Poster Paper

Quantifying Likelihood of Change through Update Propagation across Top-k Rankings*

Evica Milchevski
TU Kaiserslautern
Kaiserslautern, Germany
milchevski@cs.uni-kl.de

Sebastian Michel
TU Kaiserslautern
Kaiserslautern, Germany
smichel@cs.uni-kl.de

ABSTRACT

Rankings are a widely used techniques to condense a potentially large amount of information into a concise form. However, rankings are dynamic and undergo changes, thus need to be maintained, which can be a tedious and expensive task. Given a ranking τ that got updated to τ', we aim at identifying those rankings σ that are very likely to have changed as well, as they are close in distance to the original ranking τ. We do so by modeling the expected change in form of a hypothetical ranking σ' and mark σ to require a refresh if the expected change is above a threshold. We do this for the Footrule distance and demonstrate through a preliminary evaluation the potential of our approach.

1. INTRODUCTION

We focus on the task of maintaining a set of crowdsourced entity rankings. One important characteristic of crowdsourced rankings is that although they share the same entities, they conform to different constraints, thus, a change in one ranking does not imply the same change in another ranking. We propose a framework that uses the similarity between the rankings, to reason about the degree of change in a set of rankings due to an update in one ranking. Since the distance between two rankings resembles not only structural but semantic similarity as well, it is reasonable to assume that once a ranking changes, it is more likely that similar rankings change, rather than dissimilar ones. Specifically for top-k rankings that only report on a (usually short) subset of items, if two rankings are similar, they need to share also a fraction of items. If ranking τ changes, this means that items that are present in τ changed, respectively their features. Such changes might or might not propagate to a ranking σ that is in distance λ to τ—the likelihood of such a propagation is what we aim at quantifying in this work.

When considering rankings created over some objective (measurable) scoring function, like wealth in USD, the update of the rankings can be done by maintaining one global ranking, and directly updating all rankings affected by an update. However, keeping a global ranking in the case of crowdsourced rankings, where there is no measurable scoring function, but instead entities are ranked by some user perceived quality, like popularity, is not only expensive, but also unintuitive, as there is no ground truth.

1.1 Problem Statement

As **input** we are given a set $\mathcal{T}$ of top-k rankings τ. Each ranking τ has a domain $\mathcal{D}_\tau$—the items it ranks. We further know the Footrule distance between each pair of rankings in $\mathcal{T}$. We define an **update $u_\tau(i,j)$ to a ranking τ as a swap of two items** $i, j \in D_\tau$. A size of an update $u_\tau(i,j)$, for brevity denoted simply as $|u_\tau|$, is the difference between the positions of the swapped items, $|\tau(i) - \tau(j)|$. We denote with τ' the ranking τ after applying an update $u_\tau(i,j)$.

For a given update u over a ranking τ, the task is to compute the likelihood that u affects other rankings $\sigma \in \mathcal{T}, \sigma \neq \tau$ such that $F(\sigma, \sigma')$ is larger than some user defined threshold θ. In that case, σ is marked to be refreshed (e.g., crowdsourced) to bring it up to date. If we believe a ranking is not affected by a change but in fact it is, we suffer loss in **recall**, the ranking is not refreshed and our database $\mathcal{T}$ is getting stale. If we, however, believe it is affected and it is not, we suffer loss in **precision**, which leads to wasting cost to refresh a ranking when it is not required to do so. For an overview to methods for comparing top-k rankings, see [1].

2. APPROACH

Algorithm 1 shows the procedure for determining (allegedly) affected rankings. As input the algorithm takes a specific update, a set of rankings, where the Footrule distance between all pairs is known, and a threshold θ. First, we compute the maximum distance, d_{max} that would likely result in the updated items i, j being present in both rankings τ and σ, i.e., $i, j \in \mathcal{D}_\tau \cap \mathcal{D}_\sigma \Rightarrow F(\tau, \sigma) \leq d_{max}$ with some probability p. We explain how this bound can be computed below. The next step is computing the expected change according to the distance. For this purpose, in a pre-processing step for each possible distance, for a given k, we compute an average difference between the positions of the items in two rankings τ and σ, such that $F(\tau, \sigma) = \lambda$. We call this average displacement (see Section 2.1) and it does not depend on the actual rankings in $\mathcal{T}$. Using the average displacements, we compute the expected change, according to the actual update. If the change is larger than a user specified threshold θ, we retrieve all distances and output all rankings within the retrieved distance to the changed ranking.

When we have a sequence of updates, we can simply accumulate the change. Note that an update over several items can also be considered as a sequence of updates of two items.

2.1 Computing the Expected Change

Since we do not know the positions, if any, of the affected items in the rankings, the first step toward quantifying the expected change in a ranking is reasoning about the most

*This work has been supported by the German Research Foundation (DFG) under grant MI 1794/1-1.

input: $u_\tau, \mathcal{T}, \theta$
1 d_{max} = get_distance_limit()
2 **for each** $E[\mu]$ **in** get_expectations(k) **do**
3 e_{change}= get_expected_change($u_\tau, \tau, E[\mu]$)
4 **if** $e_{change} \geq \theta$ **then**
5 **for each** $d \leq d_{max}$ **in** get_distances(e_{change}) **do**
7 $R \leftarrow$ get_all_rankings(d)
8 **return** R

Algorithm 1: Algorithm for determining all the rankings in $\mathcal{T}$ affected by a change u_τ according to their distance.

likely position of the affected items, when the distance λ between the rankings is known. To do this we first define a displacement of an item i in two rankings τ and σ:

DEFINITION 1. **Displacement of an item:** *For two rankings τ and σ, $\tau \neq \sigma$, we define a displacement of an item i, denoted with μ_i, $i \in D_\tau \cap D_\sigma$, as the difference of the position of the item in the two rankings, i.e., $\mu_i = |\sigma(i) - \tau(i)|$. In the case when $i \in \mathcal{D}_\tau \setminus \mathcal{D}_\sigma$, $\mu_i = k+1-\tau(i)$ or vice versa.*

The Footrule distance between two rankings τ and σ is in fact the sum over the displacements of the items in $\mathcal{D}_\tau \cup \mathcal{D}_\sigma$. We can compute the most likely position of the affected items, i, j in σ as $E[\sigma(i)] = \tau(i) + E[\mu]$ and $E[\sigma(j)] = \tau(j) + E[\mu]$, where $E[\mu]$ is the average displacement between the items in two rankings within distance λ of each other.

To compute the average displacement $E[\mu]$ for a given Footrule distance λ we need to compute all the displacements that contribute to that distance. The naive way to do this is to generate all top-k rankings for a given k, compute the distances between all combinations of rankings, and compute the *average item displacements* for every distance. However, this is computationally very expensive, as generating all the rankings of size k has a complexity in $O(k!)$.

One key observation that allows us to more efficiently compute the sample space is the fact that the Footrule distance is a sum over non-negative integers, where each integer is a displacement of an item. In number theory and combinatorics, an unordered collection of positive integers whose sum is n is called a partition of n. Several efficient algorithms for generating all the partitions for a number, working in constant amortized time, have been proposed. Thus, we could use one of those algorithms to generate all the partitions for the distance λ. The resulting set of partitions could be used to compute the average displacement $E[\mu]$. Note that not all partitions of λ should be used in computing $E[\mu]$. The details of how exactly $E[\mu]$ can be computed we leave out of this paper due to lack of space. Considering an update $u_\tau(i, j)$ of τ, one can compute the expected change $E[F(\sigma, \sigma')]$ as:

$$E[F(\sigma, \sigma')] = (1 - \lambda) \times (2 \times |E[\sigma(i)] - E[\sigma(j)]|)$$

where λ is the distance between τ and σ. The reasoning is that since these are the only two items that changed in σ' with respect to σ, the change can be computed as $(2 \times |E[\sigma(i)] - E[\sigma(j)]|)$. However, since σ is only similar to τ, we do not want to fully propagate the change. Thus, we multiply the change by $1 - \lambda$ (we normalize λ, thus $0 \leq \lambda \leq 1$).

The above formula only covers the case when we assume that the affected items belong to the domains of both rankings. However, since we are working with top-k rankings it can happen that the updated items in τ cannot be found in σ at all. To eliminate the rankings that are so dissimilar to the updated ranking, and thus it is very unlikely that they changed, we define a maximum distance bound d_{max}. This bound is computed using the probability of not finding both updated items i and j in σ, $P(i, j \notin D_\tau \cap D_\sigma) = \frac{\binom{2*(k-w)}{2}}{\binom{2*k-w}{2}}$, where w is the overlap between the rankings. Since we only know the distance between the rankings, we can compute the maximum overlap that two rankings can have within a distance λ as $w_{max} = \lceil \frac{1}{2} \times (-1 + \sqrt{1 - 4 \times \lambda + 4 \times k + 4 \times k^2}) \rceil + 1$ and then plug this value into the above probability estimation formula.

	$\theta = 0.1$		$\theta = 0.15$	
	Precision	Recall	Precision	Recall
Our approach	0.58	0.6	0.59	0.41
Baseline	0.12	1	0.08	1

Table 1: Experimental results: Precision and Recall

3. EXPERIMENTS

We have implemented the described algorithm in Java 8. We created one synthetic dataset by first creating a small set of base rankings, by randomly choosing at least ρ from k items, where k is the size of the rankings, and then randomly choosing the remaining $k - \rho$ items. All the other rankings are created by swapping a random number of items from the base rankings. The dataset contains 500 rankings with size $k = 10$. We compared our approach with the baseline approach—retrieving all rankings that have at least one item in common with the affected ranking. For the experiments, we randomly selected one ranking from the dataset, randomly selected a pair of items from this ranking, and then swapped their places. We then used our method and the baseline to find the affected rankings in the dataset.

Table 1 reports the average precision and recall for the two approaches over 100 trials. We report on results for two values of the threshold θ, 0.1 and 0.15. To compute d_{max}, we set $P(i, j \notin D_\tau \cap D_\sigma)$ to 0.9. Note that the recall of the baseline is always 1 since the relevant rankings must have at least one item in common with the changed ranking. We can see that with our approach we can achieve high precisions (much higher than the baseline) while still maintaining relatively high recall.

4. RELATED WORK

Research around crowdsourcing information usually addresses the problem of reducing the cost, while still retaining high quality results. Guo et al. [3] address the problem of finding the highest ranked object using the least number of questions, from a set of objects, in a crowdsourcing database system. Wang et al. [5] use transitive relations to reduce the number of questions asked to the crowd for the case of crowdsourced joins. Gruenheid and Kossmann [2] investigate the cost and quality trade-offs of different algorithms in a crowdsourcing environments. Polychronopoulos et al. [4] propose an algorithm for creating top-k lists using the crowd, The idea behind the algorithm is to create a high agreement top-k list for a low latency and monetary cost, by adaptively choosing the number of tasks posed to the crowd. To the best of our knowledge, there has not been any work that focuses on maintaining a set of crowdsourced rankings.

5. REFERENCES

[1] R. Fagin et al. Comparing Top k Lists. SIAM J. Discrete Math., 2003
[2] A. Gruenheid and D. Kossmann. Cost and quality trade-offs in crowdsourcing. *DBCrowd*, 2013.
[3] S. Guo et al. So who won?: dynamic max discovery with the crowd. *SIGMOD*, 2012.
[4] V. Polychronopoulos et al. Human-powered top-k lists. *WebDB*, 2013.
[5] J. Wang et al. Leveraging transitive relations for crowdsourced joins. *SIGMOD*, 2013.