Numerical Facet Range Partition: Evaluation Metric and Methods

Xueqing Liu, Chengxiang Zhai
University of Illinois at Urbana-Champaign
Urbana, IL 61801
xliu93,czhai@illinois.edu

Wei Han, Onur Gungor
WalmartLabs
Sunnyvale, CA 94806
whan,ogungor@walmartlabs.com

ABSTRACT

Faceted navigation is a very useful component in today's search engines. It is especially useful when user has an exploratory information need or prefer certain attribute values than others. Existing work has tried to optimize faceted systems in many aspects, but little work has been done on optimizing numerical facet ranges (e.g., price ranges of product). In this paper, we introduce for the first time the research problem on numerical facet range partition and formally frame it as an optimization problem. To enable quantitative evaluation of a partition algorithm, we propose an evaluation metric to be applied to search engine logs. We further propose two range partition algorithms that computationally optimize the defined metric. Experimental results on a two-month search log from a major e-Commerce engine show that our proposed method can significantly outperform baseline.

Keywords

Faceted search; User search log; Information retrieval models; Non-smooth optimization

1. INTRODUCTION

Querying and browsing are two complementary ways of information access on internet. As one convenient tool to help browsing, faceted search systems have become an indispensible part of today's search engines. Figure 1 shows a standard faceted system on eBay. Upon receiving user query, it displays a ranked list of *facets*: format, artists, sub-genre and price, along with facet values under each facet. These facet values are metadata of the search results. When user selects one or more values, search results are refined by the selection, e.g., in Figure 1, the results (not displayed) only contain box set albums whose genres are Jazz. Faceted browsing is largely popular in search engines for structured entities of the same type[1] (e.g., e-Commerce products, movies, restaurants). In these engines, user often lacks the ability to specify facet values in detail [17]. Therefore, faceted system such as Figure 1 can serve as a convenient tool to elicit user's needs so they can quickly click on the suggested facet values to expand their queries. Faceted browsing is also exceedingly helpful on touch screen devices, where typing query is less convenient than clicking on a facet.

[1]In this paper, we frequently use the term 'entity' to refer to any structured entity. We do not use the term 'item' because the search object we study is more general, e.g., people search. In the experiment part where our data is from e-Commerce engine, we use the term 'product' instead.

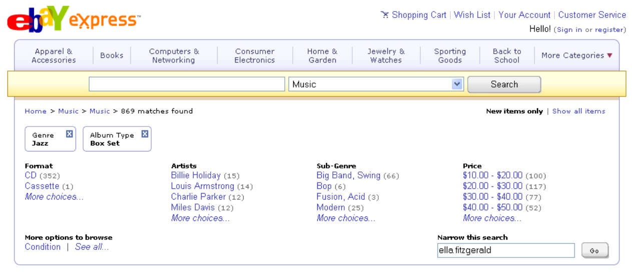


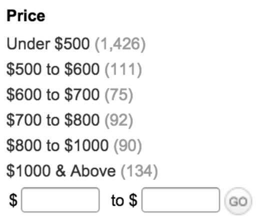
Figure 1: Snapshot of faceted search system on eBay, picture borrowed from Hearst [11] (Figure 8.12, page 195)

A faceted system consists of multiple components, which would naturally decompose its optimization into multiple sub-problems. Existing works have covered quite a few of these sub-problems, e.g., ranking facets or values [28, 14, 15], facet selection [18, 24]. However, we identify one problem which, to the best of our knowledge, has never been formally studied before. Basically, how to suggest values of a *numerical facet* to help user browse the query results? An example of numerical facet is price in Figure 1, where the result albums are partitioned into 5 non-overlapping subsets based on their prices: [0, 20), [20, 30), [30, 40), [40, 50) and [50, ∞). This is equal to saying the results are separated by 20, 30, 40 and 50. So the problem is rephrased as: given user query and results, how to find the best separating values? This problem has a clearly different goal from existing works in faceted system [24, 18, 15, 16, 10, 29]. It can be further decomposed into two parts. First, how to evaluate the quality of a set of separating values (e.g., how good is 20,30,40 and 50?)? Second, if we can find such a metric, how to find separators that optimize it?

Before we delve into answering the two questions, one may wonder why it is even important to study this problem. Arguably, numerical facets are only a small portion of all facets, and why are we unhappy with the current design? If we only consider one search engine, indeed, it usually just contains one or a few numerical facets (e.g., Figure 1). However,

website	issue	example query
amazon.com	one range dom.	refurbished laptop
ebay.com	3 ranges	laptop; camera
walmart.com	one range dom.	socks
bestbuy.com	one range dom.	phone charger
etsy.com	fixed ranges	dress; hair pins
homedepot.com	one range dom.	french door fridge
target.com	one range dom.	card game
macys.com	one range dom.	soap
lowes.com	one range dom.	pillow
kohls.com	one range dom.	socks

Table 1: Issues of suggested price ranges among top-10 shopping websites (as of 02/16/2017).



Price

Under $500 (1,426)
$500 to $600 (111)
$600 to $700 (75)
$700 to $800 (92)
$800 to $1000 (90)
$1000 & Above (134)
$ to $ GO

Figure 2: A specific example of the 'one range dominates' issue (Table 1). The snapshot was taken on 01/21/2016, on Amazon under query 'refurbished laptop'.

notice numerical facets span a wide range of applications. Some of the examples are news search (timestamp), location search (distance), e-Commerce search (price, mileage, rating) and academic search (h-index). So focusing on numerical facet does not make our study narrow. For the latter question, we conduct a case study on the price ranges from top-10 shopping websites that provide price suggestion[2]. We find several issues which we demonstrate in Table 1 and Figure 2. The most common issue is that among multiple suggested ranges, one range contains the majority of results, e.g., Figure 2 shows that range [0, 500) contains 73.9% of the products under query 'refurbished laptop'. It can be expected that the majority users would click on [0, 500), but this only reduces the total number from 1,928 to 1,426, which does not seem very helpful. Another issue we find on one website (www.etsy.com) is it appears to suggest fixed ranges (25, 50, 100) for all queries, so it is not adaptable to different queries such as 'dress' and 'hair pins'. Finally, the price ranges from eBay appear to be the most adaptable among all 10 websites, but seems its number of ranges is fixed to 3, making it unable to adapt to price-diversified categories such as camera. Based on the study results, we believe there is still plenty of room for improving range partition techniques in current search engines.

For the first question, we evaluate our problem by collecting past user search log and defining our evaluation metric on top of it. It is a common practice in information science to evaluate an information system using user's gain and cost [22, 3, 30], where the gain is often estimated as the (discounted) number of relevant entities or clicks in the log [19], and cost is often estimated as the total number of viewed entities in the log [18]. Similarly, evaluation metric for a set of numerical ranges can be defined as user's cost and gain when using the ranges to browse the results. Following existing works in faceted system [18], we fix the gain to 1 and use the cost as our evaluation metric. Under a few reasonable assumptions (Section 4.1), the cost is equal to the rank of the first clicked entity (in the log) in the unique range (among the set of ranges) that contains it.

After the first question is answered, we shift our focus to the optimization problem. From examples in Figure 1 and Figure 2, we can observe that a good partition should (at least) satisfy the following properties: first, it is good for the suggested separators to be adaptable to each query; second, instead of letting one range dominates, the number of entities in each range should be more balanced; third, our partition algorithm should be able to generate any number of ranges, instead of only one specific number like 3. There exists a simple solution that satisfies all three properties: just partition the results into k ranges, so that each range contains the same number of entities. We call this simple method the *quantile* method. Indeed, the quantile method reduces the maximum cost in Figure 2 from 1,426 to 321. But can we further improve it?

In this paper, we propose two range partition algorithms. The idea is to collect a second search log and use it for training, to help improve the performance on the search log for evaluation. In the first proposed method, training data is used for estimating the expected click probabilities in the testing data, then the range is computed by optimizing the expected cost using dynamic programming; in the second method, we propose to parameterize the problem and optimize the parameters on the training data. We conduct experiments on a two-month search log collected from Walmart search engine. Results show that our method can significantly outperform the quantile method, which verifies that learning is indeed helpful in the range partition problem.

2. RELATED WORK

During the past decades, researchers design different interfaces for faceted search and browsing. They include faceted system that displays one facet [24] and k facets [18, 29], where the facet selection is based on ranking. Due to the heterogeneity of entity structures on the web, facets ranking can be classified as ranking facet [4], ranking facet values [14] and ranking (facet, value) pairs [15]. There are also faceted systems which support image search [28] and personalized search [16]. To the best of our knowledge, we have not found any existing literature that explains how to suggest numerical ranges that are adaptable to user queries.

It is a common practice to evaluate search engine using user's gains and costs [13, 19, 3, 30]. Existing approaches would define a system's utility as the difference between user's gain and cost [22, 19], or they would evaluate gain and cost separately [3, 30]. Meanwhile, existing works in faceted systems have also defined metrics for self evaluation [18, 15, 4]. [18] defines the metric as rank of the relevant document after user selects some facets; [15] instead defines it as the total number entities after user selects facets. Between the two, we believe the former one better reflects the actual user cost, so we choose to use it in our metric (Section 4.2), although the latter one is easier to compute.

Since faceted system is an interactive environment, it is usually impossible to collect the actual user behavior on the system to test. As a result, almost all the evaluation in faceted system have to rely on making assumptions to approximate user behavior [24, 18, 15]. For example, [18] tests two assumptions: (1) user would (conjunctively) select all facets that helps to reduce the rank of relevant document; (2) user would only select the facet that reduces the most of

[2]Ranking is based on the website traffic statistics from www.alexa.com as of 02/16/2017.

this value. [24] assumes the user would follow the behavior they estimated from 20 users in a pilot study on a different environment. [32] assumes the probability for user to select each facet is proportional to the semantic similarity between the facet and the relevant document. Unlike [32], our assumption in Section 4.1 only relies on user's discminative knowledge on facet values, and unlike [18], we do not make further assumptions on user's knowledge about data distribution. So our work relaxes the assumptions made by previous works.

Our problem is remotely related to generating histograms for database query optimization [12, 1, 20]. Different from our query adaptive ranges, histograms are used for data compression so they are fixed for all queries. Same as our first method (Section 5.1), Jagadish et al. [12] also uses dynamic programming, although for a different optimization goal. Recently, [1] leverages an approximation technique and is able to replace DP with a linear time algorithm. However, this approximation technique is not applicable to our case, simply because we have a different optimization goal. Our first method would remain a super-cubic running time.

3. FORMAL DEFINITION

We formally define the numerical range partition problem and introduce notations that we will use throughout the rest of the paper. Suppose we have a working set of entities $E = \{e_1, \cdots, e_{|E|}\}$ that user would like to query on. Each entity $e \in E$ is structured, meaning it contains one or multiple facets. For example, facet values of one specific laptop entity is: Brand=Lenovo, GPU=Nvidia Kepler, etc. Here 'Brand' and 'GPU' are facets; '500GB' and 'Nvidia Kepler' are facet values. Facets are often shared by entities in E, but some facets are only shared by a subset of E. For example, some laptops do not have a GPU.

At time i user enters a query q^i, search engine retrieves a ranked list of entities $E^i \subset E$. Our problem asks, for one specific numerical facet (e.g., price), how to find a set of separating values for that facet? In order for this problem to exist, at least a significant number of entities in E^i should contain the specified numerical facet. From now on, we will just assume this facet is already specified and all the discussions are about this facet.

We further assume the number of output ranges is given as an input parameter k. k is defined by either system or user. We believe it is important to have control on the number of output ranges. Indeed, it would be bad experience if the user wants to see fewer ranges but receives an unexpectedly long list. Also, it is unfair to compare two partition algorithms if they generate different number of ranges, e.g., [0, 100), [100, 200), [200, 300), [300, 400) is almost certainly better than [0, 200), [200, 400) because user can always use the former one to zoom into a better refined results.

To summarize the input and output of a range partition algorithm: **Input**: (1) number of output ranges k; (2) query q^i; (3) ranking algorithm and ranked list E^i; numerical facet value of each $e \in E^i$, denoted as $v(e)$ (if e does not have the facet, $v(e)$ is empty); rank of each $e \in E^i$, denoted as $rank(e)$. **Output**: $k-1$ separating values $S^i = (s_1, \cdots, s_{k-1}) \in \mathbb{R}^{k-1}$, where $s_1 < \cdots < s_{k-1}$.

4. EVALUATION

In this section, we propose and formally define our evaluation techique and metric for range partition algorithms.

4.1 User Behavior Assumptions

Evaluation in IR is mainly divided into two categories: first, conduct user studies such as laboratory based experiments or crowdsourcing; second, collect search log of real user engagements in the past, define evaluation metrics on top of the log and use them to compare different systems' performances, also called Cranfield-style evaluation [25]. Since the former approach is expensive and not easy to reproduce, we choose the latter one, which is also the more frequently used approach of evaluating faceted systems in existing work [18, 29]. Collected log consists of queries, and we only keep queries with at least one clicked entity. Also in this paper, we assume user click is the only relevance judgement. That is, *relevant entity is equal to clicked entity.*

But it is not straightforward how to obtain a reusable search log for evaluating range partition algorithms. On the one hand, it is impossible for the search log to have enumerated all possible range sets. On the other hand, unlike reusable relevance judgements in Cranfield experiments, it is difficult to infer which range user would select out of one set based on her selection out of a different set in the log. Fortunately, existing work in faceted search [18] provides a hint to this challenge. It assumes user would be able to select the facet value that is most helpful in reducing the rank of the relevant document, then sequentially browse the refined document list until finding the relevant document. In other words, it assumes *user has some partial knowledge in which facet value is more relevant before actually seeing the relevant document.* Similarly, we can assume:

- **Assumption 1**. User would select the range that contains the relevant entity;
- **Assumption 2**. After selecting the relevant range, user would sequentially browse the refined results until reaching relevant entity;

Assumption 1 only requires user has a discriminative knowledge on the numerical facet (e.g., knowing which price range is more relevant); while Assumption 2 is among the basic assumptions of information retrieval [7, 23].

There are cases where our assumptions may not be true. For example, if the numerical value of relevant entity is near the borderline, it is difficult for the user to choose between the two ranges. However, we find them reasonable to make when our main purpose is to perform comparative studies between different partitioning algorithms. This is because if there is any bias introduced through these assumptions, the bias is unlikely favoring any particular algorithm.

4.2 Evaluation Metric

It is a common practice in information science to evaluate a system's performance using user's cost and gain. Previous evaluation methods can be categorized into three groups. First, evaluate cost and gain separately [30]. Since our goal is comparative study, this approach is not informative enough. Second, use the difference between gain and cost, e.g., gain divided by cost [22]. Although thereby we only have one score, this approach will likely introduce bias since gain and cost may not be on the same scale. The third approach is to control one variable while examining the other. In our problem, it is easier to control and measure gain, since it can be simply defined as the number of entities user has clicked so far. Meanwhile, reusing search log has added challenge to measuring cost of faceted system. Although cost in a no-facet search engine can be simply estimated as number of entities above relevant ones; in engines with faceted system, however, if the number of relevant entities (i.e., user clicks) is larger than 1, this definition is ambiguous, because there are many possible cases of user activity, and cost in each case is different [3].

On the other hand, if the number of clicked entities is fixed to 1, i.e., we only consider the first clicked entity in the log, it is easy to obtain an unabmiguous definition for cost: for any suggested ranges, there will be one and only one range that contains the relevant (clicked) entity. So if we apply the two assumptions in Section 4.1, user would first select that unique range, then sequentially browse entities in that range until finding the first relevant entity. Therefore, the cost is equal to the rank of the first clicked entity in its unique range. We assume that after user selects any range, relative ranks of entities inside that range do not change. Therefore the cost is well defined by the initial search results list L, the suggested range $S \in \mathbb{R}^{k-1}$ and the first clicked entity e, we denote this value as $\textit{Refined-Rank}(e, L, S)$.

Now we are ready to define the evaluation metric for a range partition algorithm A. At time i in the log, user enters query q^i, search engine returns ranked list E^i and user first clicked on entity e^i. Suppose algorithm A suggests ranges $S^i = (s_1, \cdots, s_{k-1})$ for each query q^i in the log, we evaluate algorithm A's performance using the *averaged refined rank* metric, or ARR for short:

$$\begin{aligned} RR_i &= \textit{Refined-Rank}(e^i, E^i, S^i) \\ ARR &= \frac{1}{n}\sum_{i=1}^{n} RR_i \end{aligned} \quad (1)$$

RR$_i$ and ARR will serve as the evaluation metric for all range partition algorithms throughout this paper. Since ARR only considers user's engagement before the first entity click, it remains a challenge how to measure the performance of a range partition algorithm in the whole session. We leave it for future work.

5. METHODS

In Section 1, we discuss the quantile method, which partitions E^i into k equal sized ranges. This approach is also used in database system for observing underlying data distribution or data compression (where it is called *equi-depth binning* [20]). Figure 2 shows that the quantile method performs reasonably well. However, quantile method is a simple, rule-based method without leveraging extra information. Suppose we are allowed to use any information we can collect, can we do better than quantile method?

An idea is to collect another search log for training, since it can help us make better estimation on the testing (evaluation) data. In this section, we propose two methods to leverage the training data.

5.1 First Method: Dynamic Programming

Since we have defined ARR (Equation 1) as our evaluation metric and the smaller the better, our range partition algorithm should try to minimize ARR and RR$_i$. Imagine if the clicked entity e^i was known, minimizing RR$_i$ means we should make one range only contain e^i itself. RR$_i$ in this imaginary scenario is equal to 1. In reality, although the clicked entity is not known, we can estimate the click probability using the extra search log (i.e., training data). Denote the estimated click probability on entity e as $p(e)$ (so that $\sum_{e \in E^i} p(e) = 1$). Then the expected RR$_i$ for $S = (s_1, \cdots, s_{k-1})$ is:

$$\mathbb{E}_S[RR_i] = \sum_{e \in E^i} p(e) \times \textit{Refined-Rank}(e, E^i, S) \quad (2)$$

So our first method is: for each query q^i, to suggest $S^i = \arg\min_{S \in \mathbb{R}^{k-1}} \mathbb{E}_S[RR_i]$.

To minimize Equation 2, first notice that although $\mathbb{R}^{k-1}$ is continuous, we actually only have to search for S in a discrete subspace of $\mathbb{R}^{k-1}$. The reason is explained in the following example. Suppose E^i only contains three entities (ordered by rank) e_1, e_2 and e_3. $v(e_1) = 100; v(e_2) = 200, v(e_3) = 300$; estimated probabilities are $p(e_1) = 0.4$, $p(e_2) = 0.3, p(e_3) = 0.3$; finally, $k = 2$, so $S = (s_1)$. Originally, s_1 can be any float $\in (100, 300]$ (if $s_1 \le 100$ or $s_1 > 300$, result only contains one range). However, notice objective function (Equation 2) stays the same for all $s_1 \in (200, 300]$, also for all $s_1 \in (100, 200]$. So we only have to pick $a \in (100, 200]$, and $b \in (200, 300]$ and compare the objective function with $S = (a)$ and $S = (b)$. We pick the mid point for convenience, i.e., $a = 150$ and $b = 250$.

From example above, we can see that in general, minimizing Equation 2 subject to $S \in \mathbb{R}^{k-1}$ is equal to the combinatorial optimization problem of selecting $k-1$ numbers from $|E^i| - 1$ mid points so that their combined S minimizes the objective function. We can, of course, use brute-force search, but the time cost would be $O(\binom{|E^i|-1}{k-1} + |E^i|^3 \log |E^i|)$, where the extra $|E^i|^3 \log |E^i|$ is for sorting and pre-computing $\textit{Refined-Rank}(e, E^i, S)$ for each e in each possible range. When $|E^i|$ is large, this time cost is undesirable. However, this problem has a $O(k|E^i|^2 + |E^i|^3 \log |E^i|)$ time solution using dynamic programming. This is because objective function can be rewritten as the sum of k parts, the k-th part is independent from previous $k-1$ parts (for proof of this, see Appendix A in the longer version of this paper).

One may wonder why we do not use greedy algorithm here. There are two reasons: first, greedy algorithm generally leads to sub-optimal solutions[4]; second, the computational cost of greedy algorithm is $O(k|E^i| + |E^i|^3 \log |E^i|)$, which remains large since it still has to compute ranks of each entity in each possible range.

5.2 A Second Look: Parameterization

In Section 5.1, we propose to suggest S^i that optimizes the expected RR$_i$ for each time i. Yet with access to both training and testing data, we have a second thought: can we build a machine learning model to study this problem?

Take linear regression as an example. Given training data $\{\mathbf{x}^i, y^i\}, i = 1, \cdots, n$, it defines parameter w and b, finds w and b that minimize the square loss on training data, and applies them on the testing data. In our problem, can we define a set of parameters, model ARR as a function of the parameters, find parameters that minimize ARR on training data, which could then be applied on testing data?

At the first sight, there does not seem to exist a very straightforward solution to the parameterization. One may

[3]For example, under one query, user clicked on entity e_a and e_b, and they are in range a and b (different). Case 1: user selects both a and b, browse until finding both e_a and e_b. Case 2: user selects a, browse until finding e_a, unselect a and select b, browse until finding e_b. Case 3: user selects a, browse until finding e_a, select b, browse until finding e_b.

[4]An example: suppose E^i contains four entities (ordered by rank) e_1, e_2, e_3 and e_4. $v(e_1) = 400, v(e_2) = 100, v(e_3) = 200, v(e_4) = 300$, $p(e_1) = p(e_2) = 0.2, p(e_3) = p(e_4) = 0.3, k = 3$. Optimal solution is 1.2 but greedy algorithm's solution is 1.3.

think $S = (s_1, \cdots, s_{k-1})$ can be the parameters. However, we have discussed in Section 1 that it is not a good strategy to use fixed ranges for different queries. On the other hand, we learned that the quantile method performs reasonably well. This sheds light on how we can define the parameters: using the *relative ratio* representation of S, i.e., $R = (r_1, \cdots, r_{k-1}) \in (0,1)^{k-1}$ where $r_1 < \cdots < r_{k-1}, r_0 = 0, r_k = 1$. Given the search results E^i, for any R, we can find the partition S for E^i so the ratio of number of entities in range $[s_{j-1}, s_j)$ most closest approximates, if not exactly equal to $r_j - r_{j-1}$:

$$\Delta r_j := r_j - r_{j-1} \approx \frac{|\{e \in E^i | v(e) \in [s_{j-1}, s_j)\}|}{|E^i|}$$

The R for quantile method is $(1/k, \cdots, k-1/k)$. With this representation, any R corresponds to one point $(\Delta r_1, \cdots, \Delta r_k)$ in the simplex Δ^k.

So we want to ask: among all points in Δ^k, does quantile method generate the best ARR on testing data? If not, can we achieve better ARR on testing data by finding parameter R that minimizes the ARR in training data? In this section we study how to optimize ARR with respect to R.

5.2.1 Optimizing ARR with Respect to R

It is difficult to directly optimize ARR, because same as many evaluation metrics in IR (e.g., NDCG[27], MAP[31]), ARR is a non-smooth objective function with respect to parameter R. Indeed, if the relevant entity is near the boundary, and we change R with a small enough value $\epsilon \to 0$, relevant entity would jump from one range to another, so RR_i would also jump and as a result, ARR cannot stay continuous. An example: suppose E^i only contains three entities (ordered by rank): e_1, e_2 and e_3. $v(e_1) = 100, v(e_2) = 200, v(e_3) = 300$; relevant entity is e_2 and $k = 2$. If we change $R = [0.66]$ to $R = [0.67]$, the partition would jump from $\{\{e_1\}, \{e_2,e_3\}\}$ to $\{\{e_1,e_2\}, \{e_3\}\}$, and RR_i would jump from 1 to 2.

Non-smooth optimization. In order to optimize the non-smooth ARR, first notice that ARR can be non-smooth everywhere, instead of only at a few points[5]. There exist a few derivative-free algorithms for solving optimization problem in this case. Two of them are Powell's conjugate direction method [6] and Nelder-Mead simplex method [21], we will discuss more about this topic in Section 6.

Time complexity to directly optimize ARR. Time complexity of directly optimizing ARR with the above non-smooth optimization algorithms is *at least* $O(N_{eval}T_1)$, where T_1 is the average time cost to compute ARR on one specific point, and N_{eval} is the number of such points we have to compute (number of function evaluations). In other words, N_{eval} depends on the efficiency of non-smooth optimization algorithm, and T_1 depends on the size of the data. We can observe from Equation 1 that $T_1 = O(nm \log m)$, where n is the number of queries in the training data, and m is the average number of retrieved entities $|E^i|$ for each query q^i. This is because whenever the optimization algorithm goes to a new point R, we have to recompute the ARR from scratch. To explain in more detail: whenever we are at a new point R, every RR_i in Equation 1 may have changed (as we discussed above, a small enough change in R can lead to a significant change in RR_i), so we have to recompute the RR_i in every single query; every such recomputation takes $O(m \log m)$, which is for sorting entities in the range that contains relevant entity to compute its refined rank.

[5]Therefore our optimization cannot be solved in the same as Lasso [26] which uses sub-gradient descent.

In summary, the time complexity for any optimization algorithm to directly optimize ARR is $O(N_{eval}nm \log m)$. In real world search engines, both m and n can be very large. On the other hand, we are not aware of theoretical estimation on N_{eval}, but previous work has provided empirical results. Table 1 to 3 of [9] show examples of N_{eval} in Nelder-Mead, and Table 2 of [2] shows examples of N_{eval} in Powell's method. Empirically, N_{eval} for lower dimensional problems (k ranges from 2 to 10, which is the case for numerical range partition) usually ranges from 100 to 1,500.

5.2.2 Optimizing the Surrogate Objective Function

As discussed in Section 5.2.1, the algorithm for directly optimizing ARR takes $O(N_{eval}nm \log m)$, which is time consuming when N_{eval}, n, m are all very large. In this section, we propose a three-step process that turns ARR into a surrogate objective function. We propose to optimize the surrogate function instead of directly optimizing ARR, so that time cost is significantly reduced.

Step 1: Normalization. First, for each query q^i, we normalize RR_i by the total number of retrieved entities E^i:

$$\overline{RR}_i = \frac{RR_i}{|E^i|} = \frac{\textit{Refined-Rank}(e^i, E^i, R)}{|E^i|}$$

Refined-Rank(e^i, E^i, R) is the same as *Refined-Rank* (e^i, E^i, S) where S are the separating values closest to R (see beginning of Section 5.2).

Step 2: Upper bound. By definition (Section 4.2), *Refined-Rank*(e^i, E^i, R) is bounded by the total number of entities in the unique range that contains relevant entity e^i. Denote this range as $[s_{j_i}, s_{j_i+1})$:

$$\overline{RR}_i \leq \frac{|\{e \in E^i | v(e) \in [s_{j_i}, s_{j_i+1})\}|}{|E^i|} \tag{3}$$

Step 3: Limit approaching infinity. Notice as $|E^i|$ goes to infinity, the R.H.S. of Inequality 3 approaches $\Delta r_{j+1} = r_{j+1} - r_j$ (see beginning of Section 5.2). If we denote z^i as the ratio of number of entities smaller than or equal to $v(e^i)$[6], this limit is rewritten as:

$$C^i(R) := \Delta r_{j_i+1} = \sum_{j=1}^{k} \mathbb{1}[r_{j-1} \leq z^i \leq r_j] \times \Delta r_j$$

The averaged limit over $i = 1 \cdots, n$ is defined as $C_n(R)$:

$$\begin{aligned} C_n(R) &= \frac{1}{n}\sum_{i=1}^{n} C^i(R) \\ &= \sum_{j=1}^{k} \Delta r_j \times (F_n(r_j) - F_n(r_{j-1})) \end{aligned} \tag{4}$$

Where $F_n(r) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}[z^i < r]$ for $r \in [0,1]$ is exactly equal to the empirical conditional distribution function (CDF) of z^i. Second equation in (4) follows from simple math. So instead of directly optimizing ARR, we propose to optimize $C_n(R)$ instead.

Time complexity to optimize $C_n(R)$. We can see the time cost for optimizing $C_n(R)$ is largely reduced compared with ARR. This is because the empirical CDF $F_n(r)$ can be

[6]For example: suppose E^i only contains four entities (ordered by rank): e_1, e_2, e_3 and e_4. $v(e_1) = 100, v(e_2) = 300, v(e_3) = 200, v(e_4) = 400$; relevant entity is e_2. In this example, $z^i = \frac{3}{4}$.

first computed and cached using Algorithm 1. After $F_n(r)$ is cached, at any new point R where the non-smooth optimization algorithm needs to re-compute $C_n(R)$, it only have to obtain the cached $F_n(r)$ from X_{sorted} and Y (output from Algorithm 1) for $r = r_1, \cdots, r_{k-1}$ then apply Equation 4. To obtain cached $F_n(r)$, we first use binary search on X_{sorted} to find the index i of r, then return $Y[i]$ as $F_n(r)$. Therefore, time complexity for each of the N_{eval} function evaluation is reduced to $O(k \log n_0)$.

Time costs for caching $F_n(r)$ are listed in Algorithm 1. In summary, the total time complexity for caching + optimizing $C_n(R)$ is $O(nm + n_0 \log n_0 + n \log n + n_0 \log n + N_{eval} k \log n_0)$. n_0 is the number of unique $r'_j s$ in the log, so $n_0 < |X_{ct}|m < nm$.

Algorithm 1: Caching Empirical CDF $F_n(r)$

```
$X_{ct} \leftarrow \emptyset$;                    // Set of unique $|E^i|$
$X \leftarrow \emptyset$;                         // Set of unique $r_j$'s
$Y \leftarrow []$;             // $F_n(r_j)$ values of all unique $r_j$'s
$Z \leftarrow []$;                                      // All $z^i$'s
for $i = 1, \cdots, n$ do
    if $|E^i| \notin X_{ct}$ then
        $X_{ct} \leftarrow X_{ct} \cup \{|E^i|\}$;
        for $j = 1, \cdots, |E^i| - 1$ do
            $X \leftarrow X \cup \{\frac{j}{|E^i|}\}$;
        end
    end
    $count \leftarrow 0$;
    for $e \in E^i$ do
        if $v(e) \le v(e^i)$ then
            $count \leftarrow count + 1$;                  // $O(nm)$
        end
    end
    $z^i \leftarrow count/|E^i|$;
    Append $z^i$ to the end of $Z$;
end
$n_0 \leftarrow |X|$;
$X_{sorted} \leftarrow sort(X)$;                   // $O(n_0 \log(n_0))$
$Z_{sorted} \leftarrow sort(Z)$;                      // $O(n \log n)$
for $i = 1, \cdots, |X_{sorted}|$ do
    $x \leftarrow X_{sorted}[i]$;
    $Pos \leftarrow BinarySearch(Z_{sorted}, x)$;    // $O(n_0 \log n)$
    $y \leftarrow Pos/n$;
    Append $y$ to the end of $Y$;
end
return $X_{sorted}$ and $Y$;
```

5.2.3 *Bounds on* $C_n(R)$

The Dvoretzky-Kiefer-Wolfowitz inequality [8] bounds the probability that the empirical CDF F_n differs from the true distribution F. Following the DKW inequality, we are able to prove a few bounds on $C_n(R)$. These bounds provide useful insights on the convergence rate and sample complexity of $C_n(R)$ on large scale datasets. We show them in Appendix B in the longer version of this paper.

5.3 Learning to Partition with Regression Tree

In Section 5.2 we propose to optimize $C_n(R)$ subject to the ratio parameter R, and apply it to the testing data. This means all queries in testing data shares the same R. If they can have different R's, can we further improve the results?

To differentiate each query, we define a feature vector $\mathbf{x}^i \in \mathbb{R}^d$ for query q^i. For example, $\mathbf{x}^i$ can be q^i's low dimensional representation using the latent semantic analysis (LSA). A heuristic solution, for example, is to replace R with $R^i = \beta^T \mathbf{x}^i$ in each query, and optimize C_n subject to β^T. However, C_n defined this way is much harder to optimize, because Δr_j is now different for each query, so $F_n(r)$ can no longer be pre-computed and cached.

This observation implies that we should try to make each R^i shared by at least a significant number of queries. The best machine learning method under this setting (that we are aware of) is the regression tree (CART [5]). In a regression tree, all queries inside each leaf node t share the same parameter R_t.

Training of a regression tree would recursively split examples in the current node. In each node, it chooses the dimension $j \in [d]$ and the threshold θ so that splitting by whether $\mathbf{x}^i_j > \theta$ minimizes the sum of mean square error (MSE) on each side. The overall goal of regression tree is to minimize the square error on training data. On the other hand, our goal is to minimize the ARR on training data, and because ARR is hard to compute, we minimize $C_n(R)$ instead (Section 5.2.2). Therefore, we can build a regression tree for our problem where the splitting criterion at each node is to select $j \in [d]$ and θ to minimize the sum of minimum $C_n(R)$ on each side.

- **Splitting criterion 1**. Select dimension and separating value that minimizes C_n (Equation (4));

However, it is interesting to observe how minimizing MSE resembles minimizing C_n. Imagine two different splits on the same data. Suppose that with one split, data is perfectly separated into two clusters; with the other split, however, data is still well mixed. The former one would have smaller MSE. It would also have smaller C_n, since R in each cluster is highly fitted in a small region. Therefore, we propose to use MSE as an alternative splitting criterion:

- **Splitting criterion 2**. Select dimension and separating value that minimizes the mean square error;

Criterion 2 does not compute the parameter R, so after the tree is constructed, we need extra time to compute R_t for each node t. But even so, Criterion 2 is orders of magnitude faster than Criterion 1. This is because, on the one hand, while Criterion 1 needs to reconstruct a new tree for every k, criterion 2 only needs to build one tree the whole time. On the other hand, time cost of criterion 2 in constructing each tree is significantly less than criterion 1, because computing MSE is much faster than minimizing C_n.

An important step in regression tree [5] is the minimal cost-complexity pruning. First, a full (overfitting) tree is grown, then the algorithm goes through 5 fold cross validation to select the optimal pruning for the fully grown tree. We apply the same pruning strategy for Criterion 1 and 2, where we use the 0.5 SE rule to select the optimal tree.

5.4 Testing Time and Rounding

Testing complexty. For each q^i, testing time for our first method (Section 5.1) is $O(k|E^i|^2 + |E^i|^3 \log |E^i|)$ Our second method (both Section 5.2.1 and Section 5.3) takes constant time to generate R^i, but the R^i still needs to be converted back to S^i. There are two ways to do this: first, sort E^i by $v(e)$, which takes $O(|E^i| \log |E^i|)$; second, apply the k-th smallest element algorithm[7], which takes $O(k|E^i|)$. When $|E^i|$ is large, this step can also be time consuming.

[7]e.g., quickselect `https://en.wikipedia.org/wiki/Quickselect`

However, we have to scan E^i for at least one time anyway. This is because after S^i is generated, for all $e \in E^i$ we need to find the range that contains it. So second method does not increase time complexity with respect to $|E^i|$.

Rounding. To better user experience, we need to generate easy-to-read ranges, therefore we may need to round the floating numbers in S^i. Rounding precision depends on the application scenario. For price of products, users may be expecting more friendly designs, thus they may prefer 'Below 150' to 'Below 149.7'. In other applications such as distance, users may accept higher precision such as 'Below 11.7 miles'. The rounding precision can also be tuned as a parameter.

6. EXPERIMENTS

In this section, we conduct comparative experiments on the quantile method and our two methods to answer the question in Section 1 and Section 5, i.e., can we leverage previous search logs to improve the results on test collection?

6.1 Dataset

Since no existing work has studied our problem setting (Section 3), we have to construct our own dataset. We collect a two-month search log from `www.walmart.com` between 2015/10/22 and 2015/12/22. Since the size of the entire log is intractable on a single machine, we only keep the data from two categories: 'Laptop' and 'TV', because they are among the categories with the most traffic. Our data contains multiple numerical facets, e.g., screen size and memory capacity. We select the price facet for experiment, because most product (larger than 90%) contains this facet. Although price can vary from time to time, we assume it is fixed within a short period of time, so each product in our data can only contain one price.

For each category, we separate the earlier 70% as training data and latter 30% as testing data (according to timestamps). After the separation, Laptop contains 2,279 training queries and 491 testing queries, TV contains 4,026 training queries and 856 testing queries. Data structure under each query is the same as the input described in Section 3, plus the ground truth of which entity is clicked (relevant).

6.2 Experimental Results

We compare ARR generated by four methods on testing data: `quantile`: for each query, quantile method generates k ranges so each range contains the same number of products; `dp`: for each query, our first method (Section 5.1) generates k ranges which optimize expected RR_i (Equation 2) using DP; `powell`: (Section 5.2) first use Powell's method to find R by optimizing $C_n(R)$ (Equation 4) on training data, then apply the same R to all queries on testing data; `tree`: find different R's using regression tree (Section 5.3) and apply the tree to all queries on testing data.

Of the four methods, `quantile` does not leverage training data; we use all training data to estimate $p(e)$ for `dp` (which we discuss in details in Section 6.2.5), so `dp`, `powell` and `tree` use the same amount of training data.

6.2.1 *Overall Comparative Study*

Table 2 shows the ARR of the four methods. For every method we report the best tuned ARR by varying its parameters. We can see that the overall performance of `tree` is the best among all; `powell` and `dp` are next, with `powell` slightly better in Laptop and `dp` slightly better in TV; `quantile` has the worst performance in Laptop, and similar performance as `powell` in TV. On the other hand, if we vertically compare Laptop vs. TV in each method, we can see that `quantile` and `dp` are slightly better in TV than Laptop, while `powell` and `tree` are the opposite.

We run T-test between each pair of methods in `quantile`, `dp` and `tree`. We skip T-test on `powell` because `tree` generalizes `powell`, and Table 2 shows `tree` always outperforms `powell`. From Table 2 we can see that T-test results are different in Laptop and TV. For Laptop, `tree` significantly ourperforms `quantile` and `dp` (except for `tree` vs. `dp` when $k = 2$, which may be because performance of parameterized method is hurted when degree of freedom = 1); for TV, however, T-test results are not significant; also, `dp` vs. `quantile` are not significant.

These analyses indicate `tree` and `powell` perform especially well on Laptop data. So what causes the difference between TV and Laptop?

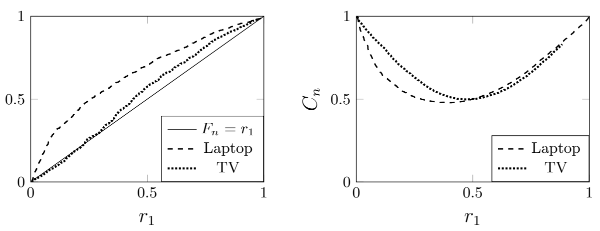


Figure 3: F_n and C_n for Laptop and TV when $k = 2$

To answer this question, we need to find out how `powell` and `tree` really works. Recall that `powell` optimizes $C_n(R)$, which is computed from $F_n(r)$ (Equation 4). When $k = 2$, that is, $R = (r_1)$, we are able to plot $C_n(R)$ and $F_n(r)$ as a function of r_1. We show the two plots in Figure 3. From Figure 3 we can see: F_n of TV is very close to linear, and (consequently) C_n of TV is very close to a quadratic function whose minimum point is $r_1 = 0.5$ (Indeed, by plugging $F_n(r_1) = r_1$ into Equation 4 we get $C_n(r_1) = 2r_1^2 - 2r_1 + 1$). For general k, the minimum point R found by these algorithms is almost equal to `quantile` method. In other words, `quantile` almost reaches the optimal R on training data in terms of $C_n(R)$.

But our final goal is to optimize ARR *on testing data.* Has `quantile` method also reached the optimal R on testing data in terms of ARR? To find out the true optimal R on testing data, we perform grid search. We exhaustively enumerate $r_j (j = 1, \cdots, k-1)$ over all candidate values (i.e., X_{sorted} in Algorithm 1); at each point, we evaluate the true ARR on testing data, and return the minimum value we find. Time complexity of this exhaustive search is $O(\binom{n_0}{k-1})$. When $k > 4$, it becomes intractable. We thus only compute the results for $k \leq 4$[8] and show them in Table 3 (`exhaustive`), compared with ARR of `quantile` method. From Table 3 we can see that `quantile` method indeed almost achieves optimal. So it is difficult for `tree` and `powell` to outperform `quantile`.

6.2.2 *Comparative Study on Non-smooth Optimization Methods*

In this section we conduct comparative study on the performance of different non-smooth optimization methods. We

[8]Although it seems we can replace exhaustive search with Powell's method, which is efficient thus can be applied to $k > 4$; notice Powell's method can not guarantee finding global optimal like exhaustive search.

		`quant.`	`dp`	`powell`	`tree`	`tree` vs. `dp`		`tree` vs. `quant.`		`dp` vs. `quant.`	
						p	t	p	t	p	t
Laptop	$k=2$	33.27	30.15	31.63	**28.00**	0.32	-0.98	9e-3	-1.45	0.15	-1.45
	$k=3$	22.07	21.22	19.95	**17.62**	0.03	-2.18	5e-4	-3.50	0.61	-0.50
	$k=4$	16.76	16.47	15.28	**13.29**	0.02	-2.23	3e-4	-3.63	0.83	-0.20
	$k=5$	13.55	13.43	11.94	**10.72**	0.04	-2.05	3e-4	-3.65	0.92	-0.09
	$k=6$	11.33	11.03	10.15	**9.03**	0.04	-2.02	2e-4	-3.69	0.76	-0.29
TV	$k=2$	31.85	30.99	31.73	**30.78**	0.89	-0.12	0.49	-0.68	0.60	-0.52
	$k=3$	21.30	20.88	21.43	**20.75**	0.89	-0.12	0.60	-0.51	0.69	-0.38
	$k=4$	16.19	15.95	16.30	**15.57**	0.63	-0.47	0.43	-0.78	0.76	-0.29
	$k=5$	13.08	12.83	13.18	**12.62**	0.75	-0.31	0.47	-0.72	0.70	-0.37
	$k=6$	10.95	10.64	10.98	**10.48**	0.76	-0.30	0.37	-0.89	0.57	-0.55

Table 2: Comparative study on the ARR of four methods. The ARR metric can be interpreted in this way: when the number of partitioned ranges is 6, users needs to read 11.33 products in average with `quantile` method; while she only needs to read 9.03 products in average with `tree` method. `dp`, `powell` and `tree` uses the same amount of training data for fair comparison.

	$k=2$	$k=3$	$k=4$
`exhaustive`	31.72	21.27	16.14
`quantile`	31.85	21.30	16.19

Table 3: Optimal ARR vs. `quantile`'s ARR for 'TV'

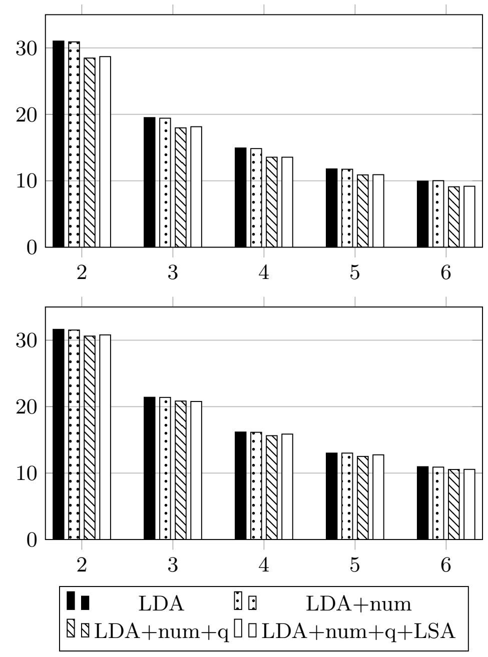


Figure 4: Compare importance of different feature groups: ARR for $k = 2, \cdots, 6$. Above: Laptop; below: TV

study five non-smooth algorithms. Besides the aforementioned 1) `powell` and 2) `nelder-mead`, we also study: 3) `cg`: conjugate gradient method in non-smooth case; 4) `bfgs`: second order optimization method in non-smooth case; and 5) `slsqp`: sequential least square programming. For all the five methods we use the implementation in Python library[9]. For each algorithm, we run 5 fold cross validation to tune the error tolerance as well as to find a good starting point. We report the performance of each algorithm in Table 4. Due to space limit and since our goal is comparative study, results in Table 4 is the average over $k = 2, \cdots, 6$. To ensure the statistical significance, we randomly restart each algorithm

[9]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`

		`powell`	`bfgs`	`nelder`	`cg`	`slsqp`
avg ARR	L	17.77	17.58	17.78	17.60	**17.50**
	T	**18.70**	18.76	18.74	19.06	18.76
time	L	0.024	**0.007**	0.028	0.012	0.027
	T	0.022	**0.008**	0.026	0.009	0.009

Table 4: Compare different non-smooth optimization methods: averaged ARR and running time over $k = 2, \cdots, 6$.

50 times and report the average (i.e., each number in Table 4 is averaged over 50× 5 values).

From Table 4 we can see that the five algorithms have slightly different performances: `slsqp` has the best performance in Laptop and `powell` has the best performance in TV. `powell` and `nelder-mead` has the largest time cost, while `bfgs` is the fasted algorithm among all. This can be explained by the fact that `bfgs` is a second order method, while `Powell` and `nelder-mead` does not leverage the gradient information compared with the other three.

6.2.3 Comparative Study on Regression Tree Features

Since regression tree method (Section 5.3) uses feature $\mathbf{x}^i$ for each query q^i, in this section, we study the influence from different features. We use three groups of features:

Semantic representation for q^i: we use both latent semantic analysis (LSA) and latent Dirichlet allocation (LDA). For each method the dimension is set to 20.

Number of explicitly mentioned facets in q^i: we use Stanford Named Entity Recognizer (NER) to label the explicitly mentioned facets in each query. For example, for query '17 in refurbished laptop', explicitly mentioned facets are screen size=17 and condition=refurbished, so this feature = 2. We manually label 40% of the queries for training, the rest are computed by the recognizer. Intuition behind this feature is when user mentions more facets, it is more likely she is looking for a higher profiled product;

Quartile absolute values of numerical facets in E^i: we use quartile facets, which are absolute values of the 25%, 50% and 75%th smallest facets in E^i. Intuition behind this feature is when retrieved products are all very expensive, user may prefer relatively less expensive products in the list;

We study four combinations of these features[10]: (1) LDA (dimension=20): using only 20 features from LDA; (2) LDA

[10]In this experiment the splitting criterion of regression tree is fixed to criterion 2 and non-smooth optimization method is fixed to Powell's method.

+ num (dimension=21): adding the number of explicitly mentioned facets; (3) LDA + num + q (dimension=24): adding the quartile absolute value features; (4) LDA + num + q + LSA (dimension=44): adding 20 features from LSA. The comparative results of the four groups is shown in Figure 4. Figure 4 shows that quartile absolute value features is most helpful in reducing ARR; number of explicitly mentioned facets does not help a lot; LSA features also do not help ARR, actually hurts ARR in many cases, which can be explained by the fact that we already have LDA features.

6.2.4 *Comparative Study on Regression Tree Splitting Criterion*

In Section 5.3, we discuss the usage of two splitting criteria for building the regression tree. Recall the first criterion is to minimize $C_n(R)$ (Equation 4), while the second criterion is to minimize MSE. Therefore, we denote the first criterion as `nonsquare` and the second criterion as `square`. In this section, we study the influence of splitting criterion on the performance of regression tree. In order to make a comprehensive comparison, we look into three trees under each criterion: first, fully grown tree without pruning, denoted as full; second, the smallest tree after pruning, which only contains the root node and two leaf nodes, denoted as min; third, the best ARR among all the pruned trees and the fully grown tree, denoted as best [11]. In Figure 5 we show p values in the T-test results between the two criteria. When criterion 2 is better, we plot the p value in positive (`square`); otherwise, we plot the p value in negative (`nonsquare`).

From Figure 5 we can see that the difference between the two criteria are basically consistent over $k = 2, \cdots, 6$. Although none of the p values is small enough to show statistical significance, we can still observe a few phenomena: first, best of `nonsquare` is slightly better than `square`; second, min of `nonsquare` is more significantly better than `square`; third, full of `square` is instead better. These observations can be naturally explained: since the splitting criterion of `nonsquare` is to optimize C_n which approximates ARR, it is expected to achieve better ARR than `square`, for the same reason its min should also have better performance. Meanwhile, due to the scarcity of data samples in leaf nodes, full of `nonsquare` should be more overfitted than `square`, because it tries to fit ARR in every possible step.

6.2.5 *Comparative Study on* $p(e)$

	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
L	63.44	59.65	55.98	54.78	51.75
T	61.78	60.42	59.39	58.29	57.16

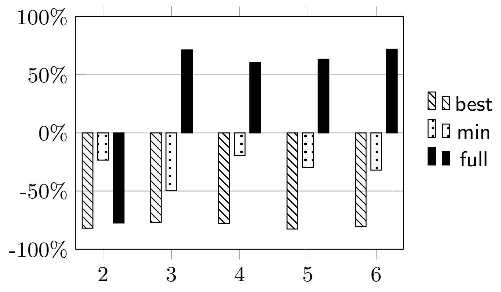
Table 5: ARR using $p(e) \propto 1/rank(e)$

In this section we study the performance of the DP algorithm using different $p(e)$'s. First, $p(e)$ used in Table 2 is a combination of the query relevance and the category relevance models:

$$\begin{aligned} p(e) &= \lambda p_q(e) + (1-\lambda) p_{cate}(e) \\ p_{cate}(e) &\propto \#click(e, cate) \\ p_q(e) &\propto \#click(e, q) \end{aligned}$$

where $\#click(e, cate)$ is the number of clicks on product e under category $cate$; $\#click(e, q)$ is the number of clicks on e under query q. These number of clicks are counted from the entire training data (Section 6.1). As a result, DP in Table 2 uses the same amount of training data as `tree` and `powell`. The best tuned parameter $\lambda = 0.5$, which we use in Table 2.

[11]In this experiment $\mathbf{x}^i$ is fixed to LDA + num + q and non-smooth optimization method is fixed to Powell's method.



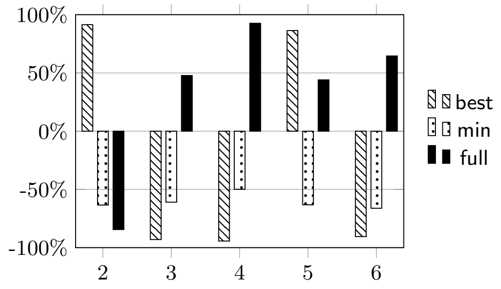


Figure 5: Compare different splitting criteria for regression tree method: p-value in T-test between minimizing mean square error (`square`) and minimizing C_n (`nonsquare`). Above: Laptop; below: TV

Alternatively, $p(e)$ can be estimated from e's rank on `www.walmart.com`, i.e., $p(e) \propto 1/rank(e)$. To compare the performance of two methods for estimating $p(e)$, we display the ARR of the second method in Table 5. From Table 5 and Table 2 we can see the first method significantly outperforms the second one, which explains that leveraging training data can help improve the performance of our first method.

7. CONCLUSION

In this paper, we introduce a new problem of numerical facet range partition. We propose evaluation metric ARR based on the browsing cost for user to navigate into relevant entities. We propose two methods that leverages training data, and compare them with the quantile method which does not use training data. Experimental results show that for the TV category, quantile method already achives near-optimal performance; while for Laptop, our second method significantly outperforms quantile method, it even significantly outperforms our first method, which leverages the same amount of training data. Our second method is robust and efficient, so it can be directly applied to any search engine that supports numerical facets.

Future directions include: First, how to generate ranges for interactive search? How to improve partition based on previous user feedback? Second, is there an easily interpretable way of partitioning categorical facets, e.g., brand? Third, how to tune parameter k and rounding precision?

Acknowledgement

This work is supported in part by NSF under Grant Numbers CNS-1513939 and CNS-1408944.

8. REFERENCES

[1] J. Acharya, I. Diakonikolas, C. Hegde, J. Z. Li, and L. Schmidt. Fast and near-optimal algorithms for approximating distributions by histograms. In T. Milo and D. Calvanese, editors, *PODS*, pages 249–263. ACM, 2015.

[2] M. B. Arouxet, N. Echebest, and E. A. Pilotta. Active-set strategy in Powell's method for optimization without derivatives. *Computational & Applied Mathematics*, 30:171 – 196, 00 2011.

[3] L. Azzopardi. Modelling interaction with economic models of search. In S. Geva, A. Trotman, P. Bruza, C. L. A. Clarke, and K. JÃd'rvelin, editors, *SIGIR*, pages 3–12. ACM, 2014.

[4] S. Basu Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 13–22, New York, NY, USA, 2008. ACM.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Pacific Grove, 1984.

[6] R. Brent. *Algorithms for minimization without derivatives*. Prentice-Hall, 1973.

[7] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 87–94, New York, NY, USA, 2008. ACM.

[8] A. Dvoretzky, J. Kiefer, and J. Wolfowitz. Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator. The Annals of Mathematical Statistics, pages 1397–1400. ACM, 1956.

[9] F. Gao and L. Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Comp. Opt. and Appl.*, 51(1):259–277, 2012.

[10] M. A. Hearst. Uis for faceted navigation: Recent advances and remaining open problems. 2008.

[11] M. A. Hearst. *Search User Interfaces*. Cambridge University Press, 1 edition, 2009.

[12] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB*, pages 275–286. Morgan Kaufmann, 1998.

[13] K. JÃd'rvelin. Cumulated gain-based evaluation of ir techniques. volume 20, page 2002, 2002.

[14] C. Kang, D. Yin, R. Zhang, N. Torzec, J. He, and Y. Chang. Learning to rank related entities in web search. *Neurocomputing*, 166:309–318, 2015.

[15] A. Kashyap, V. Hristidis, and M. Petropoulos. Facetor: cost-driven exploration of faceted query results. In J. Huang, N. Koudas, G. J. F. Jones, X. Wu, K. Collins-Thompson, and A. An, editors, *CIKM*, pages 719–728. ACM, 2010.

[16] J. Koren, Y. Zhang, and X. Liu. Personalized interactive faceted search. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 477–486, New York, NY, USA, 2008. ACM.

[17] B. Kules, R. Capra, M. Banta, and T. Sierra. What do exploratory searchers look at in a faceted search interface? In *JCDL '09: Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, pages 313–322, New York, NY, USA, 2009. ACM.

[18] S. Liberman and R. Lempel. Approximately optimal facet selection. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 702–708, New York, NY, USA, 2012. ACM.

[19] A. Moffat and J. Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Inf. Syst.*, 27(1), 2008.

[20] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In H. Boral and P.-Ã. Larson, editors, *SIGMOD Conference*, pages 28–36. ACM Press, 1988.

[21] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[22] P. Pirolli and S. Card. Information foraging. *Psychological Review*, 106. 4:634–675, 1999.

[23] S. E. Robertson. The probability ranking principle in ir. *Journal of Documentation 33*, pages 294–304, 1997.

[24] S. B. Roy, H. Wang, G. Das, U. Nambiar, and M. K. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *CIKM*, pages 13–22. ACM, 2008.

[25] K. Sparck Jones and P. Willett, editors. *Readings in Information Retrieval*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[26] R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[27] H. Valizadegan, R. Jin, R. Zhang, and J. Mao. Learning to rank by optimizing ndcg measure. In *NIPS*, pages 1883–1891, 2009.

[28] R. van Zwol, B. SigurbjÃűrnsson, R. Adapala, L. G. Pueyo, A. Katiyar, K. Kurapati, M. Muralidharan, S. Muthu, V. Murdock, P. Ng, A. Ramani, A. Sahai, S. T. Sathish, H. Vasudev, and U. Vuyyuru. Faceted exploration of image search results. In *WWW*, pages 961–970. ACM, 2010.

[29] D. Vandic, F. Frasincar, and U. Kaymak. Facet selection algorithms for web product search. In *Proceedings of the 22Nd ACM International Conference on Conference on Information &#38; Knowledge Management*, CIKM '13, pages 2327–2332, New York, NY, USA, 2013. ACM.

[30] E. Yilmaz, M. Verma, N. Craswell, F. Radlinski, and P. Bailey. Relevance and effort: An analysis of document utility. In *CIKM*, pages 91–100. ACM, 2014.

[31] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR*, pages 271–278. ACM, 2007.

[32] Y. Zhang and C. Zhai. Information retrieval as card playing: A formal model for optimizing interactive retrieval interface. In *SIGIR*, pages 685–694. ACM, 2015.