How to Find the Best Rated Items on a Likert Scale and How Many Ratings Are Enough

Qing Liu[1], Debabrota Basu[2], Shruti Goel[3],
Talel Abdessalem[4], Stéphane Bressan[5]

[1,2,5]School of Computing, National University of Singapore, Singapore, Singapore
[3]Jacobs School of Engineering, University of California San Diego, La Jolla, CA, USA,
[4]LTCI/IPAL CNRS, Télécom ParisTech, Université Paris-Saclay, France

Abstract One of the modern pillars of collaborative filtering and recommender systems is collection and exploitation of ratings from users. Likert scale is a psychometric quantifier of ratings popular among the electronic commerce sites. In this paper, we consider the tasks of collecting Likert scale ratings of items and of finding the n-k best-rated items, i.e., the n items that are most likely to be the top-k in a ranking constructed from these ratings.
We devise an algorithm, Pundit, that computes the n-k best-rated items. Pundit uses the probability-generating function constructed from the Likert scale responses to avoid the combinatorial exploration of the possible outcomes and to compute the result efficiently. We empirically and comparatively evaluate with real data sets and discuss the effectiveness and efficiency of our and competing approaches. Our method is effective and competitively efficient.
Selection of the best-rated items meets, in practice, the major obstacle of the scarcity of ratings. We propose an approach that learns from the available data how many ratings are enough to meet a prescribed error and recommends how many additional ratings should be proactively sought. We also empirically evaluate with real data sets the effectiveness of our method to recommend the collection of additional ratings. The results show that the approach is practical and effective.

1 Introduction

One of the modern pillars of collaborative filtering and recommender systems [4] is accumulation and exploitation of reviews and ratings from users. This process involves three basic steps- quantification of the reviews using a well-defined rating scale, accumulation of ratings to form a knowledge-base about the products and exploiting that knowledge-base for recommender and collaborative systems.

Likert scale is a rating scale popular among the electronic commerce sites and crowdsourced information systems, such as TripAdvisor, Amazon etc. It is used to quantify the reviews of users regarding the quality of the products and services. Originally, Likert scale is a 5-valued psychometric scale developed by Rensis Likert [13]. Each value in the scale gauges the degree of agreement or disagreement of the user towards a particular question. Instead of restricting

to the classical 5-valued Likert scale, we develop our method for a general L-valued ordinal scale and experimentally instantiate it on the classical Likert scale. Hereinafter, we call it an *L-valued Likert scale*.

Since we settle down with the L-valued Likert scale as our quantifier of user reviews, the remaining issue are twofold. The first question is how to accumulate the ratings to create a knowledge-base about the product. But aggregating individual Likert scale ratings is not always as obvious as it seems to be. For instance, ranking items using the expectations of the Likert scale responses can yield incorrect results [8]. The second question is how to exploit the knowledge-base for recommendation. We consider the task of finding the list of n items that are most likely to be the top-k in a ranking constructed from the accumulated ratings as our target recommendation task. For the sake of simplicity, we refer to this list as the *n-k best-rated items*. In this paper, we represent the aggregated ratings as discrete distributions on an L-valued Likert scale, which solves the first question. Then, let us focus on the second question.

We define the problem of finding the n-k best-rated items as a probabilistic one (Section 3.1). The uncertainty arises not from the unreliability of users but from the unavailability of ratings by all the users. We assume that the ratings from all users are correct and exact. Correctness implies that every user chooses the value of response in the Likert scale that is true to their knowledge of the product. Exactness implies that if an user rates an item r, then r unambiguously represents her response about its quality. These assumptions imply individual ratings are reliable. The probabilistic formulation of the task is not due to the aleatoric uncertainty [3]. Rather the uncertainty of the task appears due to unavailability of the ratings by the universal user-pool. This keeps the knowledge-base of the system incomplete. The knowledge-base would be complete if the underlying discrete distribution built from all ratings from all users is available. This probabilistic formulation of the task is due to the epistemic uncertainty [3] caused by insufficient ratings.

We develop a polynomial-time algorithm, Pundit, that computes the n-k best-rated items (Section 3.4). The uncertainty for each individual item is summarized into the corresponding discrete distribution defined over the L-valued Likert scale. Pundit exploits the probability-generating functions [9] of the discrete distributions of the items to avoid the combinatorial exploration of all possible outcomes and to compute the result efficiently. Algorithms devised in [12] also use probability-generating functions to answer their top-k query which is modelled on the uncertainty over existence of items. [12] does not model the epistemic uncertainty originated from scarcity of reviews. In the present formulation, the epistemic uncertainty is captured through a discrete distribution defined over the L-valued Likert scale that evolves with accumulation of ratings. Dynamic nature of this distribution provides us a single mathematical object to encompass the uncertainty rather than scanning over all possible knowledge-bases. Above all, our method is exact whereas the other methods computing the n-k best-rated items are approximate algorithms like ranking by mean ratings or Monte Carlo sampling [18] (Section 3.2 and 3.3). We empirically and comparatively evaluate

the effectiveness and efficiency of these methods using Amazon review dataset (Section 3.5). Experiments validate that our method is competitively efficient.

In practice, the problem is not solved yet. Since selection of the n-k best-rated items is constrained by insufficient ratings, corresponding discrete distributions are not "true" representations of the quality of the products rather they are representatives of the observed or "incomplete" knowledge-base. This uncertainty engendered from incompleteness of knowledge-base introduces error in the final ranking. Thus, we revisit our question about how to accumulate the ratings and twist it slightly to answer how many ratings are enough to form a reliable approximation of "true" knowledge-base. We use this opportunity to devise the score distribution error model based on KL-divergence [10] (Section 4.1).This error model estimate the deviation of the discrete distribution formed with the available data from the "true" universal distribution portraying the complete knowledge-base. As we empirically evaluate this model on the Amazon review dataset, we observe the KL-divergence based model follows inverse law (Section 4.2). Following this we use the inverse law for KL-divergence based error to recommend how many additional ratings should be proactively sought to reach a certain error threshold. The results validate that the approach is practical and effective (Section 4.3).

Before delving into the details, we discuss the related work in Section 2.

2 Related Work

We study the problem of finding the n-k best rated items. This problem is related to the probabilistic threshold top-k query [7] that returns the items having a probability of being in the top-k over a user specified threshold. [12] proposes a unified way to summarize a category of probabilistic top-k queries. Though the problem definitions seem similar, this category of probabilistic top-k queries is applicable to the scenario where the existence of an item is uncertain. Each item has a fixed known score representing its quality. However, the uncertainty modelled in our problem emerges from the unavailability of the ratings by the universal user-pool and is expressed as an evolving distribution over Likert scale.

Eclectic definitions of the uncertain top-k queries are available in the literature. For example, [11] studies the problem of ranking continuous probabilistic data, where the score of each item is modelled as a continuous probability distribution. The authors focus on finding the probability of an item being ranked at a certain position. This result cannot be applied directly to our problem since the probability of an item being in top-k is not simply the sum over the probabilities of it being at different positions in the ranking. Another variant of these queries is UTop-Rank query [19]. This query searches for the item that has the highest probability of being ranked within a certain range of positions. They solve UTop-Rank query based on Monte Carlo sampling techniques that produces an approximate result. We construct a polynomial time exact algorithm for our problem that is shown to be more efficient than the Monte Carlo method.

With emergence of the electronic commerce platforms, crowdsourcing based approaches are proposed for the ranking and top-k problems. The authors of [6] study the problem of finding the 'max' item with the help of the crowd. They asked the crowd to compare pairs of items. They propose a group of

heuristic algorithms to aggregate the ratings collected from the crowd and to find the item with the maximal score. [20] determines the maximum item by asking the crowd to select the item which they believe is the maximum from a given set. Then, they devise several heuristic max algorithms for aggregating the responses. Similarly, [1] studies the problem of finding a gold-standard ranking by aggregating the results of pairwise comparisons with crowdsourcing. This paper develops an algorithm called Crowd-BT to learn the global ranking with the crowdsourced responses. [6,1] also consider the quality of the crowd while developing their methods. Beside this, [22] provides a thorough experimental study of the crowdsourced top-k queries.

Most of the works in crowdsourcing use the *preference judgement* scheme. This scheme is based on the pairwise comparisons accumulated from the crowd for inferring the global ranking. Hybrid approaches, such as [21,17], combine preference judgement and *absolute judgement*, like ratings, to infer the ranking. These approaches either transform the absolute judgement into the preference judgement [17] or use the parametric analysis [21] which may not be suitable for the ordinal data since the intervals between the ordinal values cannot be presumed equal. In this paper, we adopt the absolute judgement in form of the correct and exact ratings to infer the global ranking of the items. The score of the item is modelled as a discrete distribution over an L-valued Likert scale.

3 How to Find the n-k Best-Rated Items?

3.1 Problem Definition

We use similar notations as in [14]. Consider a set of N items, $\mathcal{O} = \{o_1, \cdots, o_N\}$. A *scoring function* s maps the set of items $\mathcal{O}$ to a totally ordered domain $\mathcal{D}$, i.e. $s : \mathcal{O} \to \mathcal{D}$. $(\mathcal{D}, \geq)$ denotes a total order and $(\mathcal{D}, >)$ is the corresponding strict total order of $\mathcal{O}$ induced by s. We call the image $s(o)$ of an item $o \in \mathcal{O}$ by the function s the score of the item. A *ranking* $r : \mathcal{O} \to S_N$ is an indexing function induced on $\mathcal{O}$ by the total order $(\mathcal{D}, \geq)$. Here, S_N denotes the permutation group on $\{1, \ldots, N\}$. It is the set of all possible rankings of N items. For any two items o_i and $o_j \in \mathcal{O}$, if score of o_i is greater than or equal to that of o_j, i.e., $s(o_i) \geq s(o_j)$, we say that o_i is ranked equally with or above o_j, i.e., $r(o_i) \leq r(o_j)$. This deterministic definition of score function is valid if complete knowledge of the items is obtained.

But we deal with the incomplete knowledge-base scenario where the score of each item is constructed from a collection of ratings. The epistemic uncertainty introduced by insufficiency of ratings prohibits existence of a deterministic score. Instead, we model the score $s(o_i)$ of an item $o_i \in \mathcal{O}$ as a random variable X_i with a probability mass function f_i. Thus, we define the score function as $s : \mathcal{O} \to \{f : \mathcal{L} \to [0, 1]\}$. Here, $\mathcal{L} \triangleq \{1, \ldots, L\}$ is the L-valued Likert scale and f is a probability mass function defined over the support $\mathcal{L}$. We call f a *score distribution*. A score distribution represents and quantifies the underlying uncertainty at a certain instance. Its evolution through gradual accumulation of ratings echoes the growth of the knowledge-base.

Example 1. Consider the i^{th} hotel on TripAdvisor, Lakeview. 495 reviews of this hotel are collected using a 5-valued Likert scale. 125 of the users rate Lakeview

as 'Excellent(5)', 207 rate it as 'Very good(4)', 106 rate it as 'Average(3)', 36 rate it as 'Poor(2)' and 21 rate it as 'Terrible(1)'. After normalizing the count on different ratings, we get the score $s(o_i)$ for this hotel as a score distribution over $\{1, \ldots, 5\}$ defined by $s(o_i) = \{f_i[j]\}_{j=1}^{5} = \{0.04, 0.07, 0.22, 0.42, 0.25\}$.

If $x_1, \cdots, x_N \in \mathcal{L}$ are the observed ratings of the items, the probability of an item o_i to be ranked in top-k is expressed in Equation 1.

$$\mathbb{P}(r(o_i) \leq k) = \sum_{\{x_1, \cdots, x_N\} \in S_i^k} f_1(x_1) \cdots f_N(x_N). \tag{1}$$

Here, S_i^k is the set of all N-tuples $\{x_1, \cdots, x_N\}$, such that for each $\{x_1, \cdots, x_N\}$ there exist at least $(N - k + 1)$ number of x's which are less than or equal to x_i. We call $\mathbb{P}(r(o_i) \leq k)$ the *positional probability* of o_i.

Example 2. Suppose there are three items, o_1, o_2 and o_3. If $k = 1, i = 2$, $\{x_1 = 5, x_2 = 1, x_3 = 5\}$ is not in S_2^1. Because it consists no rating lower than or equal to x_2. But $\{x_1 = 1, x_2 = 1, x_3 = 1\}$ is in S_2^1. Because ratings of o_1 and o_3 are equal to the rating of o_2.

Leveraging the structure of Equation 1, score distributions and ranking functions, we define the query for finding the n-k best-rated items. We call it the *n-top-k query*. The n-top-k query searches for the sequence of n items, $\Omega = [o_1, \ldots, o_n]$, such that $\mathbb{P}(r(o_1) \leq k) \geq \cdots \geq \mathbb{P}(r(o_n) \leq k)$, and $\mathbb{P}(r(o_n) \leq k) \geq \mathbb{P}(r(o_{i'}) \leq k)$ for all $o_{i'} \notin \Omega$. This means that the items in Ω are ranked according to their probability to belong in the top-k and probability of other $N - n$ items to be in top-k is less than that of any item in Ω.

3.2 Mean Score Ranking Algorithm

A natural intuition to answer the n-top-k query is first to rank the objects according to the mean of their score distributions and then to choose the top-n of them. We call it the `Mean Score Ranking` algorithm.

Example 3. Assume that there are two items o_1 and o_2 and their score distributions are $\{0.3, 0.05, 0.65, 0, 0\}$ and $\{0.05, 0.5, 0.45, 0, 0\}$. Suppose we are interested in the 1-top-1 query. Since mean scores of o_1 and o_2 are 2.35 and 2.4 respectively, `Mean Score Ranking` returns o_2. Through direct calculations, we get $\mathbb{P}(r(o_1) \leq r(o_2)) = \mathbb{P}(s(o_1) \geq s(o_2)) = \sum_{l=1}^{L} \mathbb{P}(s(o_1) \geq l)\mathbb{P}(s(o_2) = l) = 1 \times .05 + 0.7 \times 0.5 + 0.65 \times 0.45 = 0.6925$. Similarly, we get $\mathbb{P}(r(o_2) \leq r(o_1)) = 0.64$. Thus, the 1-top-1 query should return o_1. Thus, `Mean Score Ranking` is not always correct.

3.3 Exhaustive Search and A Monte Carlo Approximation

A naïve approach of solving the n-top-k query is to search through all possible ratings of the N items, and to determine for each combination if it is in S_i^k. If it belongs to S_i^k, use its probability to calculate the positional probability of Equation 1. The number of all possible rating combinations is L^N. Thus, the naïve approach is exponentially explosive and impractical. Hence, [19] proposed a Monte Carlo search to approximate the positional probability from the observed ratings. The intuition is to rank each item based on the frequency with which it appears in top-k. But `Monte Carlo` may get stuck at the local minima or may ask for consideration of several initial configurations. Using `Monte Carlo` is expensive. It returns us an approximate solution which can be suboptimal.

3.4 Pundit: An Exact Algorithm for n-top-k Query

Motivated by the shortcomings of the basic algorithms, we develop an exact algorithm, `Pundit`, that answers the n-top-k query in polynomial time. The idea is to construct a degree N polynomial such that its coefficients are dependent on the positional probability. Following this, we apply the fast Fourier transform (FFT) to calculate the positional probability for each item o_i. Once the positional probabilities are fixed, we get the n items with the highest positional probabilities as the answer to the n-top-k query.

Construction of the Polynomial. We observe that by construction "rank of o_i is higher than or equal to k, i.e., $r(o_i) \leq k$" is equivalent to the fact that "at least $N-k$ items other than o_i have scores lower than or equal to score of item o_i, i.e., $s(o_i)$".

This fact includes k mutually exclusive cases. Case $j \in \{1, \cdots, k\}$ occurs if exactly $N-k+j-1$ items other than o_i have scores lower than or equal to $s(o_i)$ and other $k-j$ scores are higher than $s(o_i)$. Thus, if we can calculate the probability for each of the k cases, the positional probability is the sum of the probabilities of these k cases.

We threshold the score functions to capture the notion of positional probability that emerges due to thresholding of the ranking function. We look into the probability distribution $\mathbb{P}(s(o_i) = l)$ for a given l. We would show that the notion of l would introduce a threshold k on r. As we have planned to shift our calculations to polynomials, we express the distribution $\mathbb{P}(s(o_i) = l)$ in terms of probability-generating function $\mathcal{F}_i(x, l)$. We construct it as shown in Equation 2.

$$\mathcal{F}_i(x, l) \triangleq \prod_{j \neq i} (\mathbb{P}(s(o_j) \leq l) + \mathbb{P}(s(o_j) > l)x) \tag{2}$$

In Equation 2, $\mathbb{P}(s(o_j) \leq l)$ denotes the probability that score of o_j is lower than or equal to l. For a given l, $\mathcal{F}_i(x, l)$ is a polynomial of x. In particular, the coefficient of the term x^k equals to $\mathbb{P}(\sum_{j \neq i} \mathbb{I}(s(o_j) > l) = k)$ [12]. Here, $\mathbb{I}(s(o_j) > l)$ is the indicator function that returns 1 or 0 depending on whether $s(o_j) > l$ is true or not.

If $s(o_i) = l$, the coefficient of the term x^{k-j} is the probability that there are exactly $k-j$ items having scores higher than $s(o_i)$ for $j \in \{1, \cdots, k\}$. This, in turn, means there are exactly $N-k+j-1$ items having scores lower than or equal to $s(o_i)$. Thus, the coefficient of x^{k-j} exactly quantifies the j^{th} case. We observe similar relations between all the k cases and the coefficients of the probability-generating function.

Probability-generating function [9] is a common technique employed to compute the probability of a discrete random variable. It is also applied by [12] for their top-k queries. In our problem, the quality of the item and the corresponding uncertainty is quantified with a discrete score distribution instead of a fixed score like [12]. Due to the inherently different uncertainties modelled in our and [12]'s problem formulations, we cannot apply their fixed score setting and also their scheme. Thus, we propose an alternative scheme to construct the generating functions and to solve the n-top-k problem.

Algorithm 1: DAC-FFT

Input: Probabilities $p_1^l, \cdots, p_N^l$ of $\mathcal{O}\backslash\{o_i\}$
Output: The coefficient vector C^l of $\mathcal{F}_i(x, l, \mathcal{O})$
if $N == 1$ **then**
  $C^l(0) \leftarrow p_1^l$; $C^l(1) \leftarrow 1 - p_1^l$;
  Return C^l;
Divide the set $\mathcal{O}$ into two sub sets $\mathcal{O}_1$ and $\mathcal{O}_2$ evenly;
$C_1^l \leftarrow$ DAC-FFT$(p_1^l, \cdots, p_{\lfloor N/2 \rfloor}^l)$; $C_2^l \leftarrow$ DAC-FFT$(p_{\lfloor N/2 \rfloor+1}^l, \cdots, p_N^l)$;
$C^l \leftarrow$ iFFT (FFT$(C_1^l) \times$ FFT(C_2^l));
Return C^l;

Coefficients Calculation. As we have constructed the probability-generating polynomial with coefficients related to the positional probabilities, the next step is to calculate these coefficients efficiently. Since the q^{th} coefficient is given by $\mathbb{P}(\sum_{j \neq i} \mathbb{I}(s(o_j) > l) = q)$, we reconstruct the generating function of Equation 2 into the polynomial expression of x.

$$\mathcal{F}_i(x, l) = c_0(l)x^0 + \cdots + c_{N-1}(l)x^{N-1} \tag{3}$$

where $c_q(l)$ represents the q^{th} coefficient. Since $\mathbb{P}(s(o_i) > l)$ and $\mathbb{P}(s(o_j) \leq l)$ are independent events for $i \neq j$, the q^{th} coefficient is given by $c_q(l) = \sum_{A \subset \mathcal{O} \wedge |A|=q} \left(\prod_{o_j \in A} \mathbb{P}(s(o_j) > l)\right) \left(\prod_{o_j \in \mathcal{O}\backslash A} \mathbb{P}(s(o_j) \leq l)\right)$. $\mathbb{P}(s(o_j) \leq l)$ and $\mathbb{P}(s(o_j) > l)$ can be computed directly from the score distributions of items. Thus, the coefficients $c_0(l), \cdots, c_{N-1}(l)$ can be computed in $O(N^2)$ time by expanding Equation 2 into Equation 3.

We propose a fast Fourier transform (FFT) [2] based scheme for faster computation of these coefficients. FFT is a commonly adopted technique for facilitating the multiplication of polynomials. Here, we propose an efficient divide-and-conquer algorithm which applies FFT to compute the coefficients $c_0(l), \cdots, c_{N-1}(l)$. We summarize this divide-and-conquer algorithm in Algorithm 1. p_j^l is used as the shorthand notation for the positional probability $\mathbb{P}(s(o_j) \leq l)$.

At the first step, we evenly divide the set of items $\mathcal{O}$ into two subsets $\mathcal{O}_1$ and $\mathcal{O}_2$. Let $\mathcal{F}_i(x, l, \mathcal{O})$ denote the probability-generating polynomial defined on set $\mathcal{O}$. Then, $\mathcal{F}_i(x, l, \mathcal{O}) = \prod_{j \neq i \wedge o_j \in \mathcal{O}} (\mathbb{P}(s(o_j) \leq l) + \mathbb{P}(s(o_j) > l)x)$. where $\mathcal{F}_i(x, l, \mathcal{O}_1)$ and $\mathcal{F}_i(x, l, \mathcal{O}_2)$ denote the polynomials defined on $\mathcal{O}_1$ and $\mathcal{O}_2$. Let C^l denote the vector of coefficients in $\mathcal{F}_i(x, l, \mathcal{O})$. C_1^l and C_2^l denote the vector of coefficients in $\mathcal{F}_i(x, l, \mathcal{O}_1)$ and $\mathcal{F}_i(x, l, \mathcal{O}_2)$, correspondingly. Then, the q^{th} coefficient in C^l is $C^l(q) = \sum_{i=0}^{q} C_1^l(i) \times C_2^l(q - i) \quad, \forall q \in \{0, \cdots, N-1\}$. It implies that C^l is convolution of the sub-vectors C_1^l and C_2^l. Thus, we can evaluate $C^l(q)$ in $O(NlogN)$ time by employing FFT and inverse FFT [2]. Each recursive call of Algorithm 1 includes two sub-problems and a convolution calculation which takes $O(NlogN)$ by applying FFT. Thus, the runtime of DAC-FFT is $T(N) = 2T(\frac{N}{2}) + O(NlogN) = O(Nlog^2N)$. Once $c_0(l), \cdots, c_{N-1}(l)$ are computed, the positional probability for an L-valued Likert scale is calculated using $\mathbb{P}(r(o_i) \leq k) = \sum_{l=1}^{L} (c_0(l) + \cdots + c_{k-1}(l))\mathbb{P}(s(o_i) = l)$.

Pundit: The Algorithm. Since calculating the positional probability $\mathbb{P}(r(o_i) \leq k)$ for each item takes $O(Nlog^2N)$ time, it would take $O(N^2log^2N)$ time for

Algorithm 2: Pundit

Input: Score distributions $f_1, \cdots, f_N$
Output: Answer to the n-top-k query

1 **for** $l = 1 : L$ **do**
2 Calculate $p_1^l, \cdots, p_N^l$ for a given l;
3 $C^l \leftarrow$ DAC-FFT$(p_1^l, \cdots, p_N^l)$;
4 **for** $o_i \in \mathcal{O}$ **do**
5 $\mathbb{P}(r(o_i) \leq k) \leftarrow 0$;
6 **for** $l = 1 : L$ **do**
7 $C[0] \leftarrow C^l[0]/p_i^l$;
8 **for** $ind = 1 : k-1$ **do**
9 $C[ind] \leftarrow (C^l[ind] - (1-p_i^l) \times C[ind-1])/p_i^l$;
10 $\mathbb{P}(r(o_i) \leq k) \leftarrow \mathbb{P}(r(o_i) \leq k) + (\sum_{ind=0}^{k-1} C[ind])\mathbb{P}(s(o_i) = l)$;
11 Return n items with highest $\mathbb{P}(r(o_i) \leq k)$;

all the items. Here, we propose two techniques to accelerate the calculation. We name this algorithm, Pundit, that answers the n-top-k query in Algorithm 2. Time complexity of Algorithm 2 is $O(Nlog^2N + Nk)$.

The first technique is to pre-compute the coefficient C^l of $\mathcal{F}' = \prod_{o_j \in \mathcal{O}} (\mathbb{P}(s(o_j) \leq l) + \mathbb{P}(s(o_j) > l)x)$ for all $1 \leq l \leq L$. Using the shorthand notation, we get $\mathcal{F}_i(x,l) = \mathcal{F}' \left[p_i^l + (1-p_i^l)x\right]^{-1}$. Thus, we need to compute the set of coefficients once for each l and all the coefficients can be deduced correspondingly. Secondly, we observe that explicit calculation of all the coefficients is not needed. For answering n-top-k query, we calculate only the first k coefficients $c_0(l), \cdots, c_{k-1}(l)$ as shown in Line 6-9. Following this, Line 10 computes the positional probability. Once the positional probabilities for all the N items are computed, the answer to the n-top-k query is the set of n items that have the highest positional probability.

3.5 Performance Evaluation

In this section, we compare performance of `Pundit` with two baseline algorithms `Mean Score Ranking` and `Monte Carlo`, as described in Sections 3.2 and 3.3.

Datasets and Set-up. We use the Amazon review dataset[1][16,15] with six categories of products– 'Apps for Android', 'Beauty', 'Books', 'Cell phones and Accessories', 'Electronics' and 'Movies and TVs'. Each review contains a rating for an item collected using a 5-valued Likert scale. This dataset is collected from May 1996 to July 2014. We set $k = 3$ as a default value. Actually, we find that varying k does not affect the results of `Pundit` much. All experiments are run on a Windows PC with 3.4 GHz Intel Core and 8 GB memory.

When evaluating the performance of the algorithms, we remove the items from each dataset which has less than 100 reviews. The remaining number of items N in each dataset is summarized in Table 1. We aggregate the ratings for each item to construct the score distributions of the items. We measure the efficiency and effectiveness of the three algorithms based on this filtered dataset. For effectiveness, `Pundit` returns the optimal ranking list for the n-top-k query while `Mean Score Ranking` and `Monte Carlo` algorithms return the approximate results.

[1] `http://jmcauley.ucsd.edu/data/amazon/`

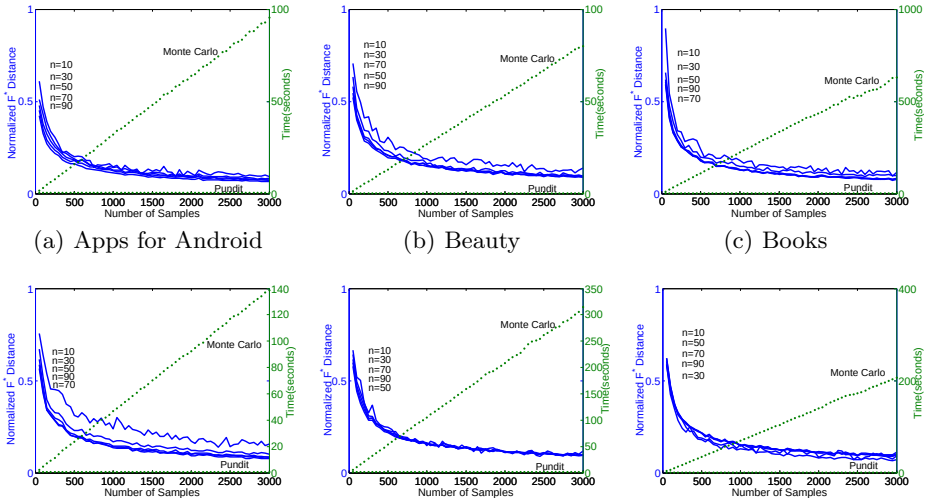
Table 1. `Pundit` vs. `Mean Score Ranking`

Dataset	N	Runtime(seconds)		Normalized F^* Distance				
		`Pundit`	`MeanScoreRanking`	$n = 10$	$n = 30$	$n = 50$	$n = 70$	$n = 90$
Apps for Android	3418	0.32	0.05	0.291	0.333	0.289	0.270	0.249
Beauty	2687	0.26	0.06	0.364	0.312	0.288	0.295	0.289
Books	28897	2.59	0.21	0.200	0.215	0.231	0.227	0.229
Cell phones and accessories	5245	0.48	0.09	0.527	0.441	0.420	0.379	0.358
Electronics	13077	1.17	0.12	0.309	0.215	0.241	0.266	0.268
Movies and TVs	8392	0.75	0.10	0.127	0.228	0.249	0.244	0.227

Metric of Comparison. We use a variation of Spearman's footrule distance, F^* [5], to measure the discrepancy between the list of n-k best-rated items returned by `Pundit` and that of the baseline algorithms. F^* distance measures the total displacement of the items which appear in either of the two lists. More specifically, $F^*(list_1, list_2) = \sum_{o_i \in list_1 \cup list_2} |r(o_i)^{list_1} - r(o_i)^{list_2}|$, where $r(o_i)^{list_a}$ is the rank of o_i in $list_a$. If o_i only appears in one of the two lists, say $list_1$, then replace $r(o_i)^{list_2}$ with a natural choice $n + 1$. We choose F^* distance for the reason that it is a metric and satisfies the interpolation criterion [5]. The maximum value of the F^* distance is $n(n + 1)$ for the lists of length n that share no items. F^* attends the minimum value 0 for the lists that have all the items in common. So, the normalized F^* distance equals to F^* distance divided by $n(n + 1)$. Shorter normalized F^* distance implies better approximation of the baseline algorithm.

Pundit versus Mean Score Ranking. Table 1 enlists runtime of `Pundit` and `Mean Score Ranking` respectively. In this experiment, we set n as the N of the corresponding category. We observe that `Pundit` is efficient. For example, `Pundit` is able to rank around 30,000 items of the 'Books' dataset in 3 seconds. `Mean Score Ranking` is faster than `Pundit`, since its time complexity is $O(Nlogn)$. But as shown in Section 3.2, ranking the items according to the mean score does not always return a correct answer. In Table 1, we also present the normalized F^* distance between the two lists of n-k best-rated items returned by `Pundit` and that returned by `Mean Score Ranking`. It elaborates change in the normalized F^* distance for different categories of items and also with increasing length of the list n. We observe that the normalized F^* distance decreases as n increases for 'Beauty' and 'Cell phones and Accessories' categories. But for the other categories the trend is not monotonous. Thus, we can hardly conclude about the influence of increasing n on the performance of `Mean Score Ranking`.

Pundit versus Monte Carlo. In figure 1, we depict the execution times of `Pundit` and `Monte Carlo` with increasing number of samples. We also instantiate the 'goodness' of approximation of the `Monte Carlo` algorithm in terms of the normalized F^* distance. All the results of `Monte Carlo` are averaged over 15 random runs. Intuitively, more samples improve performance of the `Monte Carlo` algorithm. This is observed in Figure 1 since the F^* distance decreases as the number of samples increases. However, as shown in Figure 1, execution time of the `Monte Carlo` method increases proportionally to the number of samples and to the number of items in the dataset. This shows that the `Monte Carlo` algorithm is not efficient. The `Pundit` algorithm, instead, can generate the exact n-k best-rated items within 5 seconds for all datasets.



(a) Apps for Android (b) Beauty (c) Books

(d) Cell Phones and Accessories (e) Electronics (f) Movies and TVs

Figure 1. `Pundit` vs. `Monte Carlo`[19]

4 How Many Ratings Are Enough?

Though we have formulated an exact algorithm, `Pundit`, for finding n-k best-rated item using the observed collection of ratings, the problem is not solved yet. For real applications, ratings of some items are either missing or insufficient. For example, more than 30000 books in the Amazon book dataset have only one rating while the entire population of our datasets is 8726569. If we try to find the 10 best-rated books from this dataset, we would get 10 books which are rated as 5-star by only one user. But the knowledge-base formed by one review is not only 'incomplete' but statistically insignificant and probably biased. Thus, the question that naturally appears is– "how many ratings are enough to build a knowledge-base that would be representative of the *true* one?" Since seeking for more ratings takes higher cost in terms of money or resources, answering the aforementioned question introduces a cost-effective perspective of crowdsourcing.

In Section 4.1, we investigate the estimation error of the score distribution of an item if we have a finite number of ratings. This model allows us to set a threshold in the required number of ratings for an item if we want to construct a knowledge-base without introducing remarkable error. We instantiate and evaluate applicability of the error model for such purpose in Section 4.3.

4.1 Score Distribution Error Model

Following the structure of Section 3.1, 'incomplete' knowledge-base formed by accumulation of a certain number of ratings is represented by *observed score distribution* f. We represent the 'true' knowledge-base about the quality of an item by the *oracle score distribution* f^* constructed with all the ratings from the universal user pool. For brevity, we call f^* and f the oracle distribution and the observed distribution respectively. Now, we model the discrepancy between the

oracle distribution and the *observed distribution* using KL-divergence. We apply it to analyse evolution of the observed distribution with accumulation of ratings.

The Optimization Problem. Let us consider the scenario when m reviews of an item are collected in the form of L-valued Likert scale. Suppose $z_1, \cdots, z_L$ are the number of ratings for each of the L values, such that $\sum_{i=1}^{L} z_i = m$. This is analogous to running m trials of an experiment that produces L possible outputs and obtaining the i^{th} output z_i times. If the probability of getting the i^{th} output is p_i^*, probability of reaching such a review pool can be represented by a multinomial distribution. Mathematically, we express the corresponding probability density function by Equation 4.

$$\mathbb{P}(z_1, \cdots, z_L) = \frac{m!}{z_1! \cdots z_L!} {p_1^*}^{z_1} \cdots {p_L^*}^{z_L}. \tag{4}$$

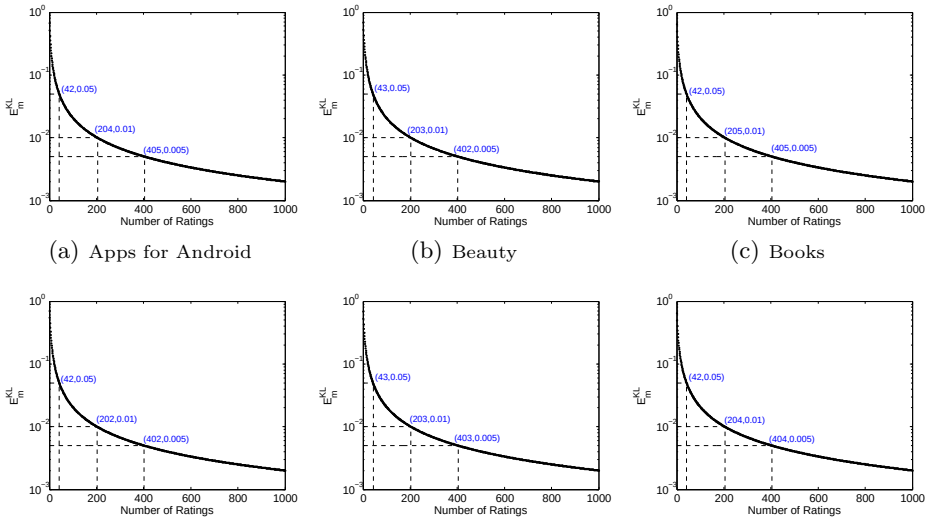
Here, p_i^* is the probability of an user rating the item i according to the oracle distribution. Thus, $\mathbb{P}(z_1, \cdots, z_L)$ is the probability of obtaining z_i users rating the item i after accumulation of m ratings. Now, we construct the observed score distribution f^m from these m ratings as a collection of empirical frequencies of an user rating a certain value $\frac{z_i}{m}$. In order to model the information gap between the observed distribution f^m and the oracle distribution f^*, we define a distance $Dist(f, f^*)$. This allows us to define the expected error of the observed score distribution by its expected distance from the oracle. Mathematically, we express the *expected score distribution error* as $E_m^{\mathrm{KL}} \triangleq \sum \mathbb{P}(z_1, \cdots, z_L)\ Dist(f^m, f^*)$. The sum is calculated over all $\{z_1, \cdots, z_L\}$ in the set of all possible L-partitions of m, $P(L, m)$. Thus, the expected error depends on three factors– the number of ratings m, the oracle distribution f^* and the distance function $Dist$. As m is given at an instance and the oracle distribution f^* is constructed with the universal review pool, modelling the expected error reduces to choice of the distance function. Since KL-divergence [10] quantifies the expected information per sample to discriminate between the uncertainty encompassed by one distribution against the other, we choose KL divergence as the eligible choice of distance function between the oracle and the observed score distributions. Thus, the expected error can be written as in Equation 5.

$$E_m^{\mathrm{KL}} = \sum_{\{z_1, \cdots, z_L\} \in P(L,m)} \left(\frac{m!\ {p_1^*}^{z_1} \cdots {p_L^*}^{z_L}}{z_1! \cdots z_L!} \sum_{i=1}^{L} \left(\frac{z_i}{m} \log \frac{z_i}{m p_i^*} \right) \right) \tag{5}$$

We want to find the minimal number of ratings m^* such that the expected error between the oracle and the observed distribution is less than a predefined threshold ϵ. Our objective is mathematically expressed in Equation 6.

$$m^* = \arg\min_m\ m \quad \text{such that, } E_m^{\mathrm{KL}} \leq \epsilon. \tag{6}$$

Efficient Solution. In order to compute m^* using Equation 6, we need to compute E_m^{KL}. E_m^{KL} depends on the oracle distribution, which is a choice, and the observed distribution f^m, which is an observable. The choice of f^* is crucial as we observe different oracle distributions introduce different representations of the true knowledge-base and different levels of difficulty for calculating the



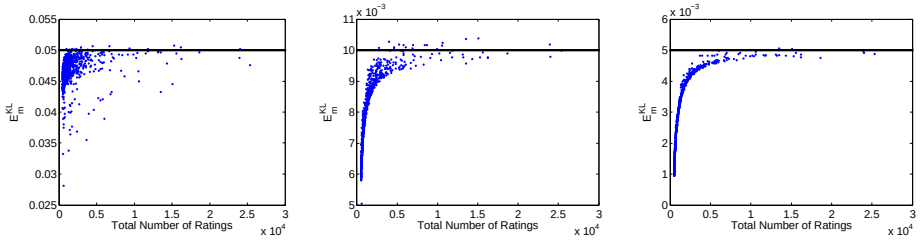
(a) Apps for Android (b) Beauty (c) Books

(d) Cell Phones and Accessories (e) Electronics (f) Movies and TVs

Figure 2. E_m^{KL} (Normalized) Between Oracle Distribution and Observed Distribution with Different Number of Ratings

score distribution error. For example, if the oracle distribution is $\{0, 0, 0, 0, 1\}$, then just one sample may lead to 0 error. As we focus on the method for efficient calculation of the error, let us assume f^* is either given as a model parameter or constructed from the user-pool of a given dataset. But even when the oracle distribution is given, the expected error is not easy to calculate. Because we sum over the set $P(m, L)$ that contains $\binom{m+L-1}{L-1}$ elements. It makes exact calculation of E_m^{KL} combinatorially expensive.

Thus, we propose a sampling approach to estimate the expected error based on the Ergodic Theorem [18]. It states that if we keep on sampling from an underlying distribution using an *aperiodic* and *irreducible* Markov chain and evaluate a function for each sample, then the mean of the function over the samples converges to the 'true' expected value of the function. For us, the underlying distribution is the oracle distribution. The target function is the KL divergence between the observed and the oracle distribution. Now, we construct a Markov chain [18] as our sampling scheme to approximately evaluate the expected error. We define a state in the chain as an L-tuple of ratings $\{z_1, \cdots, z_L\} \in P(L, m)$. Given the oracle distribution f^*, we generate a sample from the corresponding multinomial distribution as the initial state of the Markov chain. Similarly, we continue generating the next states by independently drawing samples from the multinomial distribution. For each state, we evaluate the KL divergence. The Markov chain is *aperiodic* since at any iteration the probability to return to the same step as the present one is positive. The Markov chain is *irreducible* since all states are connected. Thus, according to the Ergodic Theorem, as we repeatedly generate a chain of states using this scheme, the mean of the sampled error converges to E_m^{KL}. Once we are able to calculate a sufficient approximation of E_m^{KL} efficiently, we can formulate the relation between E_m^{KL} and m. This allows us to find the m^* either empirically or analytically.



(a) 43 Ratings (b) 205 Ratings (c) 405 Ratings

Figure 3. The Predicted E_m^{KL} (Normalized) versus the Observed E_m^{KL} (Normalized) with Total Number of Ratings of an Item

4.2 Experimental Investigation of Error Models

Dataset and Set-up. We use the Amazon review dataset described in Section 3.5. When evaluating the error model, we consider only the items with more than 500 reviews. All ratings of each of the items are accumulated to construct the oracle distribution for each of them. Once we obtain the oracle distributions, we focus on uncovering the relation between the score distribution error and the number of ratings m. In the experiments, we increase the number of ratings accumulated for the items of each category and then observe evolution of the error.

Score Distribution Error and Number of Ratings. In Figure 2, we present a smooth curve that fits the evolution of the error. We observe that the score distribution error decreases with increase in the number of ratings. This observation proves that this error model is consistent. Because the observed score distribution, that represents 'incomplete' knowledge-base, would converge to the oracle distribution, that represents the 'complete' knowledge-base, with accumulation of more ratings, i.e., information about the item. We also observe that it fits with the hyperbolic equation $E_m^{\mathrm{KL}} = \frac{c}{m}$. For the 6 categories, c is a constant between 2.01 and 2.024. The points shown on the curves reinstantiate this observation. Thus, decay of the expected score distribution error follows the inverse law. Evolution of the error is almost category independent as it quantifies the evolution of observed distribution with accumulation of ratings. The distributions 'homogenize' the items by reducing the corresponding knowledge-base into numbers on L-valued Likert scale. Thus, the score distribution error depends on the convergence process of an 'incomplete' knowledge-base to a 'complete' one by accumulation of ratings but not on the exact object names or categories.

4.3 Model in Action

As we have constructed the error model and analysed it, here we want to verify the effectiveness of the empirical laws to predict "how many ratings are enough". As the score distribution error follows the inverse law, thresholding the error to ϵ returns us an $m^* = \frac{c}{\epsilon}$. For example, the observed inverse law for *Apps for Android* items is $E_m^{\mathrm{KL}} = \frac{2.021}{m}$. This predicts that we need at least 405 ratings to reach error less than 0.005. Figure 2 also echoes this prediction. This experiment is to verify that if it is effective to use these empirical laws of error model to predict the number of ratings required to bound the error.

Since variance of the total number of ratings for an item is the largest for *Apps for Android* category, we choose it as the test case for our experiment. *Apps for Android* consists of 906 items with the total number of ratings varying from 500 to 25368. In this experiment, first we leverage the empirical laws to predict the m^* to reach below the error threshold ϵ. Then, we randomly pick m^* ratings for each item to construct the observed distribution and calculate the KL divergence between the observed distribution and the oracle distribution. We repeat this for 1000 times for each item and consider the mean of the normalized KL divergences as the corresponding score distribution error.

Figure 3 simultaneously plots the predicted score distribution error that should be reached with a predicted number of ratings (the black line) and the observed score distribution error for different objects under this amount of ratings (the blue dots). In particular, we show three settings with 43, 205 and 405 number of ratings respectively. Figure 3 shows that the observed score distribution error is almost always lower than the predicted error. Thus, this prediction method almost surely advises to collect the number of ratings that would never enhance the error above the desired threshold. But we also do not want the observed m^* to be much less than the predicted one. Since collecting ratings require investment of certain resources, thus a cost-effective prediction should never overpredict much beside not enhancing the error. Since p-value statistically quantifies deviation of the observed score distribution errors from the predicted ones, we calculate the p-values for 43, 205 and 405 ratings and they are respectively $1.64e-17$, $3.16e-275$ and $3.10e-293$. These values being less than 0.05 substantiate that the deviation of the number of ratings actually needed to reach the error threshold from the predicted m^* is not statistically large. Figure 3 also shows that the observed error asymptotically reaches the predicted error as the total number of ratings of an item, i.e., its user pool, gets larger. It is intuitive since the oracle constructed from a huge user-pool would need more number of ratings to construct an approximate representative of it. This shows that leveraging the empirical laws of score distribution error to predict the number of ratings required to reach an error threshold is a simple but effective approach.

5 Conclusion

In this paper, we study the problem of finding the *n-k* best-rated items by exploiting the accumulated ratings from the users. We devise an exact algorithm, Pundit, that solves this problem efficiently. We empirically and comparatively evaluate the effectiveness and efficiency of our approach and the competing ones with the Amazon review dataset. We develop the score distribution error model to quantify the effect of the accumulation of ratings and to answer "how many ratings are enough?". We instantiate this model using the Amazon review dataset to uncover the fact that the error follows the inverse law. Following that, we put this inverse law in action to predict minimal number of ratings that should be accumulated to meet a prescribed error. Experiments validate that this empirical approach is effective. Though we present only the score distribution error as the predictor, we are developing and testing another error model comparing the lists of *n-k* best-rated items obtained by Pundit. We are also theoretically analysing

the emergence of the inverse law. We are planning to deploy the error model as a prediction system that would recommend a crowdsourcing platform the number of ratings that has to be accumulated to meet a prescribed error.

References

1. Chen, X., Bennett, P.N., Collins-Thompson, K., Horvitz, E.: Pairwise ranking aggregation in a crowdsourced setting. In: WSDM. pp. 193–202 (2013)
2. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of computation 19(90), 297–301 (1965)
3. Der Kiureghian, A., Ditlevsen, O.: Aleatory or epistemic? does it matter? Structural Safety 31(2), 105–112 (2009)
4. Ekstrand, M.D., Riedl, J.T., Konstan, J.A., et al.: Collaborative filtering recommender systems. Foundations and Trends in Human–Computer Interaction 4(2), 81–173 (2011)
5. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. SIAM 17(1), 134–160 (2003)
6. Guo, S., Parameswaran, A., Garcia-Molina, H.: So who won?: dynamic max discovery with the crowd. In: SIGMOD. pp. 385–396 (2012)
7. Hua, M., Pei, J., Zhang, W., Lin, X.: Ranking queries on uncertain data: a probabilistic threshold approach. In: SIGMOD. pp. 673–686 (2008)
8. Jamieson, S., et al.: Likert scales: how to (ab)use them. Medical education 38(12), 1217–1218 (2004)
9. Johnson, N.L., Kemp, A.W., Kotz, S.: Univariate Discrete Distributions. John Wiley & Sons (2005)
10. Kullback, S., Leibler, R.A.: On information and sufficiency. The annals of mathematical statistics 22(1), 79–86 (1951)
11. Li, J., Deshpande, A.: Ranking continuous probabilistic datasets. VLDB 3(1-2), 638–649 (2010)
12. Li, J., Saha, B., Deshpande, A.: A unified approach to ranking in probabilistic databases. VLDB 2(1), 502–513 (2009)
13. Likert, R.: A technique for the measurement of attitudes. Archives of psychology 144, 1–55 (1932)
14. Liu, Q., Basu, D., Abdessalem, T., Bressan, S.: Top-k queries over uncertain scores. In: On the Move to Meaningful Internet Systems: OTM 2016 Conferences. pp. 245–262 (2016)
15. McAuley, J., Pandey, R., Leskovec, J.: Inferring networks of substitutable and complementary products. In: SIGKDD. pp. 785–794 (2015)
16. McAuley, J., Targett, C., Shi, Q., van den Hengel, A.: Image-based recommendations on styles and substitutes. In: SIGIR. pp. 43–52 (2015)
17. Niu, S., Lan, Y., Guo, J., Cheng, X., Yu, L., Long, G.: Listwise approach for rank aggregation in crowdsourcing. In: WSDM. pp. 253–262 (2015)
18. Robert, C.P., Casella, G.: Monte Carlo Statistical Methods. Springer (2004)
19. Soliman, M.A., Ilyas, I.F., Ben-David, S.: Supporting ranking queries on uncertain and incomplete data. VLDB 19(4), 477–501 (2010)
20. Venetis, P., Garcia-Molina, H., Huang, K., Polyzotis, N.: Max algorithms in crowdsourcing environments. In: WWW. pp. 989–998 (2012)
21. Ye, P., Doermann, D.: Combining preference and absolute judgements in a crowdsourced setting. In: ICML. pp. 1–7 (2013)
22. Zhang, X., Li, G., Feng, J.: Crowdsourced top-k algorithms: An experimental evaluation. VLDB 9(8), 612–623 (2016)