Online Learning of Optimally Diverse Rankings

Extended Abstract

Stefan Magureanu
KTH Royal Institute of Technology
Stockholm, Sweden
magur@kth.se

Marcus Isaksson
Spotify AB
Stockholm, Sweden
isaksson@spotify.com

Alexandre Proutiere
KTH Royal Institute of Technology
Stockholm, Sweden
alepro@kth.se

Boxun Zhang
Spotify AB
Stockholm, Sweden
boxun@spotify.se

ABSTRACT

Search engines answer users’ queries by listing relevant items (e.g., documents, songs, products, web pages, ...). These engines rely on algorithms that learn to rank items so as to present an ordered list maximizing the probability that it contains relevant item. The main challenge in the design of learning-to-rank algorithms stems from the fact that queries often have different meanings for different users. In absence of any contextual information about the query, one often has to adhere to the diversity principle, i.e., to return a list covering the various possible topics or meanings of the query. To formalize this learning-to-rank problem, we propose a natural model where (i) items are categorized into topics, (ii) users find items relevant only if they match the topic of their query, and (iii) the engine is not aware of the topic of an arriving query, nor of the frequency at which queries related to various topics arrive, nor of the topic-dependent click-through-rates of the items. For this problem, we devise LDR (Learning Diverse Rankings), an algorithm that efficiently learns the optimal list based on users’ feedback only. We show that after $T$ queries, the regret of LDR scales as $O(N \log(T))$ where $N$ is the number of all items. This scaling cannot be improved, i.e., LDR is order optimal.

CCS CONCEPTS

• Computing methodologies → Sequential decision making: Online learning settings; • Information systems → Content ranking;

KEYWORDS

Learning to rank; multi-armed bandits; online learning; diversity

1 INTRODUCTION

Search engines have become a critical component of many online services. They answer users’ queries by listing relevant documents available online or in the catalog of available products. These engines rely on algorithms that learn to rank items (e.g., documents, songs, products, web pages, ...) so as to present an ordered list maximizing users' satisfaction, i.e., maximizing the probability that there exists a relevant item in the list. One of the main challenges in the design of learning-to-rank algorithms stems from the fact that queries often have different meanings for different users. For example, the meaning of “happy music” may significantly differ across users, the query “jaguar” can be related to cars, the animal, a sport team, etc. In absence of any contextual information about the query, one often has to adhere to the diversity principle, i.e., to return a list covering the various possible topics or meanings of the query. Ideally, one would wish to learn the list having maximal click-through-rate (i.e., the probability that one item in the list is relevant), but the latter clearly depends on both the unknown frequencies of queries related to the various possible topics, and the unknown topic-dependent click-through-rates of all possible items.

In this paper, we consider the online learning-to-rank problem where the optimal list should be learned in an online manner through users’ feedback only. In this problem, the search engine sequentially receives the same query from users interested in various topics. It then returns an ordered list of $L$ items chosen out of $N$ possible items. The user parses the list in order, and clicks on the first item she judges relevant. The engine observes where a click occurred, if any, and refines its displayed list for the next user accordingly. This model of user behavior is known in the literature as the cascading-click model. The relevance of the cascading-click model is showcased in [2], making it one of the most popular single-click model of user behavior in the learning to rank literature ([6], [1],[4],[5] etc.). We assume that the unknown topic of a user’s query is drawn in an i.i.d. manner from a distribution $\phi = (\phi_1, \ldots, \phi_M)$ over the $M$ possible topics. This distribution is also unknown. The click-through-rate of item $k$ depends on the topic of the query, and is equal to $\theta_{km}$ for queries of topic $m$. $\theta = (\theta_{km})_{k,m}$ is also initially

\footnote{The probability that the user clicks on the item if inspected in the displayed list.}
unknown. The objective is to devise an algorithm sequentially selecting lists, depending on past displayed lists and corresponding feedback, and maximizing the cumulative number of clicks over a fixed but large number of successive queries. Equivalently we look for an algorithm minimizing regret defined as the difference between the average cumulative number of clicks under the optimal list and that achieved under the algorithm. We make the following reasonable structural assumptions, justified in settings where diversity is required, i.e., when optimal lists contain items of different topics:

**Assumption 1** The set \( N \) of items is partitioned into \( M \) non-overlapping subsets \( N_1, \ldots, N_M \) where each subset corresponds to items related to a particular topic. This partition is assumed to be known, which essentially means that items have been categorized into topics using previous observations, and available meta-data.

**Assumption 2** A user interested in a topic \( m \) is very unlikely to click on an item related to other topics, i.e., \( \theta_{km} \approx 0 \) if \( k \notin N_m \).

Under the above assumption on \( \theta \), we could characterize the optimal list using a simple greedy procedure if \( \phi \) and \( \theta \) were known. Now in this paper, if \( \theta \) and \( \phi \) are unknown, we devise LDR (Learning Diverse Rankings), an algorithm with low regret, namely scaling at most as \( O(N-L) \log(T) \), which is provably order-optimal. LDR carefully balances exploration and exploitation. Its novelty lies in the fact that it relies on two types of exploration procedure: a first procedure meant to rank all items, and a second aiming at ranking items related to the same topic. We believe that this double exploration provides an elegant and efficient way to quickly identify the optimal list, without actually estimating the \( \theta_{km} \)’s and the \( \phi_m \)’s individually. We show that LDR is order-optimal and that it outperforms existing algorithm on artificial and real-world data (we tested the algorithm on data provided by Spotify, one of the most popular music streaming services.

## 2 Preliminaries

### 2.1 Model

The set \( N \) of \( N \) items is partitioned into \( M \) non-overlapping subsets \( N_1, \ldots, N_M \), each containing items related to a given topic. The decision maker is aware of this partition. We define the mapping \( h : N \to \{1, \ldots, M\} \) such that for any item \( k \), \( h(k) \) denotes its topic, i.e. \( h(k) = m \) iff \( k \in N_m \). The click-through-rate (CTR) of item \( k \) depends on the topic of the query: for a query related to topic \( m \), the user finds \( k \) relevant with probability \( \theta_{km} \). The click-through-rate (CTR) of item \( k \) is unknown, but we assume that \( \theta_{km} = 0 \) whenever \( k \notin N_m \) (a user interested in topic \( m \) finds items not related to \( m \) irrelevant).

Queries arrive at the decision maker sequentially, and the topic of the \( n \)-th query, denoted by \( m(n) \), is unknown. \((m(n))_{n \geq 1} \) is an i.i.d sequence of r.v. with values in \( M = \{1, \ldots, M\} \), and with distribution \( \phi = (\phi_1, \ldots, \phi_M) \), also unknown to the decision maker. Note that the notion of query is loosely defined here: a query may for example corresponds to text strings containing a particular set of keywords (e.g. Christmas). We look at instants or rounds where the engine receives the same query.

After receiving the query in the \( n \)-th round, the decision maker returns an ordered list \( u(n) = (u_1(n), \ldots, u_L(n)) \in N^L \) of \( L \) items (\( L \) is typically much smaller than \( N \)). The user then scans the items in the list in order, and clicks on the first relevant item, if any. If the user clicks on an item, the decision maker observes the slot or position of the corresponding item in the list, and gets a unit reward. The decision maker gets no reward if the user does not click on any item. For the \( n \)-th query, a binary random vector \( X(n) = (\{x_i(n) : i = 1, \ldots, L\}) \) indicates whether the various items in the displayed list are relevant, i.e., \( \Pr[X_i(n) = 1 | u(n) = u, m(n) = m] = \theta_{u,m} \).

Given the sequence of queries and displayed lists, the r.v. \( (X_i(n))_{n \geq 1, l} \) are independent. The average reward of a list \( u \) is then:

\[
\mu_{\phi,\theta}(u) = \sum_{m=1}^M \phi_m \prod_{l=1}^L \theta_{u_m,l} \prod_{i=1}^{l-1} (1-\theta_{u_i,m}).
\]

Throughout this paper we will use the following shorthand notation: \( \mu_{\phi,\theta}(u) = \mu(u) \). We also denote by \( u^* = u^*(\phi, \theta) = \arg\max_u \mu_{\phi,\theta}(u) \) the optimal list (assumed to be unique for simplicity), and we assume without loss of generality that \( u^* \in \{1, \ldots, L\} \)

A sequential decision policy \( \pi \) selects lists depending on the previous selected lists and the corresponding users’ feedback, and we denote by \( u^\pi(n) \) the list chosen under \( \pi \) for the \( n \)-th query. We denote by \( \Pi \) the set of such policies. The problem is to identify \( \pi \in \Pi \) minimizing its regret \( R^\pi(T) \) after \( T \) queries, where:

\[
R^\pi(T) = T \mu(u^*) - E \left[ \sum_{n=1}^T \mu(u^\pi(n)) \right].
\]

### 2.2 Computing the Optimal List \( u^* \)

We establish that when the parameters \( \phi \) and \( \theta \) are known, we can identify the optimal list \( u^* \) using a low complexity recursive greedy procedure. This is possible only thanks to the structural assumption made on \( \theta \) (in absence of such assumption, computing \( u^* \) is NP-hard as stated earlier). We first introduce the success rate \( v(l|u) \) at position \( l \) in an ordered list \( u \) as

\[
v(l|u) = \sum_{m=1}^M \phi_m \theta_{u_m,l} \prod_{s=1}^{l-1} (1-\theta_{u_s,m}).
\]

The success rate \( v(l|u) \) is the probability that the item in position \( l \) is clicked if the displayed list is \( u \). It does not depend on items listed below \( l \) in \( u \) or on the order of items listed ahead of \( l \) in \( u \).

We prove that the following recursive greedy procedure outputs \( u^* \). In what follows, we denote by \( u^l \) the list of length \( l \) with maximal average reward, i.e., \( u^l \) maximizes among all lists \( u \) of length \( l \):

\[
\sum_{m=1}^M \phi_m \sum_{s=1}^l \theta_{u_s,m} \prod_{i=1}^{s-1} (1-\theta_{u_i,m}).
\]

The recursive procedure sequentially constructs lists \( u^l \), \( l = 1, \ldots, L \) of increasing length. We will establish that \( u^L = u^* \).

1. Set \( u^1 = \{k_1\} \) where \( k_1 = \arg\max_{k \in N} \phi_k \theta_{k,m} \).
2. For \( l = 2 \) to \( l = L - 1 \), given \( u^l = \{k_1, \ldots, k_l\} \), denote by \( U(l^u) \) the set of lists of length \( l + 1 \) of the form \( \{k_1, \ldots, k_l, k\} \) for \( k \in N \). Then \( u^{l+1} = \arg\max_{u \in U(l+1)} v(l+1|u) \).

**Proposition 2.1.** The above greedy procedure returns \( u^* \), namely \( u^{L+1} = u^* \).

---

As in [1], we can generalize our model and results to the case where the reward depends on the position of the first relevant item in the list.
3 THE LDR ALGORITHM AND ITS REGRET

The design of the LDR algorithm is guided by the principle of parsimonious exploration. The algorithm maintains a leader, the list believed to be optimal and denoted by \( u^*(n) \) in round \( n \), and displays the leader most of the times. When it explores (when the leader is not displayed), the explored list is just a slight modification of the leader. More precisely, LDR explores items only in the first and last slot of lists. Type-2 exploration, which uses the first slot, determines the order of items within their corresponding topic and Type-1 exploration, which uses the last slot and ensures none of the apparently sub-optimal items can favorably replace the last (and therefore worst, due to the greedy construction if the leader) item in the leader.

**Type-1 Exploration.** LDR explores, in the last slot of lists, items that are not in the leader, but that could advantageously replace the item placed last in the leader (i.e., the weakest item in the leader). The upper confidence bound index for item \( k \) is here defined as a classical KL-UCB index [3]:

\[
d_k(n) = \sup \{ x \in (0, 1) : t_k(n) I(c_k(n), x) \leq f(n) \},
\]

where \( f(n) = \log(n) + 4 \log \log(n) \), \( t_k(n) \) is a counter incremented if \( k \) is displayed and either (i) the true (i.e. not shuffled) leader is displayed or (ii) LDR performs a Type-1 exploration, and \( c_k(n) \) denotes the empirical success rate of item \( k \) (the number of clicks on item \( k \) divided by \( t_k(n) \)). The second type of exploration, together with the fact that in LDR, the leader is updated using the values of the \( c_k(n) \)'s, ensure that \( c_k(n) \) converges the desired success rate of \( k \). More precisely, when \( k \in u^* \), it converges to the success rate of \( k \) when displayed in \( u^* \). When \( k \notin u^* \), it converges to the success rate of \( k \) when displayed in the last slot of \( u^* \).

**Type-2 Exploration.** LDR explores item \( k \) in the first position of the list, to get an estimate of \( \hat{\phi}_{h(k)} \hat{\theta}_{h(k)} \) and hence to be able to rank items related to the same topic.

\[
b_k(n) = \sup \{ x \in (0, 1) : t_k(n) I(\hat{c}_k(n), x) \leq f(n) \},
\]

where \( t_k(n) \) denotes the number of times item \( k \) has been displayed up to round \( n \) in a list where no other item of the same topic was placed before \( k \), and \( \hat{c}_k(n) \) is its empirical success rate for such events – hence \( E[\hat{c}_k(n)] = \hat{\phi}_{h(k)} \hat{\theta}_{h(k)} \). \( b_k(n) \) is an upper confidence bound on \( \hat{\phi}_{h(k)} \hat{\theta}_{h(k)} \).

LDR admits the following regret guarantees:

**Theorem 3.1.** The regret under LDR satisfies:

\[
r_{LDR}(T) = O((N - L) \log(T)).
\]

REFERENCES

[1] Richard Combes, Stefan Magureanu, Alexandre Proutiere, and Cyrille Laroche. 2015. Learning to Rank: Regret Lower Bounds and Efficient Algorithms. SIGMETRICS Perform. Eval. Rev. 43, 1 (June 2015), 231–244. https://doi.org/10.1145/2796334.2745852

[2] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An experimental comparison of click position-bias models. In Proceedings of the 2008 International Conference on Web Search and Data Mining. ACM, 87–94.

[3] Aurélien Garivier and Olivier Cappé. 2011. The KL-UCB Algorithm for Bounded Stochastic Bandits and Beyond. In COLT. 359–376.

[4] Branislav Kveton, Csaba Szepesvari, Zheng Wen, and Azin Ashkan. 2015. Cascading bandits: Learning to rank in the cascade model. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15). 767–776.

[5] Anne Schuth, Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. 2013. Lero: An online learning to rank framework. In Proceedings of the 2013 workshop on Living labs for information retrieval evaluation. ACM, 23–26.

[6] Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. 2016. Multileave gradient descent for fast online learning to rank. In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. ACM, 457–466.

Algorithm 1 LDR (Learning Diverse Rankings)

Initialize for all item \( k : \hat{c}_k(0) = 0.5, \hat{\theta}_k(0) = 0.5, \tau_k(0) = 1, \tau_k(0) = 1, u^*(−1) = 0.

for \( n = 0 \) to \( T \) do

for all \( k \), Update \( \hat{c}_k(n), \hat{\theta}_k(n), \tau_k(n) \) and \( \tau_k(n) \).

end for

for all \( k \), Update \( u^*(n) = u^*(n−1); \)

end for

1. Updating the Leader

if \( n \mod 4 = 0 \) then

for \( l = 1 \) to \( L \) do

\( a_l(n) = \arg \max_{k \notin u^*(n−1)} \hat{c}_k(n); \)

end for

end if

2. Exploiting the Leader.

if \( W = 0 \) or \( W = 3 \) then

if \( W = 3 \) then \( u(n) = \text{shuffle}(u(n)) \) end if

Play \( u(n) \) and continue to round \( n + 1. \)

end if

3. Type-2 Exploration

if \( W = 1 \) then

if \( \exists k \notin u(n) \) and \( l \leq K : b_k(n) > \hat{\theta}_{u_1(n)}(n) \) and \( h(k) = h(u_l(n)) \) then

\( u_{l+1}(n) = u_l(n) \) for \( l = 1 \ldots L - 1, u_1(n) = k. \)

Play \( u(n) \) and continue to round \( n + 1. \)

end if

end if

4. Type-1 Exploration

if \( \exists k : d_k(n) > \hat{\theta}_{u_1(n)}(n) \) and \( h(k) \neq h(u_1(n)) \) then

\( u_1(n) = k. \)

Play \( u(n) \) and continue to round \( n + 1. \)

end if

Play \( u(n). \)