "failure"

- Click
- Buy
- Convert

Results are compared using *probabilities* of a success

- "The CTR of version A is 3.5% and version B is 3.8%"

Beta Distrubutions

In a MAB, we can model the success/failure as a Bernoulli trial, where p represents the probability of a success from a single draw from a given arm

We use the **Beta distribution** to quantify the uncertainty in probability p

The Beta distribution is the conjugate prior for the Bernoulli (a special case of the binomial distribution with $n = 1$)

The Beta distribution is defined on the interval [0,1], which makes it ideal for modeling probabilities and proportions

When new outcomes are realized, we can update the beliefs of p

Using the Beta distribution

$$p \sim \text{Beta}(\alpha, \beta)$$

There are two parameters $\alpha > 0$ and $\beta > 0$ which dictate the shape of this distribution

$$E[p] = \frac{\alpha}{\alpha + \beta}$$

This means the expected probability can be represented as a ratio of successes divided by (successes + failures)

Updating posterior beliefs

Suppose midway through an A/B test you have the following for ad A:

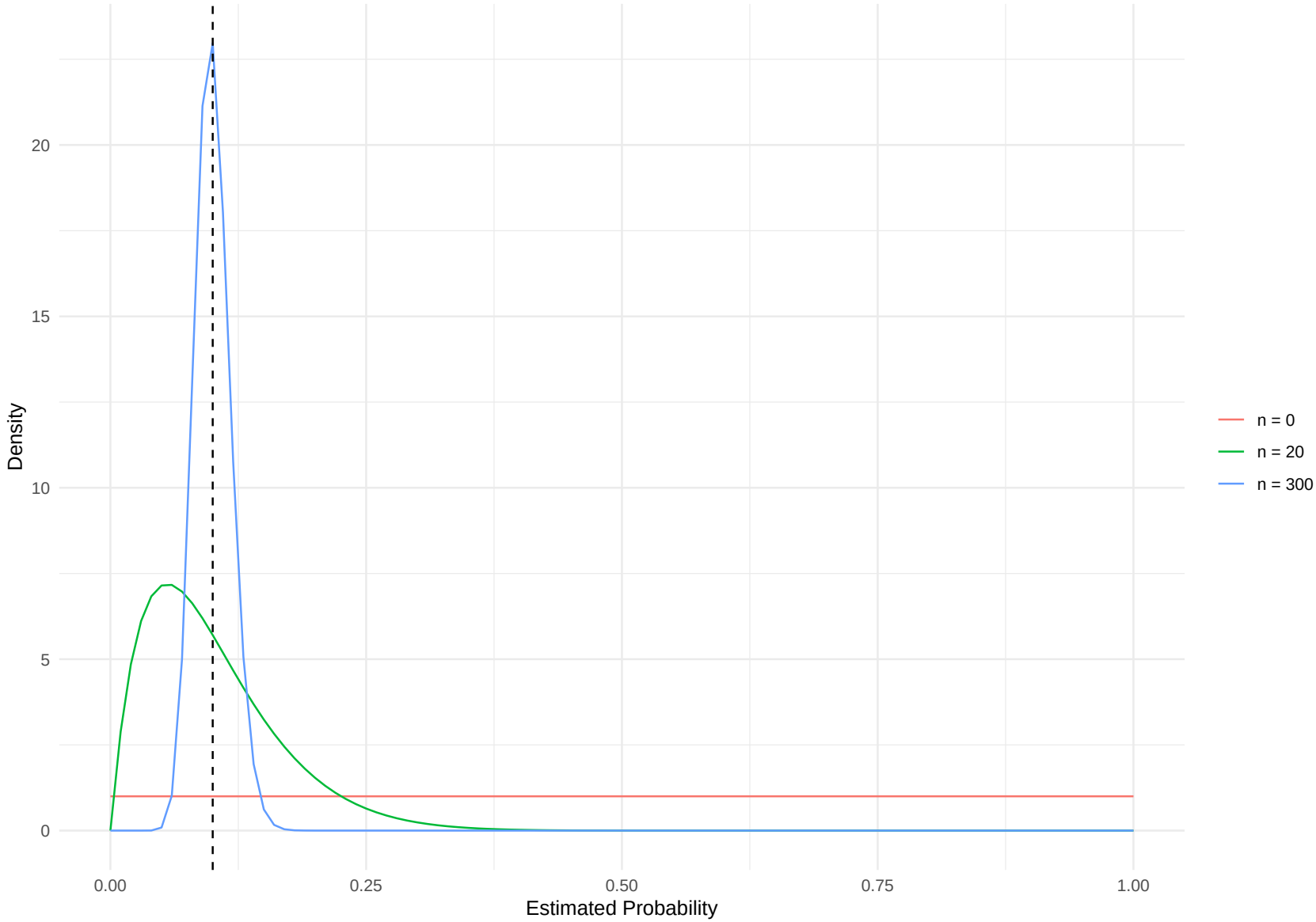
- 10 successes and 90 failures $\rightarrow (\alpha = 10, \beta = 90)$

Ad A will have an expected value of $\frac{10}{10+90} = 10\%$

A new impression is generated and clicked on (success). We can update the posterior as follows: $\alpha = 10 + 1 = 11$

With more impressions, the dispersion around the mean will become tighter

Fro $n > 0$, $E[p] = .1$


20
15
10
5
0
Density
0.00
0.25
0.50
0.75
1.00
Estimated Probability
n = 0
n = 20
n = 300

Thompson Sampling code

Key modification: expected reward (CTR) is drawn from Beta distribution

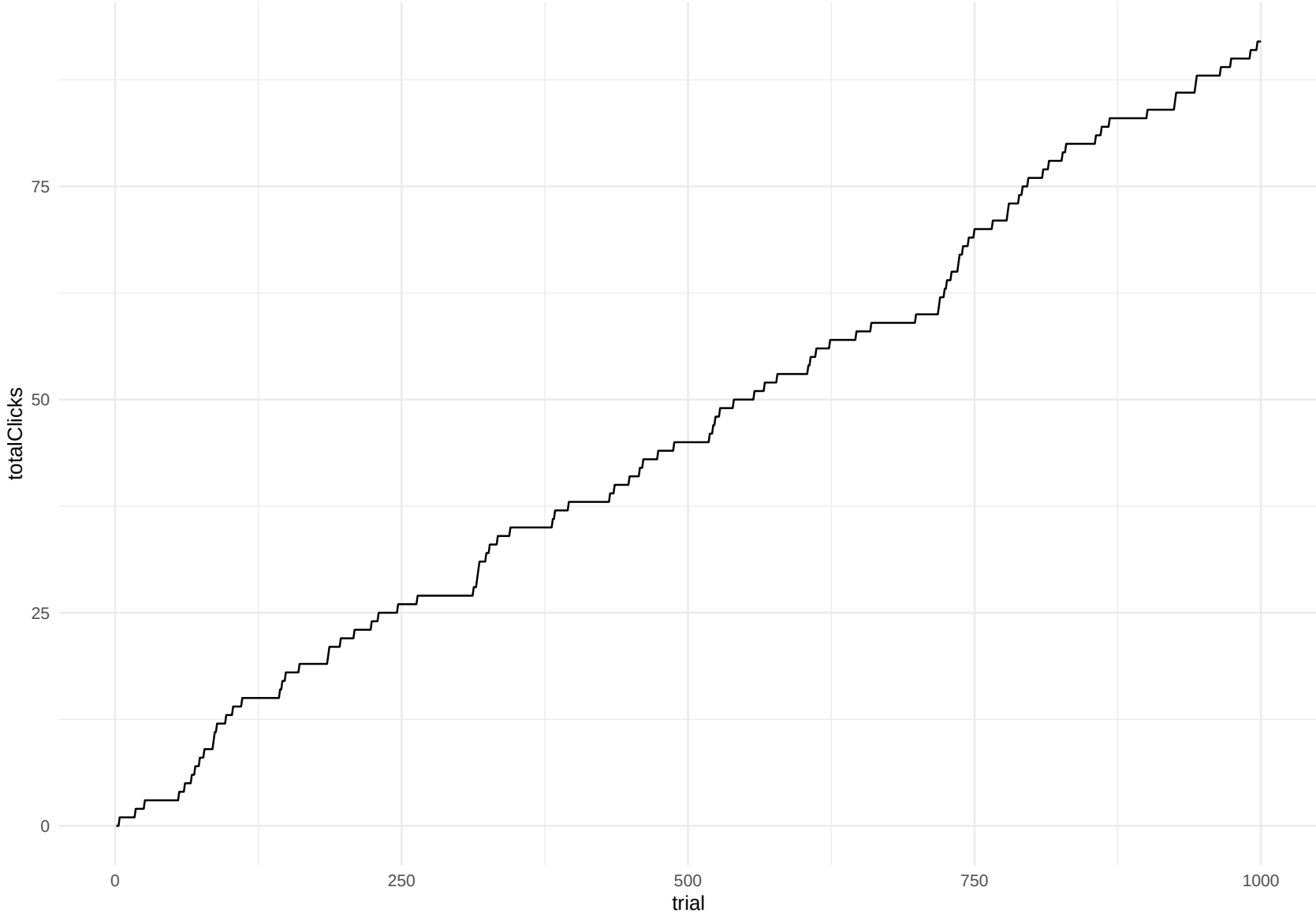
See `R` for full code

```
clicks_a_t  = subset(ig_ts,group == 'A' &
                       trial < t)$click
clicks_b_t  = subset(ig_ts,group == 'B' &
                       trial < t)$click
sample_A    = rbeta(1,sum(clicks_a_t == 1)+1,
                      sum(clicks_a_t == 0)+1)
sample_B    = rbeta(1,sum(clicks_b_t == 1)+1,
                      sum(clicks_b_t == 0)+1)
arm_t       = ifelse(sample_B  >  sample_A,'B','A')
```

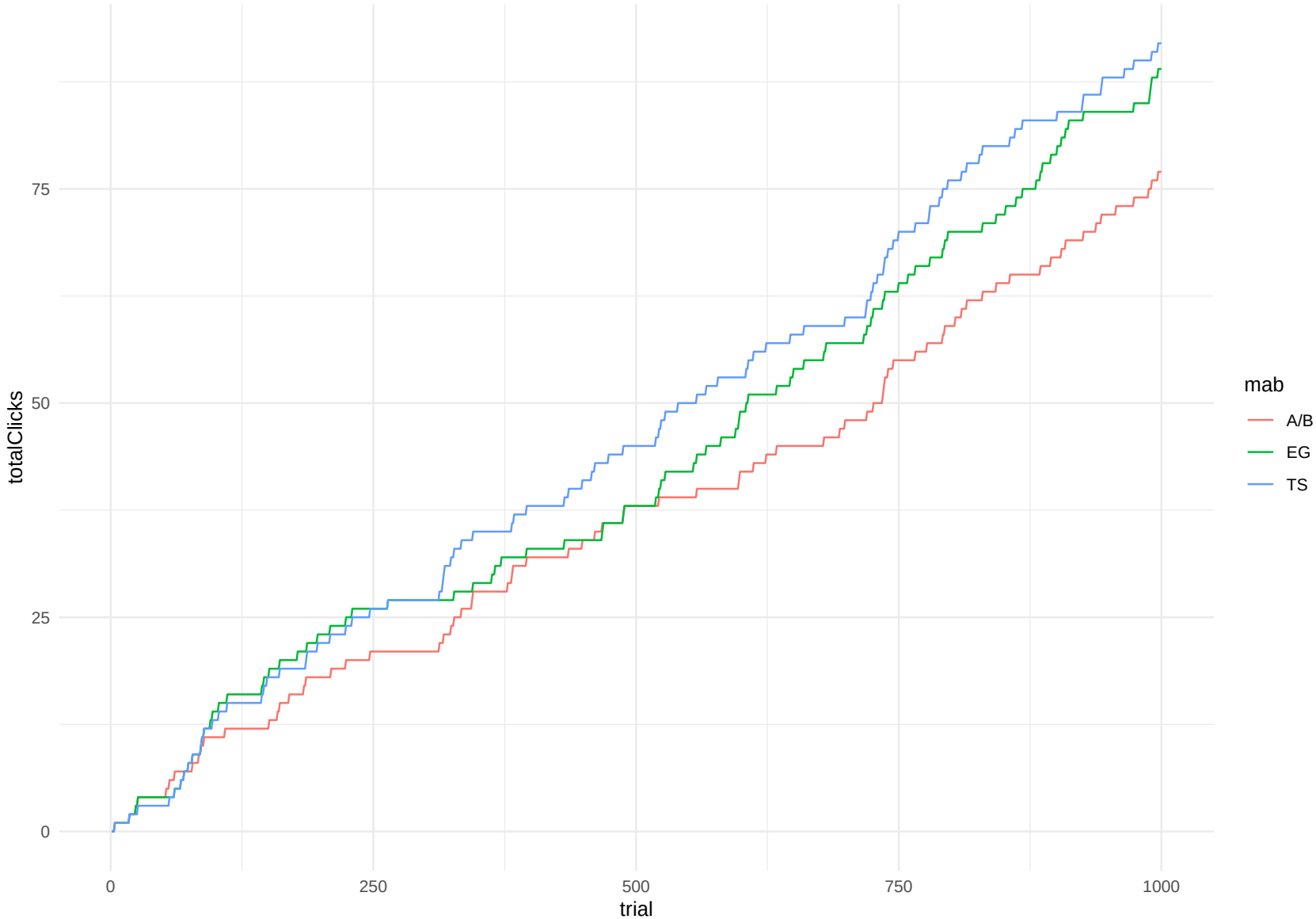
Thompson Sampling results

```
## # A tibble: 2 x 2
##   group click_bar
##   <chr>     <dbl>
## 1 A        0.0952
## 2 B        0.0873
```

Thompson Sampling cumulative clicks



Comparisons so far. . .



Contextual Bandits

What are contextual bandits?

An extension of the standard MAB problem that incorporates **contextual information**

What is contextual information? Any observed data on an indivudal

- Demographics
- Product characteristics
- Historical data
- Basically anything in X

Contextual Bandit code

Key modification: predict clicks conditional on context

See `R` for full code

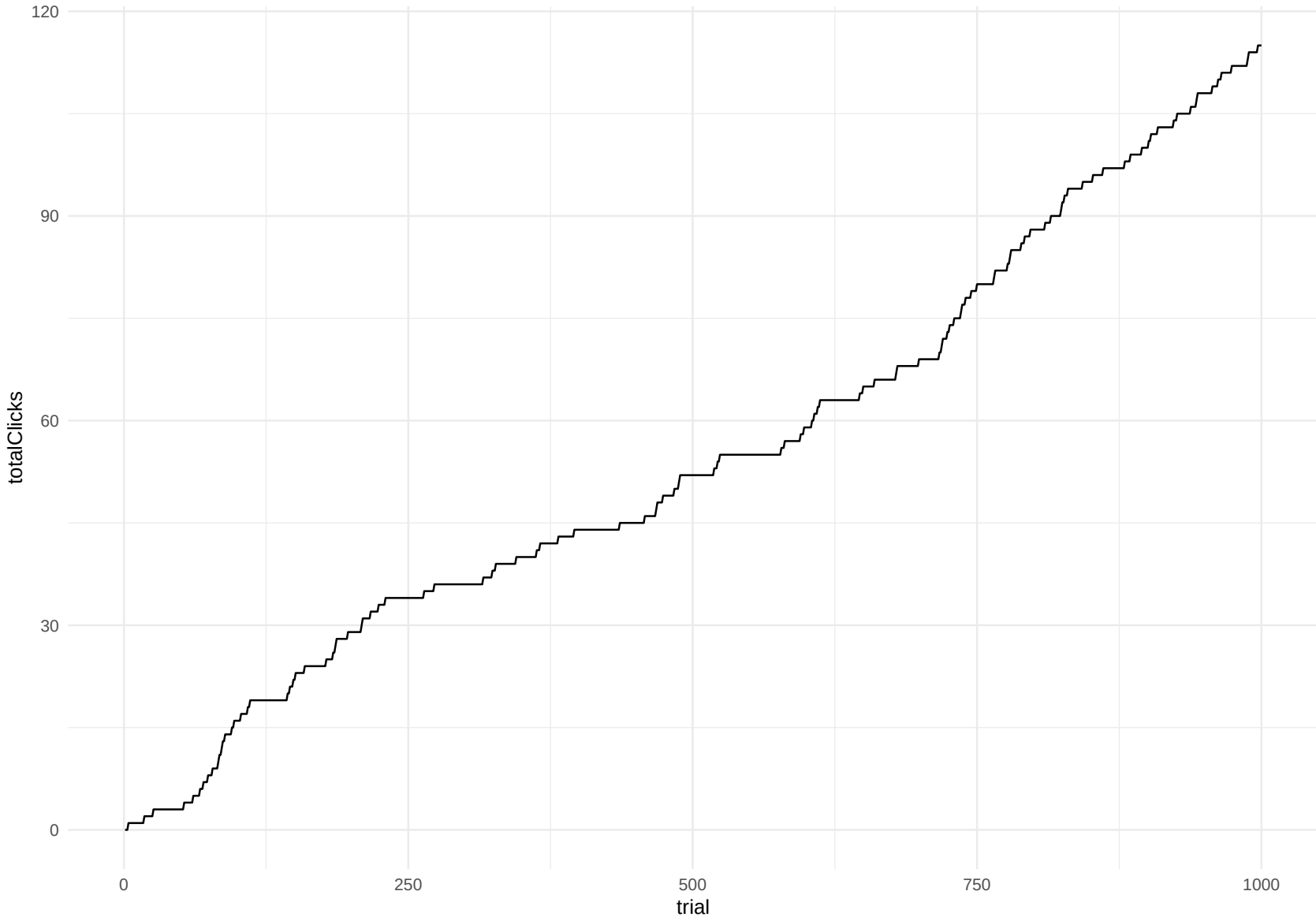
```
#this code is repeated for 'b'
ind_a      = ig_cb$group == 'A' & ig_cb$trial < t
clicks_a_t = ig_cb$click[ind_a]
context_a  = cbind(mobile[ind_a],male[ind_a])
beta_a     = coef(lm(clicks_a_t ~ context_a))
context_t  = c(1,mobile[t],male[t])
phat_a     = context_t%*%beta_a

arm_t   = ifelse(phat_b  >  phat_a,'B','A')
```

Contextual Bandit results

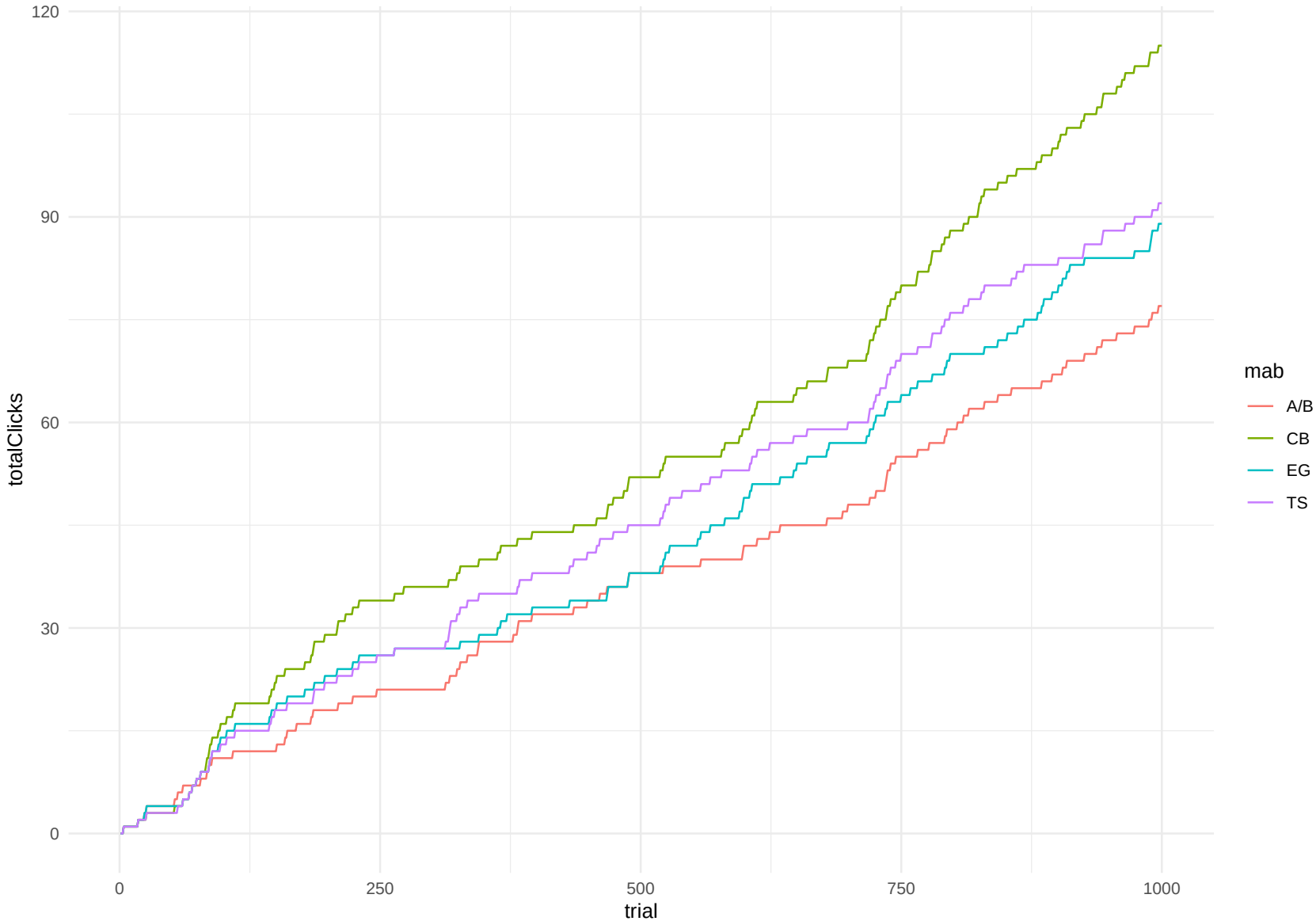
These are now conditional means

```
## # A tibble: 2 x 2
##   group click_bar
##   <chr>     <dbl>
## 1 A        0.0954
## 2 B        0.166
```

Contextual Bandit cumulative clicks



Compare all


120
90
60
30
0
totalClicks
0
250
500
750
1000
trial
mab
A/B
CB
EG
TS

Contexual Bandit use cases

Unlike standard MAB, which only considers past rewards, contextual bandits use **machine learning models** to predict the best action given the current context.

Google Ads uses contextual bandits to optimize search ads based on user behavior, time of day, and device type

Netflix and Spotify recommend content dynamically based on past viewing history, user preferences, and current trends

Instacart

Use CB to personalize the shopper experience (e.g., some shopper prefer low priced products)

Challenge: CB needs multiple examples of each action, which is hard as number of actions grows

Traditionally, used ML trained on past data to predict predict probability of adding an item to the cart, but this does not extend to search queries

Trained a CB model "to select the best search ranker for each <user, query> context so as to increase the average relevance of search results as measured by cart_adds_per_search (items added to cart per search query)"

Netflix

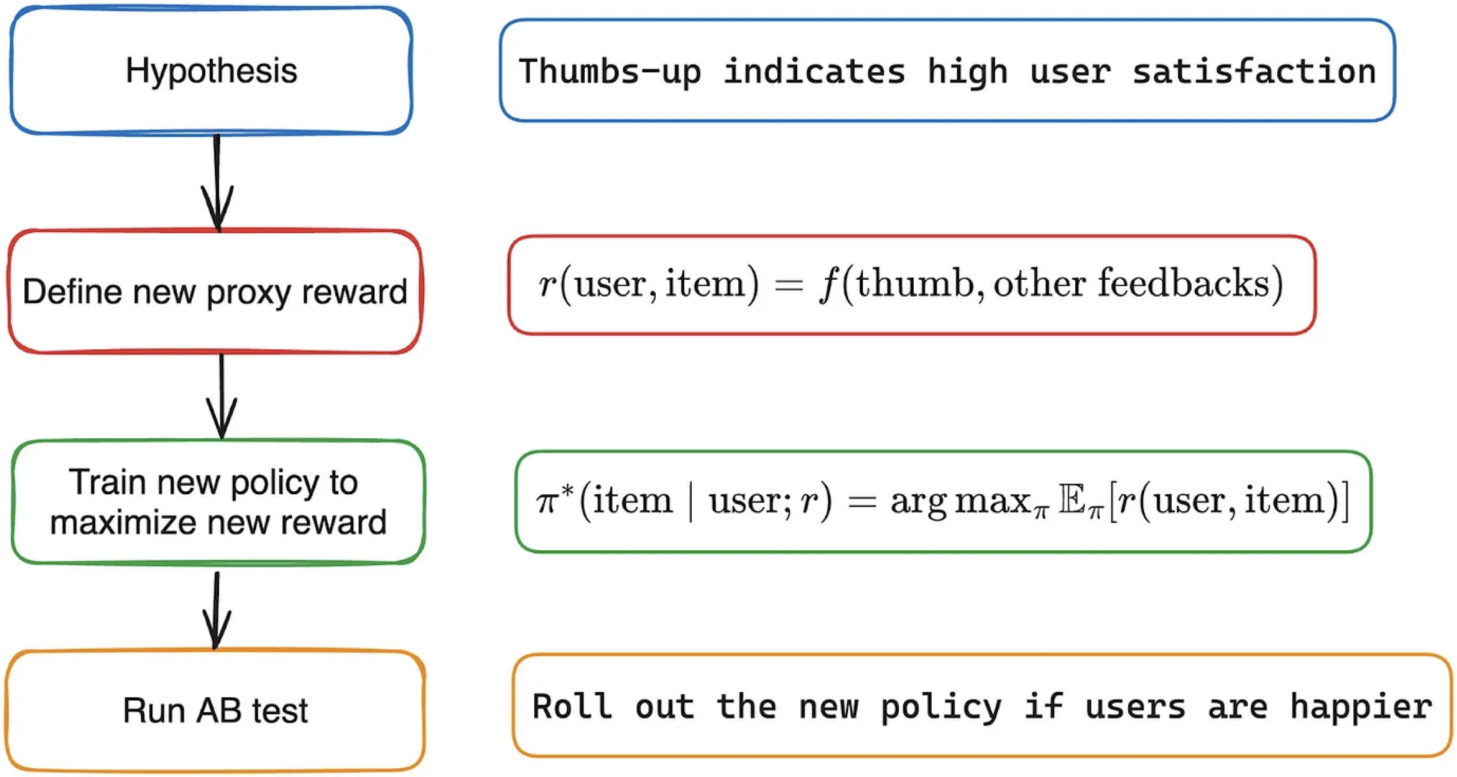
"These feedback signals can be immediate (skips, plays, thumbs up/down, or adding items to their playlist) or delayed (completing a show or renewing their subscription)"

"We can define reward functions to reflect the quality of the recommendations from these feedback signals and then train a contextual bandit policy on historical data to maximize the expected reward"

Click-through rate (CTR), or in our case play-through rate, can be viewed as a simple proxy reward

- Fast season completion (binging): good
- Thumbs-down after completion: bad
- Playing a movie for just 10 minutes: ambiguous
- Discovering new genres: very good

Netflix


Hypothesis
Thumbs-up indicates high user satisfaction
Define new proxy reward
$r(\text{user}, \text{item}) = f(\text{thumb}, \text{other feedbacks})$
Train new policy to maximize new reward
$\pi^*(\text{item} \mid \text{user}; r) = \arg\max_\pi \mathbb{E}_\pi[r(\text{user}, \text{item})]$
Run AB test
Roll out the new policy if users are happier

Choosing Your Bandit

Do you have user context?

- Yes $\rightarrow$ Contextual Bandit
- No $\rightarrow$ Want smarter exploration?
  - Yes $\rightarrow$ Thompson
  - No $\rightarrow$ Epsilon-Greedy

Implementing MAB

What makes contextual bandits ideal for marketers?

- New user, little history
- Cold-start for new creatives
- Rapidly changing behavior (e.g., Black Friday)
- Multiple customer segments

How to execute:

- Platforms (Google, Meta, Mailchimp)
- DIY with:
  - Real-time pipeline
  - Monitoring dashboard
  - Backend experimentation

MAB $\neq$ just code — it's a smarter way to run marketing campaigns