Intro : LTR through supervised learning.

Learning to Rank (LTR) is:

"... the task to automatically construct a ranking model using training data, such that the model can sort new objects according to their degrees of relevance, preference, or importance."

— Liu et al. (2009)

Problem definition:

The **ranking** R of **ranker** f_θ (ranker function) over a document set D is:

$$R = (R_1, R_2, R_3, \ldots)$$

where documents are **ordered** by their (descending) **scores**:

$$f_\theta(R_1) \geq f_\theta(R_2) \geq f_\theta(R_3) \geq \ldots,$$

and **every document** is in the ranking:

$$d \in D \iff d \in R.$$

The goal of learning to rank LTR is to find parameters θ for the model f_θ where sortings items according to their scores, yields the most optimal metrics

↳ this will be defined later.

Supervised Learning to Rank

Before going into assesing ranking based on customer interactions, which is a bit more abstract, let's understand ranking through a more clear situation.

Supervised LTR methods require supervision from **annotated datasets**, these contain:

- **Queries**, representing queries users will issue,
- **Documents**, per query a preselected set of documents to be ranked,
- **Relevance Labels**, indicating relevance/preference per document-query pair.

↳ requires user labelling.

→ $y(d)$

Supervised LTR methods are commonly divided in **three groups**:

- Pointwise, Pairwise, and Listwise.

Pointwise approach

The pointwise approach casts LTR as the standard machine learning task of **label prediction**, by using a classification or regression loss.

For instance, the mean squared error is a common **regression loss**:

$$\mathcal{L}_{pointwise} = \frac{1}{N}\sum_{i=1}^{N}\left(f_\theta(d_i) - y(d_i)\right)^2.$$

it's not a pure LTR technique.
(if this worked well we wouldn't need a separate field to study LTR)

The **issue** with pointwise methods is that they **ignore** that model **scores** are **used to rank** documents.

In other words:

- A pointwise loss only wants scores to be **close to the labels**,
- LTR only wants scores to result in the **correct ordering**.

In practice, pointwise methods **compromise ordering** to get scores closer to the labels.

- Pointwise approaches only look at a single document at a time.
  ↳ this means that the score given to each document is independent from the rest).

Pairwise approach

The pairwise approach realizes that **ordering** is based on **relative score differences**. It uses a loss based on **pairs of documents** with a difference in relevance (Joachims, 2002).

For instance, an (unnormalized) **pairwise hinge-loss**:

$$\mathcal{L}_{pairwise} = \sum_{y(d_i)>y(d_j)} \max\Big(0, 1 - \big(f_\theta(d_i) - f_\theta(d_j)\big)\Big).$$

pushes the scoring so that the score of d_i is bigger than d_j (because the relevance label of i is bigger than j)

this 0 is just to cap the optimising process when items are ranked as desired

↳ sum over all pairs of relevance labels for each item (where 1 item more relevant than the other)

⇒ scores $f_\theta(d)$ don't have to match the value assigned to the relevance label $y(d)$

↳ only the ordering matters.
(unlike pointwise where we want $f_\theta(d) = y(d)$)

The **problem** with the pairwise approach:

- **every document pair** is treated as **equally important**,
- often **users care** more about the **top-10** than the **top-100**.

Thus pairwise methods may compromise quality in the top-10 to improve the ordering in the tail of the top-100.

↳ This method is also known as RankNet.

- The goal for the ranker is to minimize the number of pair results where there is a wrong order (in scores) relative to the ground truth (labels)
- RankNet optimizes the cost function using Stochastic Gradient Desc.

Listwise methods

look at the ENTIRE LIST of ITEMS and try to come up with the optimal ordering of it.

The idea of **listwise methods** is to **optimize ranking metrics** directly. However, metrics based on the rank function are **not differentiable**.

For instance, the **Discounted Cumulative Gain** metric:

examples of metrics in next page.

$$\text{DCG} = \sum_{i=1}^{N} \frac{y(d_i)}{\log_2(\text{rank}(d_i) + 1)}.$$

Problem: $\log_2(\text{rank}(d_i) + 1)$ is not differentiable.

Solutions to this problem:

- use **probabilistic approximations** of ranking:
  e.g. ListNet (Cao et al., 2007), ListMLE (Xia et al., 2008),
- use **heuristics** or **bounds** on metrics:
  e.g. LambdaRank (Burges, 2010), LambdaLoss (Wang et al., 2018c).

For instance, the **LambdaRank** loss is a proven bound on DCG:

$$\mathcal{L}_{LambdaRank} = \sum_{y(d_i) > y(d_j)} \log\left(1 + e^{f_\theta(d_j) - f_\theta(d_i)}\right) \cdot |\Delta\text{DCG}|.$$

↳ LambdaRank improves over RankNet due to: (no need for slow stochastic gradient descent)

1. There is no need to use the costs for SGD, you only need the gradients of cost with respect to the model score.
2. If you also scale these gradients using nDCG it improves both in speed and accuracy.

how much you want to move each document.

LamdaMART

- LambaMART seems to be the current state of the art algorithm for learning to rank.
- It combines LambaRank and MART (Multiple additive trees) (basically a GBM type model)
  - ↳ MART uses gradient boosted decision trees for prediction tasks.
  - ↳ LambaMART uses gradient boosted decision trees using a cost function derived from LambdaRank for solving a ranking task.

1) What does Lambda represent in LambaRank?

How display of 2 documents should look like based on scoring

Lambda represents the gradients (the strength) of how much to move each document.

difference in DCG is we swap position of i and j in the ranking

$$\mathcal{L}_{LambdaRank} = \sum_{y(d_i)>y(d_j)} \log\left(1 + e^{f_\theta(d_j) - f_\theta(d_i)}\right) \cdot |\Delta DCG|.$$

↳ knowing d_i is more relevant than d_j

2) How is LambdaMART implemented?

Algorithm: LambdaMART

Initialization

set number of trees N, number of training samples m, number of leaves per tree L, learning rate η
for $i = 0$ to m **do**
  $F_0(x_i) = \text{BaseModel}(x_i)$ //If BaseModel is empty, set $F_0(x_i) = 0$
end for
for $k = 1$ to N **do**

Boosted Tree Step

  for $i = 0$ to m **do** — Calculate lambda and weight
    $y_i = \lambda_i$
    $w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$
  end for
  $\{R_{lk}\}_{l=1}^{L}$ // Create L leaf tree on $\{x_i, y_i\}_{i=1}^{m}$ — Create regression trees for lambda
  $\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$ // Assign leaf values based on Newton step.
  $F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$ // Take step with learning rate η. — Calculate leaf node values and update the predicted score
end for

METRICS

Flat and "Rank-less" Evaluation metrics

- Accuracy metrics
  - MAE (mean absolute error) or RMSE (root mean squared error)
  - These focus on comparing actual vs predicted ratings.
  - For example, this could be useful if we wanted to predict the rating of a new product

- Decision support metrics
  - Precision, recall, F1, etc..
  - These focus on how well a recommender helps users make good decisions.
    - ↳ Precision is the percentage of selected items that are actually relevant
    - ↳ Recall is the percentage of relevant items that the system selected
    - ↳ F1 is a harmonic mean to balance precision and recall.

These metrics have 2 main problems:

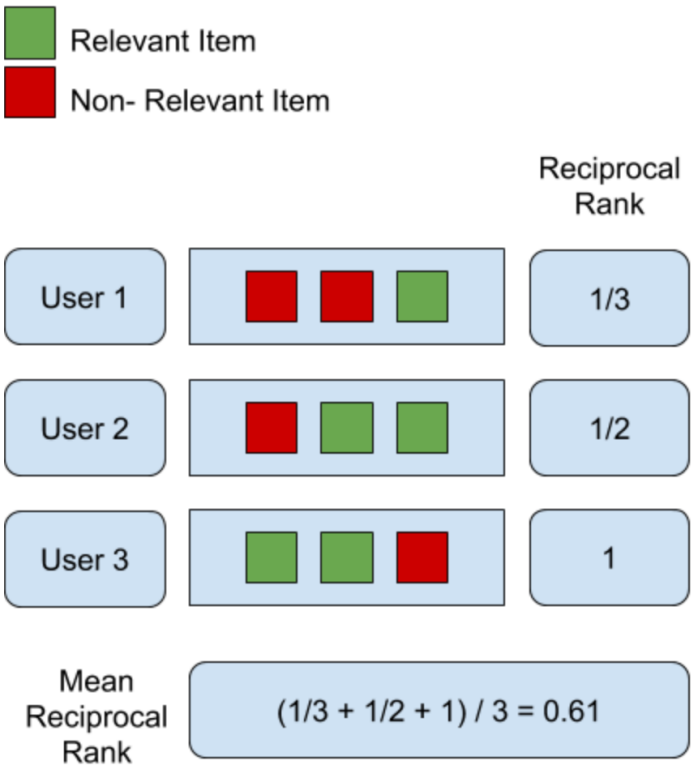
1) Entire dataset vs Top-n: these metrics are computed for the entire set of items passed for evaluation
   ↳ one could fix this through Precision@N
   BUT

2) Finding vs Ranking: these metrics are good at FINDING items, not at RANKING.

Rank-aware metrics

1. MRR - Mean reciprocal rank.

- Probably the simplest learning to rank metric.
- Measures "where is the FIRST relevant item"



For each user u:

- Generate list of recommendations
- Find rank k_u of its first relevant recommendation (the first rec has rank 1)
- Compute reciprocal rank $\frac{1}{k_u}$

Overall algorithm performance is mean recip. rank:

$$\text{MRR}(O, U) = \frac{1}{|U|} \sum_{u \in U} \frac{1}{k_u}$$

+ Easy to compute
+ Good for know-item or well defined searches (for example, looking for facts).

- Doesn't focus on the rest of the list of items, only on the first.
- Not recommended when a user wants to explore and compare.

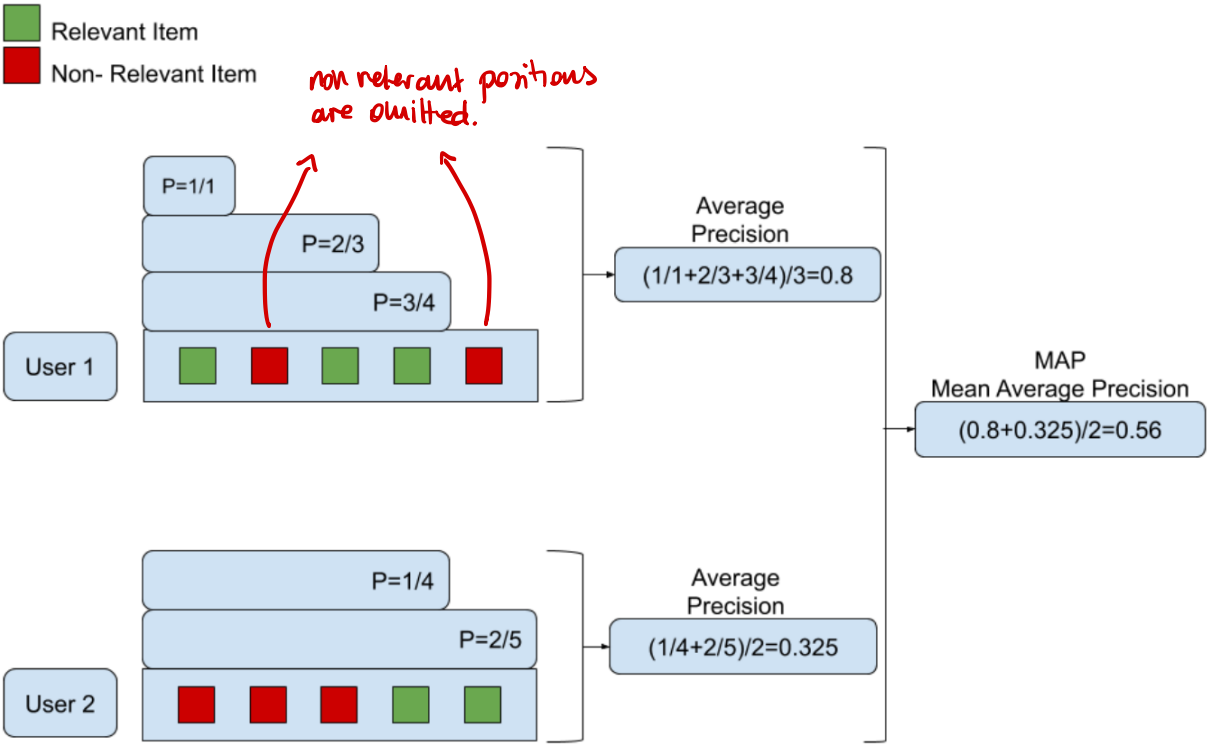
2. MAP → Mean average position

- Works with binary relevance data sets. (1 or 0)
- Remember that Precision@N had the problem that all items in the evaluated set are treated equally
  - ↳ we would prefer to have few errors in top ranks and don't really care about errors in lower positions.

- Average precision is a metric that tells you how much of the relevant items are concentrated in the highest ranked positions.

$$AP = \sum_k (\text{Recall@k} - \text{Recall@(k-1)}) \cdot \text{Precision@k}$$

For each user
- For each relevant item
  - Compute precision of list *through* that item
- Average sub-list precisions



+ Provides a single metric to represent the complex Area under the Precision-Recall curve
+ Able to give more weight to errors that happen high up in the recommended list
  - ↳ but it also downgrades weight of errors that happen deeper in the list.

- only works for binary relevance metrics.

3. nDCG → normalized discounted cumulative gain.

- Like MAP, nDCG values putting highly relevant items high up the recommended lists.
- However, nDCG can improve over MAP by working with non-binary relevance scores
  - ↳ this would allow us to define more important/relevant items than other.

Cumulative Gain

$$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$$

p elements in the recommended ranking

graded relevance of the result at position i (gain)

Discounted Cumulative Gain (standard)

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

logarithmic reduction factor to penalize proportionally to the position of the result

Discounted Cumulative Gain (industry)

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

graded relevance to get a stronger emphasis on retrieving relevant documents

Ideal Discounted Cumulative Gain

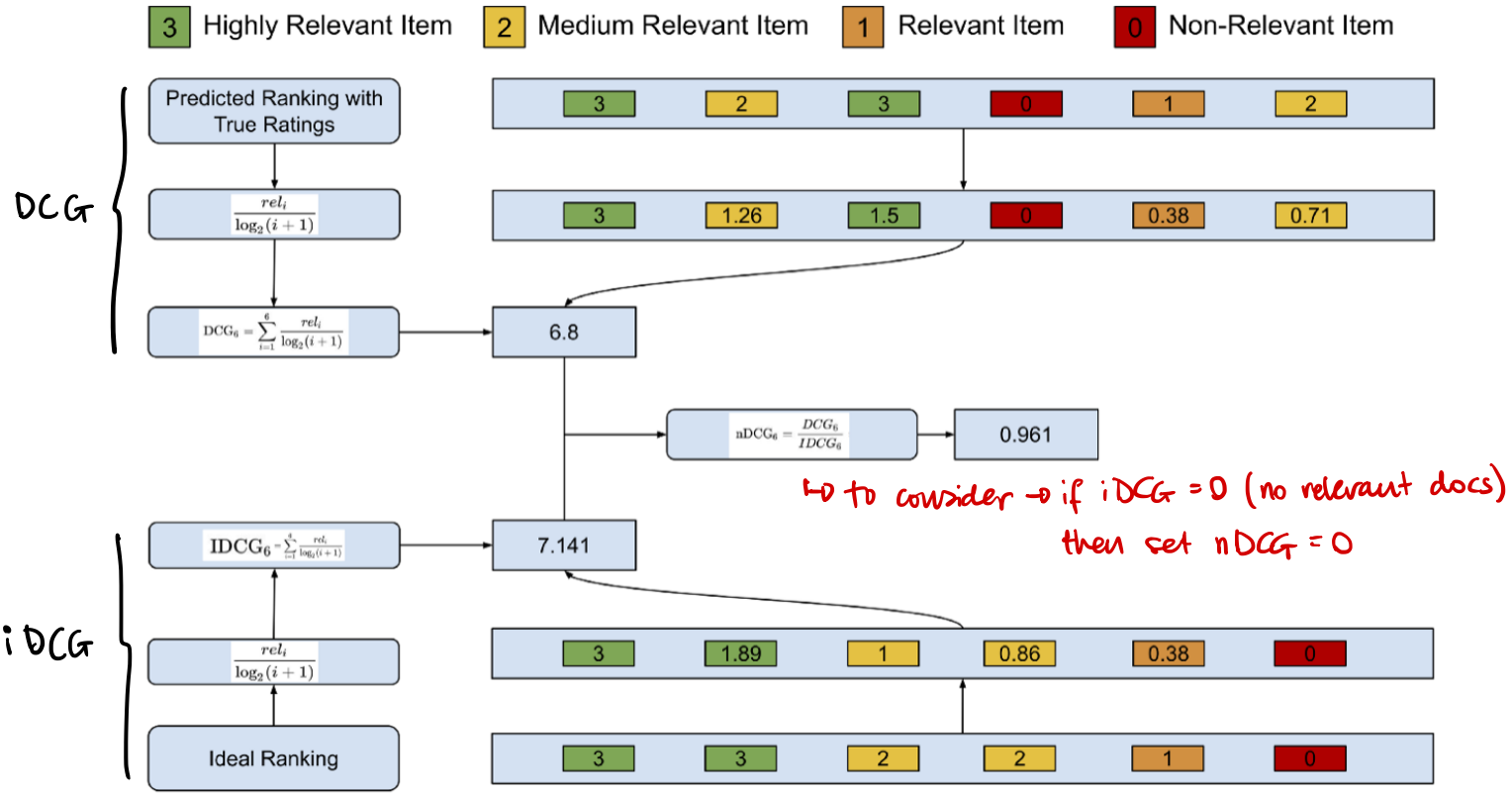
$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

list of relevant documents (ordered by their relevance) in the corpus up to position p

Normalized Discounted Cumulative Gain

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$$

Average Normalized Discounted Cumulative Gain Across Users



investigate

ERR:

(Expected reciprocal rank)

Yandex pfound.

Limitations of Annotated Datasets

Annotated datasets are **valuable** and have an **important place in research and development**.

However, the supervised approach is:

- **Unavailable** for practitioners without a **considerable budget**.
- **Impossible** for certain ranking problems.
- Often **misaligned** with *true* user preferences.

Therefore, there is a **need** for an **alternative** learning to rank approach.

Some of the most substantial limitations of **annotated datasets** are:

- **expensive** to make (Qin and Liu, 2013; Chapelle and Chang, 2011).
- **unethical** to create in **privacy-sensitive settings** (Wang et al., 2016a).
- **impossible** for small scale problems, e.g., **personalization**.
- **stationary**, cannot capture **future changes in relevancy** (Lefortier et al., 2014).
- **not necessarily aligned with actual user preferences** (Sanderson, 2010),
  i.e., annotators and users often disagree.

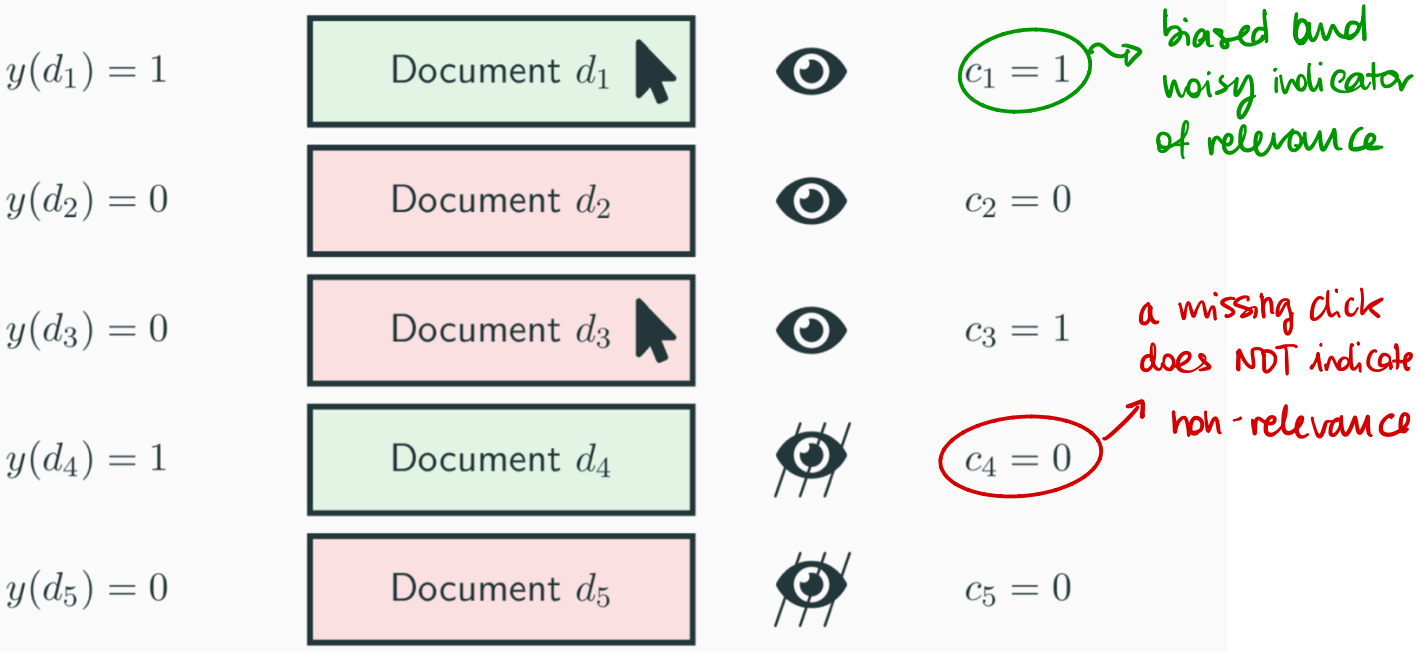
Part 2:

Counterfactual (unsupervised) offline evaluation

Counterfactual Evaluation

Evaluation is incredibly **important before deploying** a ranking system.

we often do not know the TRUE relevance of an item → But we do get to know the implicit information provided by clicks



If we are dealing with noise (someone clicking accidentally), then averaging over many clicks will remove the effect

↳ but if we are dealing with Bias, then averaging will NOT remove this effect.

(example, even if you have millions of click data, items in page 100 of the results page might never be clicked)

However, with the **limitations of annotated datasets**, can we **evaluate** a ranker **without deploying** it or **annotated data**?

without.

Counterfactual Evaluation: Examination User Model

Before starting to formalise models to estimate how well we are ranking without knowing true relevance, let's introduce the formalisation of the user behaviour presented in the previous slide.

If we **only** consider **examination** and **relevance**, a user click can be modelled by:

- The probability of document d_i **being examined** ($o_i = 1$) in a ranking R:

  based on ranking

$$P(o_i = 1 \mid R, d_i).$$

- The probability of a **click** $c_i = 1$ on d_i given its **relevance** $y(d_i)$ and whether it was **examined** o_i:

  only dependant on the item

  a relevant doc should get more clicks

$$P(c_i = 1 \mid o_i, y(d_i)).$$

  but was it observed?

- **Clicks only occur on examined documents**, thus the probability of a click in ranking R is:

  ranking

$$P(c_i = 1 \wedge o_i = 1 \mid y(d_i), R) = P(c_i = 1 \mid o_i = 1, y(d_i)) \cdot P(o_i = 1 \mid R, d_i).$$

  prob of click & being observed | relevance | prob of click given it was observed | prob of document being examined.

this is the main assumption for this examination user model. For example:

- a user is not randomly clicking without observing what they click
- a user cannot click on an item in page 100 of results, if they haven't been exposed/shown it.

Counterfactual Evaluation: Naive Estimator → the WRONG WAY !!!

If you remember, from the supervised learning to rank section, we introduced the idea of a Δ function:

where we had the TRUE relevance.

$$\Delta(f_\theta, D, y) = \sum_{d_i \in D} \lambda(rank(d_i \mid f_\theta, D)) \cdot y(d_i),$$

$\lambda \equiv$ rank aware metrics

⇒ There is no bias within the formula

As a first NAIVE model, we can take this same function and assume that clicks are equivalent to TRUE relevance:

- if user clicks ⇒ relevant
- if user doesn't click ⇒ not relevant.

A **naive way** to estimate is to assume clicks are a unbiased relevance signal:

$$\Delta_{NAIVE}(f_\theta, D, c) = \sum_{d_i \in D} \lambda(rank(d_i \mid f_\theta, D)) \cdot c_i.$$

clicks ≡ relevance

Even if **no click noise** is present: $P(c_i = 1 \mid o_i = 1, y(d_i)) = y(d_i)$, this estimator is **biased** by the examination probabilities:

$$\mathbb{E}_o[\Delta_{NAIVE}(f_\theta, D, c)] = \mathbb{E}_o \left[\sum_{d_i : o_i = 1 \wedge y(d_i) = 1} \lambda(rank(d_i \mid f_\theta, D)) \right]$$

condition for no click noise → item was observed and clicked (relevant)

expectation

$$= \sum_{d_i : y(d_i) = 1} P(o_i = 1 \mid R, d_i) \cdot \lambda(rank(d_i \mid f_\theta, D)).$$

sum of relevant items → for each item we have

weighted rank
- good because it's what we want
- remember λ can be nDCG or other

but we have this term that is biasing the final result because it is required that the user has seen the item

In rankings, **documents at higher ranks** are more likely to be examined: **position bias**.

Position bias causes **logging-policy-confirming** behavior:

→ we deployed a ranker and we think it's the best ranker because it gets clicks.

- Documents displayed at **higher ranks during logging** are incorrectly considered as **more relevant**.

Inverse Propensity Scoring

We are going to try to get rid of that unbiased term so that our Δ function is only dependant on λ weighting scores

Counterfactual evaluation accounts for bias using **Inverse Propensity Scoring (IPS)**:

$$\Delta_{IPS}(f_\theta, D, c) = \sum_{d_i \in D} \frac{\lambda(rank(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)} \cdot c_i,$$

new possible ranking

ranker used to collect click data.

(*)

where

- $\lambda(rank(d_i \mid f_\theta, D))$: (weighted) rank of document d_i by ranker f_θ,
- c_i: observed click on the document in the log,
- $P(o_i = 1 \mid R, d_i)$: examination probability of d_i in ranking R displayed during logging.

This is an **unbiased estimate** of any additive linearly decomposable IR metric.

(*) In simple terms, by adding the bias in the denominator we are:

1. If probability of document being observed is high (normally top positions) then overall weighting will decrease
2. And obviously the inverse. The less chance there was in the past of being observed the higher the weighting score for the new ranker.

Counterfactual Evaluation: Proof of Unbiasedness

If no click noise is present, this provides an **unbiased estimate**:

same assumption as for the naive approach

$$\mathbb{E}_o[\Delta_{IPS}(f_\theta, D, c)] = \mathbb{E}_o\left[\sum_{d_i \in D} \frac{\lambda(rank(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)} \cdot c_i\right]$$

$$= \mathbb{E}_o\left[\sum_{d_i: o_i=1 \wedge y(d_i)=1} \frac{\lambda(rank(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)}\right]$$

$$= \sum_{d_i: y(d_i)=1} \frac{P(o_i = 1 \mid R, d_i) \cdot \lambda(rank(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)}$$

$$= \sum_{d_i \in D} \lambda(rank(d_i \mid f_\theta, D)) \cdot y(d_i)$$

$$= \Delta(f_\theta, D, y).$$

this is very powerful because by plugging clicks we are going to get something out of it that tells us something about true relevance.

by the way, by having this in the denominator we implicitly introducing the idea that it's impossible that a document has 0 probability of being observed (can't divide by 0).

Counterfactual Evaluation: Robustness of Noise

So far we have **no click noise**: $P(c_i = 1 \mid o_i = 1, y(d_i)) = y(d_i)$.

However, the IPS approach still works without these assumptions, as long as:

one doc is more relevant than the other

$$y(d_i) > y(d_j) \Leftrightarrow P(c_i = 1 \mid o_i = 1, y(d_i)) > P(c_j = 1 \mid o_j = 1, y(d_j)).$$

probability of clicks should also be larger given that both were observed.

Since we can prove **relative differences** are inferred unbiasedly:

$$\mathbb{E}_{o,c}[\Delta_{IPS}(f_\theta, D, c)] > \mathbb{E}_{o,c}[\Delta_{IPS}(f_{\theta'}, D, c)] \Leftrightarrow \Delta(f_\theta, D) > \Delta(f_{\theta'}, D).$$

with noise you won't be able to know the exact relevance or utility for each document but you will know that one is better than another (which is what you want → the relative differences)

↳ ie → is my new ranker better based on historical interaction data

Part 3:

Counterfactual learning to Rank.

(unsupervised)

Propensity-weighted Learning to Rank

Now that we have an unbiased estimate, how can we maximise the value of f_θ (new ranker)?

The inverse-propensity-scored estimator can unbiasedly estimate performance:

$$\Delta_{IPS}(f_\theta, D, c) = \sum_{d_i \in D} \frac{\lambda(rank(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)} \cdot c_i.$$

2 immediate problems with this estimator !!!

How do we **optimize** for this **unbiased performance estimate**?

- It is **not differentiable**. (rank functions are not soft)
- **Common problem for all ranking metrics**.

Solution 1:

Upper Bound on Rank → whilst we can't optimise on the global rank, we can optimise the upper bound, which implicitly, minimising upper bound lowers the rank.

Rank-SVM (Joachims, 2002) optimizes the following **differentiable upper bound**:

$$\begin{aligned} rank(d \mid f_\theta, D) &= \sum_{d' \in R} \mathbb{1}[f_\theta(d) \leq f_\theta(d')] \\ &\leq \sum_{d' \in R} \max(1 - (f_\theta(d) - f_\theta(d')), 0) = \overline{rank}(d \mid f_\theta, D). \end{aligned}$$

Alternative choices are possible, i.e., a **sigmoid-like bound** (with parameter σ):

$$rank(d \mid f_\theta, D) \leq \sum_{d' \in R} \log_2(1 + \exp^{-\sigma(f_\theta(d) - f_\theta(d'))}).$$

Commonly used for pairwise learning, LambdaMart (Burges, 2010), and Lambdaloss (Wang et al., 2018c).

Propensity-weighted LTR: Average Relevance Position

it's nice to implement for ARP but there are other metrics

Then for the Average Relevance Position metric:

$$\Delta_{ARP}(f_\theta, D, y) = \sum_{d_i \in D} rank(d_i \mid f_\theta, D) \cdot y(d_i).$$

This gives us an **unbiased estimator** and **upper bound**:

$$\begin{aligned}\Delta_{ARP\text{-}IPS}(f_\theta, D, c) &= \sum_{d_i \in D} \frac{rank(d_i \mid f_\theta, D)}{P(o_i = 1 \mid R, d_i)} \cdot c_i \\ &\leq \sum_{d_i \in D} \frac{\overline{rank}(d_i \mid f_\theta, D)}{P(o_i = 1 \mid R, d_i)} \cdot c_i,\end{aligned}$$

This upper bound is **differentiable** and **optimizable** by stochastic gradient descent or Quadratic Programming, i.e., Rank-SVM (Joachims, 2006).

Propensity-weighted LTR: Additive Metrics

A similar approach can be applied to **additive metrics** (Agarwal et al., 2019a).

If λ is a **monotonically decreasing** function:

$$x \leq y \Rightarrow \lambda(x) \geq \lambda(y),$$

then:

$$rank(d \mid \cdot) \leq \overline{rank}(d \mid \cdot) \Rightarrow \lambda(rank(d \mid \cdot)) \geq \lambda(\overline{rank}(d \mid \cdot)).$$

This provides a **lower bound**, for instance for Discounted Cumulative Gain (DCG):

$$\frac{1}{\log_2(1 + rank(d \mid \cdot))} \geq \frac{1}{\log_2(1 + \overline{rank}(d \mid \cdot))}.$$

if we maximise this upper bound, then we indirectly optimise the true score for the rank.

Propensity-weighted LTR: Discounted Cumulative Gain

Then for the Discounted Cumulative Gain metric:

$$\Delta_{DCG}(f_\theta, D, y) = \sum_{d_i \in D} \log_2(1 + \mathit{rank}(d_i \mid f_\theta, D))^{-1} \cdot y(d_i).$$

This gives us an **unbiased estimator** and **lower bound**:

$$\begin{aligned}\Delta_{DCG\text{-}IPS}(f_\theta, D, c) &= \sum_{d_i \in D} \frac{\log_2(1 + \mathit{rank}(d_i \mid f_\theta, D)^{-1}}{P(o_i = 1 \mid R, d_i)} \cdot c_i \\ &\geq \sum_{d_i \in D} \frac{\log_2(1 + \overline{\mathit{rank}}(d_i \mid f_\theta, D)^{-1}}{P(o_i = 1 \mid R, d_i)} \cdot c_i.\end{aligned}$$

This lower bound is **differentiable** and **optimizable** by stochastic gradient descent or the Convex-Concave Procedure (Agarwal et al., 2019a).

Propensity-weighted LTR: Walkthrough → summary

Overview of the approach:

- Obtain a **model of position bias**.
- Acquire a **large click-log**.
- Then for every click in the log:
  - Compute the **propensity of the click**:

$$P(o_i = 1 \mid R, d_i).$$

  - Calculate the **gradient** of the **bound** on the **unbiased estimator**:

$$\nabla_\theta \left[\frac{\overline{\mathit{rank}}(d_i \mid f_\theta, D)}{P(o_i = 1 \mid R, d_i)} \right].$$

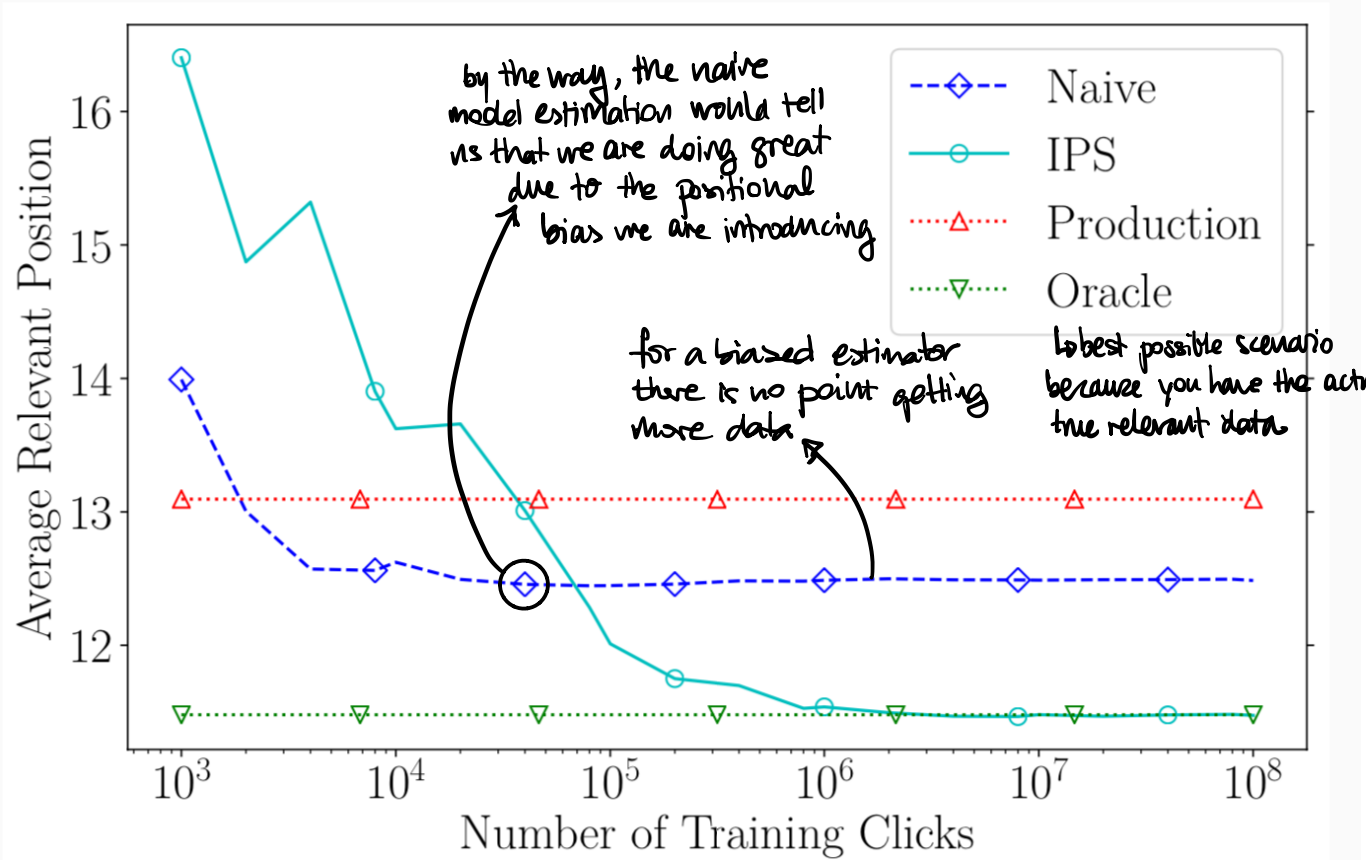
  - **Update the model** f_θ by adding/subtracting the gradient.

Experiments to understand effects.

Unbiased LTR methods are commonly **evaluated** through **semi-synthetic experiments** (Joachims, 2002; Agarwal et al., 2019a; Jagerman et al., 2019).

The experimental setup:

- Traditional LTR dataset, e.g., Yahoo! Webscope (Chapelle and Chang, 2011).
- Simulate queries by uniform sampling from the dataset.
- Create a ranking according to a baseline ranker.
- Simulate clicks by modelling:
  - **Click Noise**, e.g., 10% chance of clicking on a non-relevant document.
  - **Position Bias**, e.g., $P(o_i = 1 \mid R, d_i) = \frac{1}{rank(d|R)}$.
- Hyper-parameter tuning by unbiased evaluation methods.



↳ unfortunately, unless you have TRUE relevance labels, you can't produce this analysis on your production system

Part 4: Estimating position bias

(can we calculate a score for this bias?)

Estimating Position Bias

So far we have seen how to:

- Perform **Counterfactual Evaluation** with **unbiased estimators**.
- Perform **Counterfactual LTR** by optimizing **unbiased estimators**.

At the core of these methods is the propensity score: $P(o_i = 1 \mid R, d_i)$, which helps to remove bias from user interactions.

In this section, we will show how this **propensity score** can be **estimated** for a specific kind of bias: **position bias**.

Recall that position bias is a form of bias where higher positioned results are more likely to be observed and therefore clicked.

Assumption: The **observation probability** only depends on the rank of a document:

$$P(o_i = 1 \mid i).$$

The objective is now to **estimate**, for each rank i, the propensity $P(o_i = 1 \mid i)$.

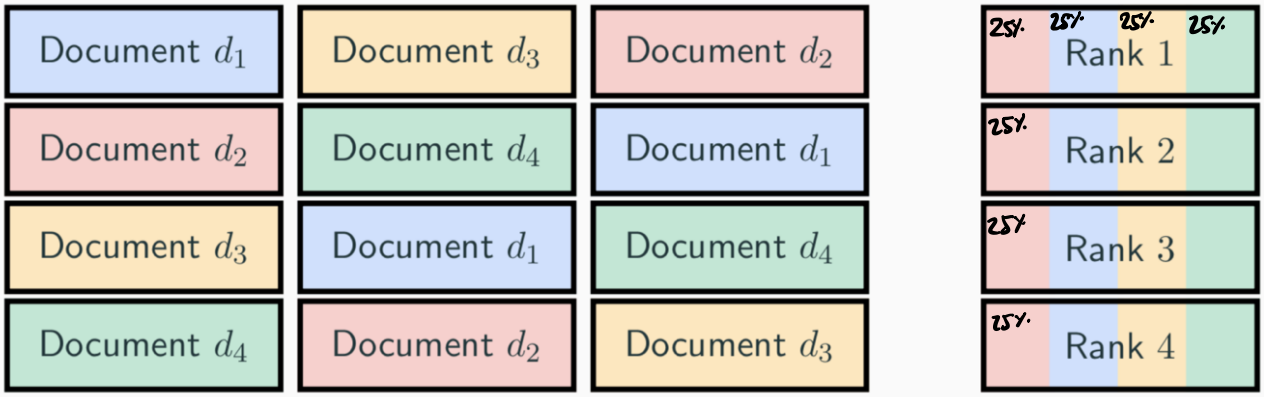
This user model was first formalized by Craswell et al. (2008).

Ideas on how to deal with positional bias:

- Random sampling - rand-TopN algorithm
- Filter "unseen" results - takes results above last clicked position
- Positional weighting
- Add positional features (rule #36 of Googles rules of ML)

Estimating Position Bias

RandTop-n Algorithm: Once you decide your top-n document, implement a random shuffle.



↳ with enough user clicks, each document would have uniformly landed in top-N.

↳ In fact, what we have achieved is a way to remove relevance of the documents in the top N positions → the chances of positional display bias are reduced

- 25% of time doc 1 in rank-n
- rank 1 has 25% of times doc 1, 2, 3, 4

↳ In other words, irrespective of what document you place in position 1 or 2, we can start measuring how much clicks would we expect for position 1 and 2, and this wouldn't be affected by the relevance of the document.

RandTop-n Algorithm:

1. Repeat:
   - Randomly shuffle the top n items
   - Record clicks
2. Aggregate clicks per rank
3. Normalize to obtain propensities $p_i \propto P(o_i \mid i)$

in fact you don't really need to know that in position 1 there is 90% of being observed → what you want to know is that pos2 is half as likely to be observed than 1. (relative differences)

Note: we only need propensities proportional to the true observation probability for learning.

↳ obvious problem → you might be losing money with this approach!

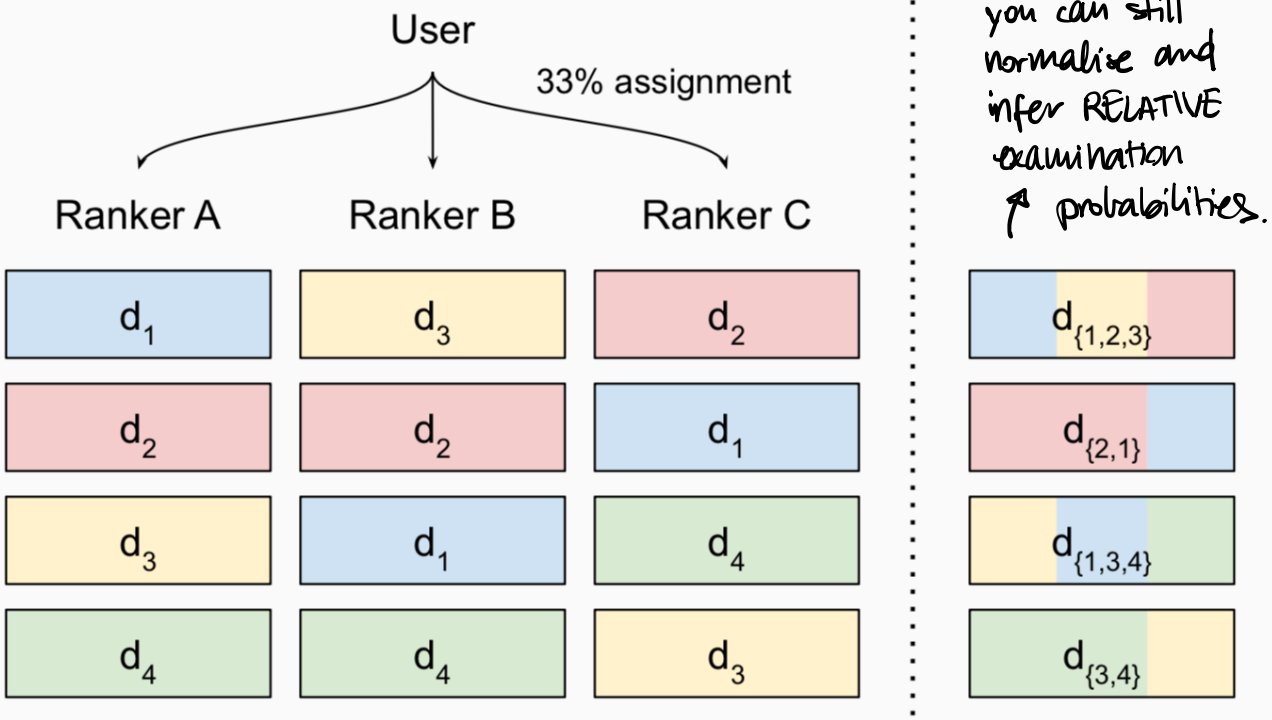
Uniformly **randomizing** the top n results may negatively impacts users during data logging.

There are various methods that minimize the impact to the user:

- **RandPair:** Choose a pivot rank k and only swap a random other document with the document at this pivot rank (Joachims et al., 2017b).
- **Interventional Sets:** Exploit inherent "randomness" in data coming from multiple rankers (e.g., A/B tests in production logs) (Agarwal et al., 2017).

- As we have seen, to measure position bias, the most straightforward approach is to perform randomization.
- Naturally, we want to avoid randomizing because this negatively affects the end-user experience.
- **Main idea**: In real-world production systems many (randomized) interventions take place due to *A/B tests*. Can we use these interventions instead?
- This approach is called *intervention harvesting* (Agarwal et al. (2017); Fang et al. (2019); Agarwal et al. (2019c))

Intervention Harvesting



Part 5 : Jointly learning & estimating

(Expectation - Maximisation algorithm)

Jointly Learning and Estimating

In the previous sections we have seen:

- Counterfactual ranker evaluation with unbiased estimators.
- Counterfactual LTR by optimizing unbiased estimators.
- Estimating propensity scores through randomization. → estimating position bias.

Instead of treating **propensity estimation** and **unbiased learning to rank** as two separate tasks, recent work has explored **jointly learning rankings and estimating propensities**.

↳ instead of optimising based on unbiased estimators and then checking what position bias exists, could we do both things together?

Recall that the probability of a click can be decomposed as:

$c_i \equiv$ clicked
$o_i \equiv$ observed

$$\underbrace{P(c_i = 1 \wedge o_i = 1 \mid y(d_i), R)}_{\text{click probability}} = \underbrace{P(c_i = 1 \mid o_i = 1, y(d_i))}_{\text{relevance probability}} \cdot \underbrace{P(o_i \mid R, d_i)}_{\text{observation probability}}.$$

In the previous sections we have seen that, if the **observation probability** is known, we can find an unbiased estimate of relevance via IPS.

It is possible to **jointly learn and estimate** by iterating two steps:

(before you know the position bias → that's why we correct for it)

1. Learn an optimal ranker given a correct propensity model:

$$\underbrace{P(c_i = 1 \mid o_i = 1, y(d_i))}_{\text{relevance probability}} = \frac{P(c_i = 1 \wedge o_i = 1 \mid y(d_i), R)}{P(o_i \mid R, d_i)}.$$

prob of being clicked and observed given Relevance and ranker, and corrected by position

2. Learn an optimal propensity model given a correct ranker:

$$\underbrace{P(o_i \mid R, d_i)}_{\text{observation probability}} = \frac{P(c_i = 1 \wedge o_i = 1 \mid y(d_i), R)}{P(c_i = 1 \mid o_i = 1, y(d_i))}.$$

1) Learn a ranker without knowing position bias → it would be better than what you have but still sub-opt.
2) Then you update the observation probability (position bias) based on how the ranker shows the documents.
3) Back to step 1, but this time you have a better understanding of position bias to correct for it.

Part 6 :

Addressing trust bias

Addressing Trust Bias

In recent work Agarwal et al. (2019b) also address trust bias.

Trust bias:

- Users more often **overestimate** the **relevance** of **higher** ranked documents, and more often **underestimate** the **relevance** of **lower** ranked documents (Agarwal et al., 2019b; Joachims et al., 2017a).

position bias is more about not having the chance to be exposed to the document. trust bias is about the perception that, the same document would be trusted more upper in the ranking

Trust bias is related to position bias but involves more than just examination bias.

Modelling Trust Bias

Clicks are now modelled on the **perceived relevance** $\tilde{y}(d_i)$ instead of the **actual relevance** $y(d_i)$:

still captures position bias

$$P(c_i \mid d_i, R, y) = P(\tilde{y}(d_i) = 1 \mid y(d_i), R) \cdot P(o_i = 1 \mid R, d_i).$$

Agarwal et al. (2019b) model the perceived relevance conditioned on the actual relevance and **display position** $\mathit{rank}(d_i, R) = k$:

$$P(\tilde{y}(d_i) = 1 \mid y(d_i) = 1, k) = \epsilon_k^+$$
$$P(\tilde{y}(d_i) = 1 \mid y(d_i) = 0, k) = \epsilon_k^-$$

ϵ_k^+ → probability of correct click

ϵ_k^- → probability of incorrect click.

↳ when user thinks it is relevant but it is NOT

↳ in fact, if these probabilities were the same for all ranks (k), then we would eliminate trust bias and only be left with position bias

Correcting for Trust Bias

The new estimator becomes:

$$\Delta_{\text{Bayes-IPS}}(f_\theta, D, c) = \sum_{d_i \in D} P(y(d_i) = 1 | c_i = 1, k) \cdot \frac{\lambda(\text{rank}(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)} \cdot c_i$$

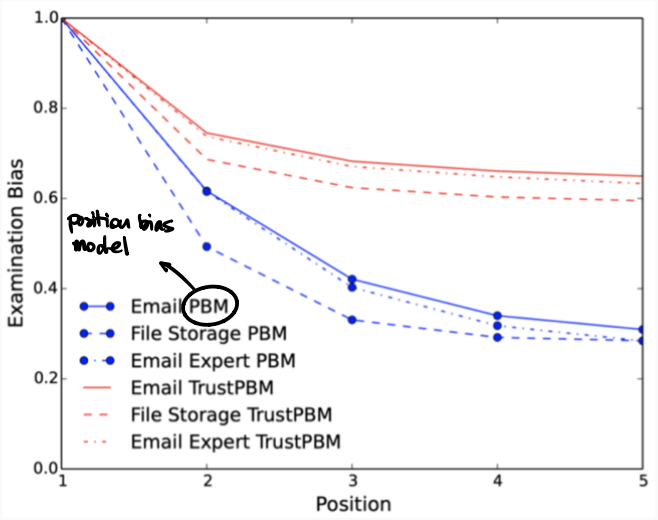
$$= \sum_{d_i \in D} \frac{\epsilon_k^+}{\epsilon_k^+ + \epsilon_k^-} \cdot \frac{\lambda(\text{rank}(d_i \mid f_\theta, D))}{P(o_i = 1 \mid R, d_i)} \cdot c_i.$$

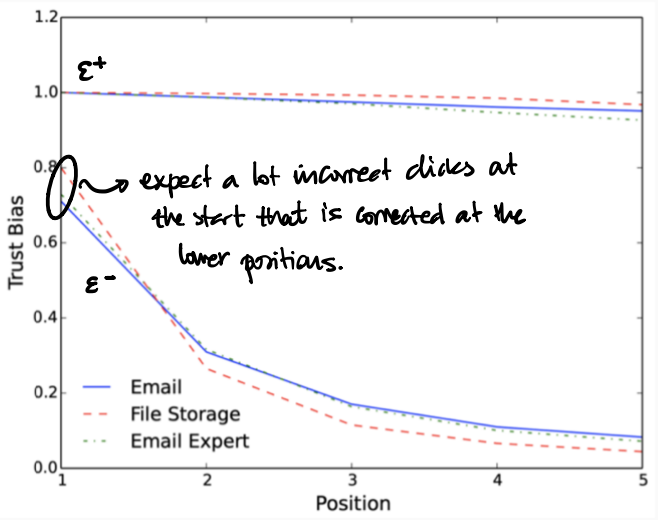
probability that this was actually relevant

everything we have seen so far

The ϵ values can **not be inferred** through **randomization experiments**, but can be estimated through **EM-optimization**.

Disentangled Examination and Trust Bias





If trust bias is **not modeled separately**, then the estimated examination bias will be affected by it. This may explain why the **performance gains** are **somewhat limited**.

By introducing TrustPBM, we have manage to spread out clicks at other positions (for the PBM only model, position 1 is pretty doubling clicks than position 2).

↳ This is done by documents getting clicked less if the document is deemed not relevant.

Part 7 :

Practical considerations

Practitioners of counterfactual LTR systems will run into the problem of **high variance**.

High variance can be due to many factors:

- Not enough training data
- Extreme position bias and very small propensity
- Large amounts of noisy clicks on documents with small propensity

The usual suspect is one or a few data points with extremely small propensity that overpower the rest of the data set.

If you divide by number close to 0, then your score will be massive and ranking is over-powered

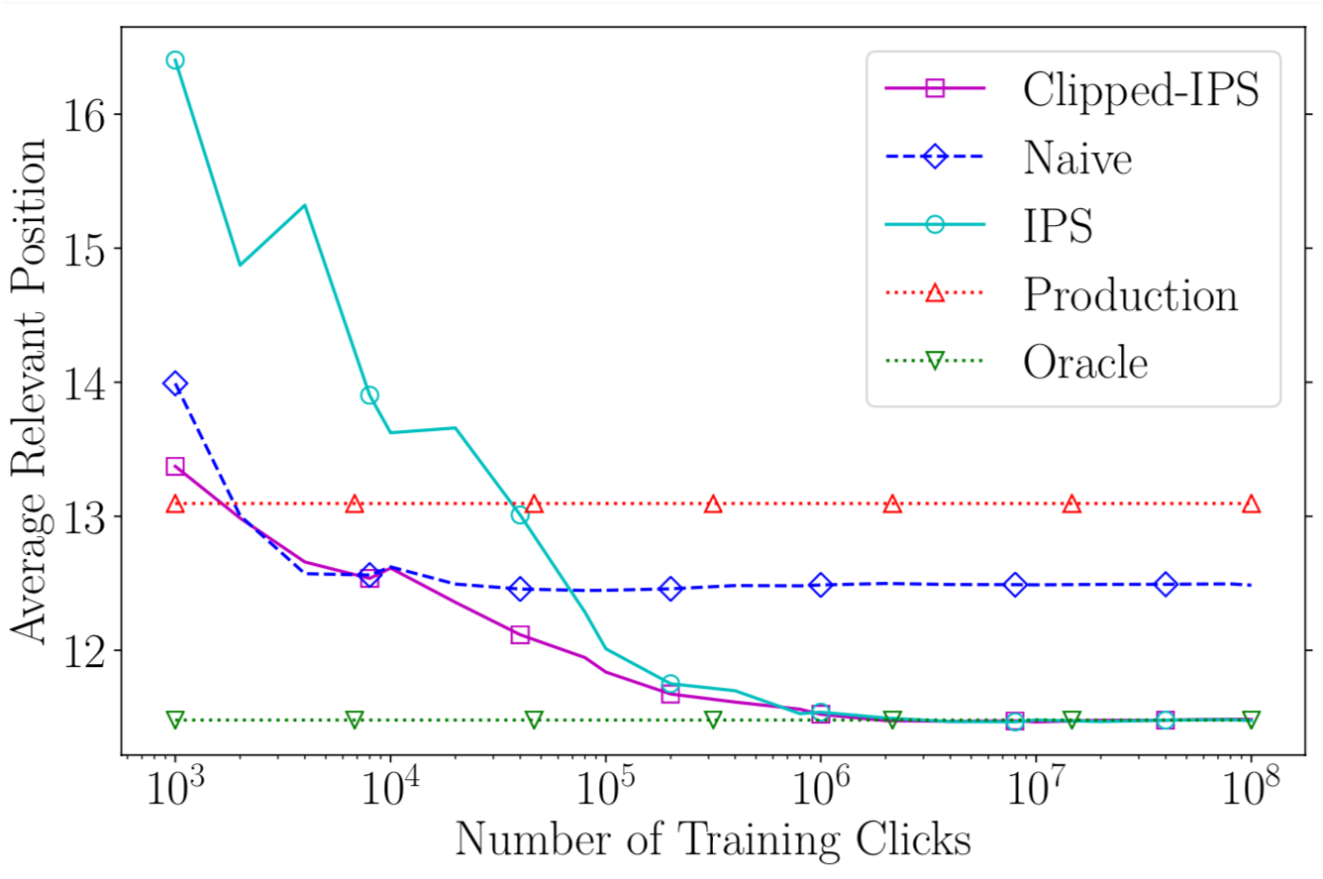
A typical solution to **high variance** is to apply **propensity clipping**.

Propensity clipping: Bound the *propensity*, to prevent any single sample from overpowering the rest of the data set:

$$\Delta_{\textit{Clipped-IPS}}(f_\theta, D, c) = \sum_{d_i \in D} \frac{\lambda(\textit{rank}(d_i \mid f_\theta, D))}{\max\{\tau, P(o_i = 1 \mid R, d_i)\}} \cdot c_i.$$

This solution trades off bias for variance: it will introduce some amount of bias but can substantially reduce variance.

Note that when $\tau = 1$, we obtain the biased naive estimator.



Click models:

DBN → dynamic bayesian networks

CoEC → clicks over expected clicks