A Rating-Ranking Method for Crowdsourced Top-k Computation

Kaiyu Li, Xiaohang Zhang, Guoliang Li
Department of Computer Science and Technology, Tsinghua University, Beijing, China
liky15@mails.tsinghua.edu.cn,zhangxiaohang12@tsinghua.org.cn,liguoliang@tsinghua.edu.cn

ABSTRACT

Crowdsourced top-k computation aims to utilize the human ability to identify top-k objects from a given set of objects. Most of existing studies employ a pairwise comparison based method, which first asks workers to compare each pair of objects and then infers the top-k results based on the pairwise comparison results. Obviously, it is quadratic to compare every object pair and these methods involve huge monetary cost, especially for large datasets. To address this problem, we propose a rating-ranking-based approach, which contains two types of questions to ask the crowd. The first is a rating question, which asks the crowd to give a score for an object. The second is a ranking question, which asks the crowd to rank several (e.g., 3) objects. Rating questions are coarse grained and can roughly get a score for each object, which can be used to prune the objects whose scores are much smaller than those of the top-k objects. Ranking questions are fine grained and can be used to refine the scores. We propose a unified model to model the rating and ranking questions, and seamlessly combine them together to compute the top-k results. We also study how to judiciously select appropriate rating or ranking questions and assign them to a coming worker. Experimental results on real datasets show that our method significantly outperforms existing approaches.

ACM Reference Format:
Kaiyu Li, Xiaohang Zhang, Guoliang Li. 2018. A Rating-Ranking Method for Crowdsourced Top-k Computation. In *Proceedings of 2018 International Conference on Management of Data (SIGMOD/PODS '18).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3183762

1 INTRODUCTION

Given a set of objects, top-k computation aims to find the top-k objects from the set. Due to its widespread applications, it is widely used in many real-world systems, such as search engine and recommendation system. Top-k computation requires to use comparison operations to compare different objects. However, many comparison operations are hard to process by computers but can be easily compared by human, e.g., comparing the clarify of two photos, comparing the understanding difficulty of two sentences, and comparing answers' quality of a question in Yahoo! answers.

Recently some crowdsourcing platforms, e.g., Amazon Mechanical Turk and CrowdFlower have been deployed, and we can harness the crowd to compute top-k objects. However, the crowd may return incorrect answers, and traditional top-k algorithms cannot tolerate the errors from the crowd. Thus, crowdsourced top-k computation has been widely studied, which asks the crowd to compare objects and infers the top-k objects based on the crowdsourced comparison results. There are many real-world applications for crowdsourced top-k computing. For example, in Q&A systems, like Yahoo! Answer and Stack Overflow, we need to rank the answers of a question [18]. In image search, given 1000 photos of a restaurant, we want to find top-k photos that are most appealing and best describes the restaurant, for doing advertisements [16].

Existing algorithms[5, 16, 31, 37] utilize pairwise comparison to compute the top-k results, which asks the crowd to compare two objects and infers the top-k answers based on the comparison results. However, it is quadratic to compare every object pair. Since the crowd is not free, these methods involve huge monetary cost. For example, given 10 thousand objects, it requires to ask 50 million pairwise questions. To address this problem, we propose a rating-ranking based approach, which contains two types of questions. The first is a rating question, which asks the crowd to give a score for each object. The second is a ranking question, which asks the crowd to rank several (e.g., 3) objects. Rating questions are coarse grained and can roughly get a score for each object, which can be utilized to prune the objects whose scores are smaller than those of the top-k objects. Ranking questions are fine grained and can be used to refine the scores. We seamlessly combine the two types of questions together to compute the top-k results. Our method can significantly reduce the monetary cost.

Our approach contains two main steps. The first step infers top-k results based on the current answers of rating and ranking questions, called *top-k inference*. We model the score of each object as a Gaussian distribution, utilize the rating and ranking results to estimate the Gaussian distribution, and infer the top-k results based on the distributions. The second step selects questions for a coming worker, called *question selection*. Based on the probability of an object in the top-k results, we can get two distributions: real top-k distribution and estimated top-k distribution. Thus we propose an effective question selection strategy that selects questions to minimize the distance between the real distribution and the estimated distribution. As it is rather expensive to minimize the difference, we propose effective heuristics to improve the performance.

To summarize, we make the following contributions.
(1) We propose a rating-ranking based framework for crowdsourced top-k computation (Section 2). To the best of our knowledge, this is the first framework that uses the rating and ranking questions to compute the crowdsourced top-k answers.
(2) We design a unified model to model ranking questions and rating questions and seamlessly combine rating answers and ranking answers to infer the top-k results (Section 3).

SIGMOD/PODS '18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3183762

[1]Guoliang Li is the corresponding author.

(3) We devise an effective question-selection framework to judiciously assign questions to workers (Section 4).
(4) Experimental results on real datasets show that our method significantly outperforms existing approaches (Section 5).

2 A RATING-RANKING BASED FRAMEWORK

We first formulate the problem and then introduce truth model, question model and worker model. Next we present our framework.

2.1 Problem Formulation

Definition 2.1 (Crowdsourced top-k Computation). Given an object set $O = \{o_1, o_2, \cdots, o_n\}$, where the objects are comparable but hard to compare by machines, find a k-size object set $\mathcal{K} = \{o_1, o_2, \cdots, o_k\}$ where o_i is preferred to o_j (denoted by $o_i \prec o_j$) for $o_i \in \mathcal{K}$ and $o_j \in O - \mathcal{K}$.

For example, Table 1 shows six objects. Suppose we want to identify top-3 earliest released movies. o_1, o_2, o_3 are the top-3 results. If we only have movie titles or movie posters, it is hard to compute the top-3 results. Instead, we utilize the crowd to find top-3 results.

2.2 Question, Truth, Worker Models

Truth Model. We model each object o_i by a Gaussian distribution $\mathcal{N}(\mu_i, \sigma_i)$. If we have the value μ_i, we can easily sort the objects and compute the top-k results [31, 35, 43]. We will discuss how to compute the two parameters μ_i and σ_i based on workers' answers and how to utilize them to infer the top-k answers later. Our framework can be extended to support other data distributions of the object scores. Appendix A gives the details.

Question Model. To ask the crowd, we use two types of questions, rating questions and ranking questions, formally defined as below.

Definition 2.2 (Rating Question). Rating question asks the crowd to give a rating for an object o_i.

For example, consider object o_2="The Pianist" in Table 2. A rating question is to ask the crowd to provide the release year of the movie. Table 2 shows the answers for each rating question, where each question is assigned to 3 workers. The rating answers for o_2 are *(2002, 1997, 2005)*. The truth of o_2 is 2002.

Definition 2.3 (Ranking Question). Ranking question asks the crowd to rank y objects, e.g., $y = 3$ objects.

For example, Table 3 shows three ranking questions, where each question has 3 objects and is assigned to 5 workers. The ranking answers for (o_1, o_2, o_3) are $\{\langle o_2, o_1, o_3\rangle, \langle o_3, o_2, o_1\rangle, \langle o_1, o_2, o_3\rangle, \langle o_1, o_2, o_3\rangle, \langle o_1, o_3, o_2\rangle\}$. And the truth for (o_1, o_2, o_3) is $\langle o_1, o_2, o_3\rangle$.

Worker Model. We model worker quality for the rating question and the ranking question respectively.

(1) Worker Model for Rating Question. Let r_{ij} denote the rating result of worker w_j on object o_i, and $\mathcal{R}_i$ denote the set of answers on object o_i from all workers. Suppose we compute the average $\mu_i = \frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|} r_{ij}$ and variance $\sigma_i = \sqrt{\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}(r_{ij} - \mu_i)^2}$. Intuitively, if r_{ij} is closer to μ_i, the worker w_j has higher quality. We use the cumulative distribution function C to evaluate the closeness between r_{ij} and μ_i. If $r_{ij} \le \mu_i$, $C(r_{ij}; \mu_i, \sigma_i)$ denotes the cumulative distribution on the left of r_{ij} as shown in Figure 1(a). Obviously the larger the area on the left of r_{ij} is, the closer r_{ij} and μ_i are. Thus, $\frac{C(r_{ij};\mu_i,\sigma_i)}{1/2}$ can be used to evaluate the closeness. If $r_{ij} > \mu_i$, $1 - C(r_{ij}; \mu_i, \sigma_i)$ denotes the cumulative distribution on the right of r_{ij} as shown in Figure 1(b), and $\frac{1-C(r_{ij};\mu_i,\sigma_i)}{1/2}$ can be used to evaluate the closeness. Then combining the two cases, we can evaluate the worker quality of w_j on o_i.

Table 4: Notations.

O	a set of n objects
$\mathcal{R}$	all rating answers
$\mathcal{P}$	all ranking answers
$\mathcal{R}_i$	rating answers for object o_i
$\mathcal{P}_i$	ranking answers for ranking question p_i
r_{ij}	rating answer for o_i that answered by worker w_j
p_{ij}	ranking answer for p_i that answered by worker w_j
$\mathcal{E}^r_{w_i}$	rating quality estimation for worker w_i
$\mathcal{E}^p_{w_i}$	ranking quality estimation for worker w_i
μ_i	average value for object o_i
σ_i	variance value for object o_i
$\mathcal{R}_{w_i}$	ratings answered by worker w_i
$\mathcal{P}_{w_i}$	rankings answered by worker w_i
Pr_i	overall top-k probability estimation for o_i
C	cumulative distribution function
$\lambda_{r_{ij}}$	confidence for rating answer r_{ij}
$\lambda_{p_{ij}}$	confidence for ranking answer p_{ij}

Let $\mathcal{R}_{w_j}$ denote the set of rating answers by worker w_j. By considering all the questions worker w_j has answered, we can compute the rating accuracy for worker w_j, denoted by $\mathcal{E}^r_{w_j}$.

$$\mathcal{E}^r_{w_j} = \frac{\sum_{i=1}^{|\mathcal{R}_{w_j}|} 2\eta \cdot C(r_{ij}; \mu_i, \sigma_i) + 2(1-\eta)\cdot(1 - C(r_{ij}; \mu_i, \sigma_i))}{|\mathcal{R}_{w_j}|} \tag{1}$$

where

$$\eta = \begin{cases} 1 & \text{if } r_{ij} \le \mu_i \\ 0 & \text{if } r_{ij} > \mu_i \end{cases}$$

Based on the rating answers in Table 2, suppose we have computed the distributions for o_1, o_2, o_3: $\mathcal{N}(2004.7, 0.47)$, $\mathcal{N}(2001.3, 3.30)$, $\mathcal{N}(2000.3, 3.77)$. Consider worker w_1 who rates o_1, o_2, o_3. w_1 gives the rating result 2005 for o_1, 2002 for o_2, and 2003 for o_3. For the first rating result 2005 of o_1, as 2005 is on the right of the average value 2004.7, so $\eta = 0$ in this case. We should compute the right part of cumulative distribution as shown in Figure 1, which is $1 - C(2005; 2004.7, 0.47) = 0.26$. Then the closeness for rating result 2005 is computed as $\frac{1-C(2005;2004.7,0.47)}{1/2} = 0.52$. For the second rating result 2002 for o_2, we find that 2002 is still at the right part of the average value, so the closeness for 2002 is computed as $\frac{1-C(2002;2001.3,3.30)}{1/2} = 0.83$. For the last rating result 2003 for o_3, the rating result is on the right of the average value, then η is 0, and the closeness for 2003 is computed as $\frac{1-C(2003;2000.3,3.77)}{1/2} = 0.47$. So we can compute the average closeness for all the rating results answered by worker w_1 as $\frac{0.52+0.83+0.47}{3} = 0.61$. In the same way, we can compute the accuracy for workers w_2, w_3, w_4, w_5, w_6 as $(0.28, 0.30, 0.43, 0.17, 0.59)$.

(2) Worker Model for Ranking Question. Let p_{ij} denote the answer for a ranking question p_i by worker w_j. As p_i may have multiple answers from different workers, we need to aggregate them to generate a ranking p^*_i as the true ranking result for ranking question p_i. Then we can compute a distance between p_{ij} and p^*_i. The smaller

	Title	Truth
o_1	*A Beautiful Mind*	2001
o_2	*The Pianist*	2002
o_3	*Finding Nemo*	2003
o_4	*Million Dollar Baby*	2004
o_5	*Batman Begins*	2005
o_6	*The Departed*	2006

Table 1: Objects.

	o_1	o_2	o_3	o_4	o_5	o_6
w_1	2005	2002	2003	–	–	–
w_2	2004	1997	–	2004	–	–
w_3	2005	2005	1995	–	–	–
w_4	–	–	–	2004	2005	2006
w_5	–	–	–	1980	2015	2001
w_6	–	–	2003	–	2004	2004

Table 2: Rating Results

	(o_1, o_2, o_3)	(o_2, o_3, o_4)	(o_1, o_2, o_4)
w_3	–	$\langle o_3, o_2, o_4\rangle$	$\langle o_1, o_2, o_4\rangle$
w_4	$\langle o_2, o_1, o_3\rangle$	$\langle o_2, o_3, o_4\rangle$	–
w_5	$\langle o_3, o_2, o_1\rangle$	$\langle o_4, o_3, o_2\rangle$	$\langle o_4, o_1, o_2\rangle$
w_6	$\langle o_1, o_2, o_3\rangle$	–	$\langle o_2, o_1, o_4\rangle$
w_7	$\langle o_1, o_2, o_3\rangle$	$\langle o_2, o_3, o_4\rangle$	$\langle o_1, o_2, o_4\rangle$
w_8	$\langle o_1, o_3, o_2\rangle$	$\langle o_2, o_3, o_4\rangle$	$\langle o_1, o_2, o_4\rangle$

Table 3: Ranking Results

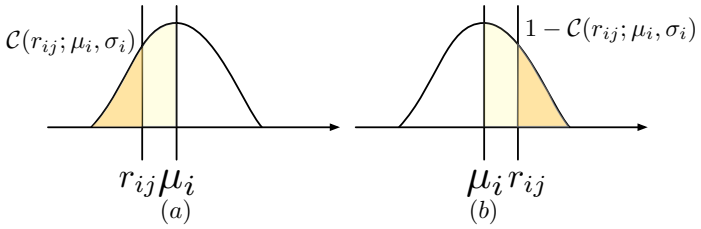


Figure 1: Rating Quality Estimation.

the distance is, the better the ranking answer p_{ij} is. Next we compute the average distance for all the ranking results from worker w_j. We use this average distance to represent the ranking quality for worker w_j. There are many functions to model this distance, and we use the Kendall tau distance [40] in this paper, which is the number of inverse object pairs between two rankings. Consider two ranking results p_{ij} and p_{il} for the same ranking question p_i by worker w_j and w_l. $p_{ij}[o_u]$ is the position of o_u in p_{ij} and $p_{il}[o_v]$ is the position of o_v in p_{il}. The Kendall tau distance is defined as:

$$\texttt{ktd}(p_{ij}, p_{il}) = \sum_{u<v} \mathbb{I}(u, v) \tag{2}$$

where

$$\mathbb{I}(u,v) = \begin{cases} 1 & \text{if } p_{ij}[o_u] < p_{ij}[o_v] \;\&\; p_{il}[o_u] > p_{il}[o_v] \\ 0 & \text{otherwise} \end{cases}$$

For example, the Kendall tau distance for $\langle o_1, o_2, o_3\rangle$ and $\langle o_1, o_3, o_2\rangle$ is 1, because the number of inverse pairs for the two rankings is 1.

Given a ranking question p_i, we compute an aggregated ranking result p_i^* that has the least number of inverse pairs compared with all the collected ranking results for p_i. Let $\texttt{ktd}(p_{it}, p_i^*)$ denote the Kendall tau distance between p_{ij} and p_i^*. The number of object pairs in ranking question p_i is $\binom{|p_i|}{2}$ which is the maximal Kendall tau distance among all ranking results p_{it} to p_i^*. Obviously, the larger $\texttt{ktd}(p_{ij}, p_i^*)$ is, the larger the number of inverse pairs is, and the lower quality p_{ij} is. Then we can use $\texttt{ktd}(p_{ij}, p_i^*)$ to compute the quality of p_{ij}. We use $(\frac{\binom{|p_i|}{2} - \texttt{ktd}(p_{ij}, p_i^*)}{\binom{|p_i|}{2}})$ to compute the quality of ranking answer p_{ij}. Assuming $p^* = \langle o_1, o_2, o_3\rangle$, and $p_{ij} = \langle o_2, o_1, o_3\rangle$, so we can compute the quality for ranking answer p_{ij} as $\frac{\binom{|p_i|}{2} - \texttt{ktd}(p_{ij}, p_i^*)}{\binom{|p_i|}{2}} = \frac{3-1}{3} = 67\%$.

Given the ranking answer set $\mathcal{P}_{w_j}$ of all the ranking results answered by worker w_j, we define the ranking quality of worker w_j as below.

$$\mathcal{E}^p_{w_j} = \frac{\sum_{i=1}^{|\mathcal{P}_{w_j}|} (\frac{\binom{|p_i|}{2} - \texttt{ktd}(p_{ij}, p_i^*)}{\binom{|p_i|}{2}})}{|\mathcal{P}_{w_j}|} \tag{3}$$

Algorithm 1: Crowdsourced Top-k Framework

Input: Objects O; Budget $\mathcal{B}$; Budget in each round b
Output: top-k highest ranked objects $\mathcal{K}$

1 $\mathcal{P} = \phi$; // collected ranking answers
2 $\mathcal{R} = \phi$; // collected rating answers
3 $\mathcal{E} = \phi$; // quality estimation for workers
4 **while** $\mathcal{B} > 0$ **do**
5 $\quad Q$ =Selection(O,b,$\mathcal{P}$,$\mathcal{R}$, $\mathcal{E}$);
6 $\quad$ Publish questions Q to the crowdsourcing platform;
7 $\quad$ Collect ratings $\bar{\mathcal{R}}$ and rankings $\bar{\mathcal{P}}$ from the crowd;
8 $\quad \mathcal{R} = \mathcal{R} + \bar{\mathcal{R}}$; $\mathcal{P} = \mathcal{P} + \bar{\mathcal{P}}$; $\mathcal{B} = \mathcal{B} - b$;
9 $\quad \mathcal{K}$ =Inference(O,$\mathcal{R}$, $\mathcal{P}$, $\mathcal{E}$);
10 Return $\mathcal{K}$;

For example, worker w_3 has given the ranking results $\langle o_3, o_2, o_4\rangle$, $\langle o_1, o_2, o_4\rangle$ for questions (o_2, o_3, o_4) and (o_1, o_2, o_4). For question (o_2, o_3, o_4), based on current ranking results in Table 3, we find that $\langle o_2, o_3, o_4\rangle$ is the best aggregated ranking result. The Kendall tau distance for $\langle o_3, o_2, o_4\rangle$ and $\langle o_2, o_3, o_4\rangle$ is 1. The number of object pairs in a the ranking result (o_2, o_3, o_4) is $\binom{3}{2} = 3$. So the quality for ranking result $\langle o_3, o_2, o_4\rangle$ is computed as $\frac{3-1}{3} = 0.67$. For question (o_1, o_2, o_4), based on ranking results as shown in Table 3, the best aggregated ranking result is $\langle o_1, o_2, o_4\rangle$. And the Kendall tau distance of the ranking result $\langle o_1, o_2, o_4\rangle$ answered by worker w_3 for (o_1, o_2, o_4) is 0 compared to the current best aggregated ranking result. Besides, the maximal Kendall tau distance for $\langle o_1, o_2, o_4\rangle$ and all the ranking results is 2. So the quality for ranking result $\langle o_1, o_2, o_4\rangle$ is computed as $1 - \frac{0}{2} = 1.0$. Then we can compute an average ranking quality for w_3 as $\frac{0.67+1.0}{2} = 0.83$. Similarly, based on Equation 3, we can compute the ranking accuracy for $(w_4, w_5, w_6, w_7, w_8)$ as $0.83, 0, 0.75, 1.0, 0.89$.

2.3 Algorithm Overview

Algorithm 1 shows the framework of our top-k algorithm. The algorithm contains two main components: top-k inference and question selection. When a worker requests questions, the question selection component selects some ranking or rating questions to ask (line 5). When a worker submits her answers, the top-k inference component infers the top-k answers based on the current results (line 9). Given a budget $\mathcal{B}$, it adopts an iterative strategy. In each iteration, it selects b rating and ranking questions and publishes all these questions to the crowdsourcing platform. After collecting the results of the asked questions, the algorithm will infer the top-k answers based on the current results. The algorithm terminates when the budget runs out.

2.3.1 Top-k Inference. The top-k inference includes two main steps. (1) It computes the average rating μ_i and variance σ_i based on the current results. (2) It infers the top-k results based on average μ_i and variance σ_i.

(1) Computing μ_i and σ_i. A simple way is to only use the rating results: $\mu_i = \frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|} r_{ij}$, $\sigma_i = \sqrt{\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}(r_{ij}-\mu_i)^2}$. However, this approach has some drawbacks. The first drawback is that it neglects the fact that the confidence for each rating answer is different. The second drawback is that this method only considers rating answers. To overcome these two shortcomings, we propose an effective way. Firstly, we compute a weight for each rating and ranking answer to denote the confidence of the answers. Secondly, we propose a model that combines rating and ranking answers together. By optimizing this model, we get an estimation for each Gaussian parameter μ_i and σ_i. Section 3 gives the details.

(2) Inferring top-k results. A simple way is to sort the objects based on μ_i in an ascending order and identify the first k objects as top-k results. However, if the estimation of μ_i is not accurate, this method will have low quality, because some objects should be in top-k results but their μ values are not accurately estimated. To address this problem, we first select a pivot value p_x, and then compute the probability for o_i in the top-k results, which is probability that μ_i smaller than p_x and can be computed by the cumulative distribution function as below

$$Pr_i = C(p_x; \mu_i, \sigma_i) \tag{4}$$

which is the size of the left area of the cumulative distribution for o_i in Figure 2.

Then we will sort all the overall top-k probabilities in descending order, and select the k objects with the largest probabilities as the top-k results.

Note that it is rather important to select a good value of p_x since different values have different effects on the top-k results. A small value will lead to that many objects have probability of 0 and a large value will cause that many objects have probability of 1, and thus it is hard to differentiate such objects. To address this issue, we propose an effective method to estimate p_x.

Once the value of pivot p_x is determined, based on the current Gaussian distribution for each object, the overall top-k probability for each object is determined based on Equation 4. Then we will sort all the overall top-k probabilities in a descending order, and get a discrete probability curve D_R as shown in Figure 3. Note that in the ground truth, an object will either belong to top-k set $\mathcal{K}$ or not. So the overall top-k probability for each object in the ground truth is 1 or 0. If we sort all the ground truth overall top-k probability in descending order, then we will get a two-stage discrete curve D_G as shown in Figure 3. Thus we want to select p_x that minimize the KL distance between D_G and D_R. To achieve this goal, we can enumerate every μ_i and select the one minimizing the KL distance. Note that actually, we only need to enumerate the objects close to μ_k to achieve high performance.

2.3.2 Question Selection. Based on the two distributions in Figure 3, i.e., top-k probability distribution for ground truth D_G and the estimated probability distribution D_R, our goal in question selection is to narrow the distance between D_G and D_R as soon as possible by utilizing both rating and ranking questions. As mentioned in Section 2.3.1, the distance is evaluated by the relative entropy of D_G and D_R, which is $KL(D_G||D_R)$. For rating questions, we enumerate every object, and compute the expected improvement for the probability curve respectively. For ranking questions, assuming the size of the ranking question is y, there are total $\binom{n}{y}$ combinations of objects. We calculate the expected improvement of the probability curve for each combination. Then we select the question to minimize the distance $KL(D_G||D_R)$ from those n rating questions and $\binom{n}{y}$ ranking questions. If multiple questions should be selected, we select questions based on the distance in descending order. However, there are two challenges. The first is how to estimate the answer of a worker if a question is assigned to the worker and how to estimate the probability distribution. The second is that the complexity for the enumeration algorithm is high. We will address these two challenges in Section 4.

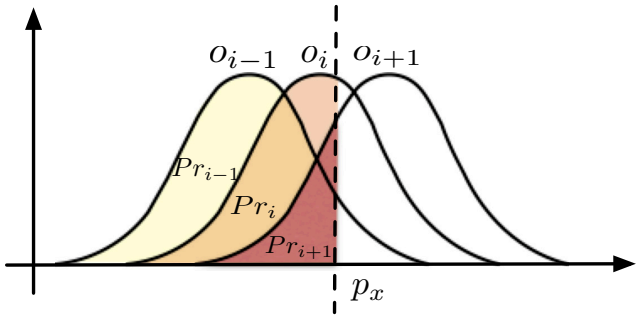


Figure 2: Top-k probability of o_{i-1}, o_i, o_{i+1}.

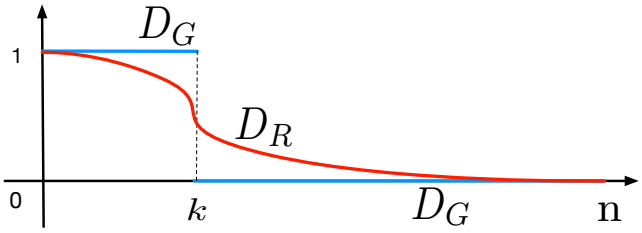


Figure 3: Ground truth distribution D_G and estimated distribution D_R.

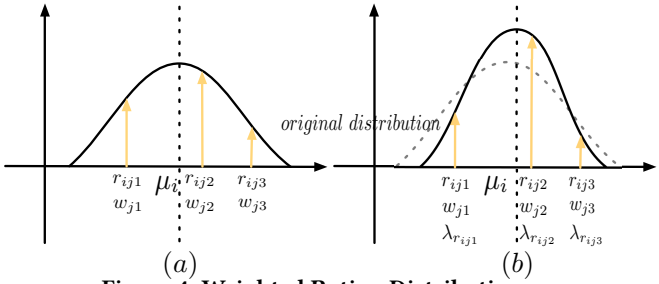


Figure 4: Weighted Rating Distribution.

3 INFERENCE METHOD

This section focuses on how to effectively compute the average μ_i and variance σ_i. We first compute a confidence for each rating or ranking result, and then aggregate all the rating and ranking results together utilizing the calculated confidences to compute the average and variance.

3.1 Computing Confidence For Each Result

The confidence for each rating or ranking result is different. The larger the confidence, the more reliable of the result. Besides, based on the value of confidence, we can effectively narrow the variance for each object. To compute the confidence, we consider two factors, worker's accuracy and the distribution of the answers.

3.1.1 Computing Confidence For Rating Result. We are more confident for a rating answer given by a worker with higher accuracy. In example of Figure 4, we can see o_i is labeled by three workers w_{j1}, w_{j2}, w_{j3}, their rating answers are respectively $r_{ij1}, r_{ij2}, r_{ij3}$. We assume their rating accuracies in descending order are respectively $\mathcal{E}^r_{w_{j2}}, \mathcal{E}^r_{w_{j1}}, \mathcal{E}^r_{w_{j3}}$. Thus we believe r_{ij2} is more reliable than r_{ij1}, which in turn is more reliable than r_{ij3}.

We expect that the variance of an object is small such that the distribution overlap of different objects are small, because with a smaller value of variance, the object can be more distinguishable, and we can narrow the distance of the current probability curve and the ground truth probability curve faster. To this end, we want to assign the rating answers close to the average result with a large weight, and assign the rating answers having large distance to the average result with a small weight. Based on this idea, we compute the distribution weight e^r_{ij} for rating result r_{ij} based on the probability density function as below.

$$e^r_{ij} = f(r_{ij}; \mu_i, \sigma_i) = \frac{1}{\sqrt{2\sigma_i^2\pi}} exp[-\frac{(r_{ij}-\mu_i)^2}{2\sigma_i^2}] \tag{5}$$

Then we can combine the worker accuracy and rating answer distribution together to compute a confidence for each rating result as follows

$$\lambda_{r_{ij}} = \sqrt{(\mathcal{E}^r_{w_j})^2 + (e^r_{ij})^2} \tag{6}$$

Therefore, $\lambda_{r_{ij1}} = \sqrt{(\mathcal{E}^r_{w_{j1}})^2 + (e^r_{ij1})^2}$, $\lambda_{r_{ij2}} = \sqrt{(\mathcal{E}^r_{w_{j2}})^2 + (e^r_{ij2})^2}$, $\lambda_{r_{ij3}} = \sqrt{(\mathcal{E}^r_{w_{j3}})^2 + (e^r_{ij3})^2}$.

Based on the confidence for each rating answer, we can compute a weighted average and variance values based on Equation 7 and Equation 8.

$$\mu_i = \frac{\sum_{j=1}^{|\mathcal{R}_i|} \lambda_{r_{ij}} r_{ij}}{\sum_{j=1}^{|\mathcal{R}_i|} \lambda_{r_{ij}}} \tag{7}$$

$$\sigma_i^2 = \frac{\sum_{j=1}^{|\mathcal{R}_i|} \lambda_{r_{ij}} (r_{ij}-\mu_i)^2}{\sum_{j=1}^{|\mathcal{R}_i|} \lambda_{r_{ij}}} \tag{8}$$

In Equations 7 and 8, $\lambda_{r_{ij}}$ is the confidence for this rating result. After the computation of the weighted Gaussian distribution, we will obtain a new probability distribution as shown in Figure 4(b). Based on the new probability distribution, we can update the accuracy estimation for w_{j1}, w_{j2} and w_{j3} using Equation 1. Moreover, in the new distribution Figure 4(b), the distribution weight e_{ij1}, e_{ij2} and e_{ij3} are also updated. Thus we can iteratively update the confidence $\lambda_{r_{ij}}$ for each rating result r_{ij}, and compute a new probability distribution. This procedure will continue until the confidences λ are finally converge. By utilizing λ, after the iteratively computation of Equations 7 and 8, we can narrow the variance for each object.

For example, the rating answers for object o_2 are {2002, 1997, 2005}. If we only consider the rating result itself, then the average value for o_2 is 2001.3, and the variance is 3.3. Based on the probability density function of $\mathcal{N}$ (2001.3, 3.3), the probability for 2002 is 0.118, and the probability for 1997 is 0.051, and the probability for 2005 is 0.065. So we think that for object o_2, 2002 is more reliable if only considering the data distribution itself.

Next we discuss how to compute the worker accuracy by taking o_2 as an example, which is labeled by worker $\{w_1, w_2, w_3\}$. Based on all the rating answers, the rating accuracy for them are {0.61, 0.275, 0.301} based on Equation 1. Then the confidences for $\{(w_1, 2002), (w_2, 1997), (w_3, 2005)\}$ are {0.61, 0.28, 0.31}. Based on the confidence we can compute a new distribution for o_2, and the new average is 2001.6, and the new variance is 1.63. When the confidences for rating answers are gradually converged, the confidences for $\{(w_1, 2002), (w_2, 1997), (w_3, 2005)\}$ are {0.57, 0.34, 0.23}.

3.1.2 Computing Confidence For Ranking Result. To compute the confidence for ranking answers, the basic idea is similar to the confidence computation of rating answers. We still consider worker's quality and the distribution of the ranking answers. We have defined how to compute the worker quality.

Recall that p_i^* is the aggregated ranking result based on the ranking answers. p_{ij} is the answer for question p_i by worker w_j. $\texttt{ktd}(p_i^*, p_{ij})$ is the Kendall tau distance (i.e., the number of inverse pairs) between p_i^* and p_{ij}. $\binom{|p_i|}{2} - \texttt{ktd}(p_{ij}, p_i^*)$ is the number of pairs with the same order between p_i^* and p_{ij}. Thus the larger $\binom{|p_i|}{2} - \texttt{ktd}(p_{ij}, p_i^*)$ is, the higher the confidence is. Thus we use e^p_{ij} to capture the weight for ranking answer p_{ij} according to the distribution of ranking answers.

$$e^p_{ij} = \frac{\binom{|p_i|}{2} - \texttt{ktd}(p_{ij}, p_i^*)}{\binom{|p_i|}{2}} \tag{9}$$

If we also take the worker's ranking accuracy into consideration, then the confidence for each ranking answer can be computed based on the following equation

$$\lambda_{p_{ij}} = \sqrt{(\mathcal{E}^p_{w_j})^2 + (e^p_{ij})^2} \tag{10}$$

Based on the confidence for each ranking answer, we can recompute a new aggregated ranking result according to Equation 11.

$$p_i^* = \arg\max \sum_{j=1}^{|\mathcal{P}_i|} \frac{\lambda_{p_{ij}}}{1 + \texttt{ktd}(p_i^*, p_{ij})} \tag{11}$$

In Equation 11, $\lambda_{p_{ij}}$ is the confidence for ranking answer p_{ij}. $\texttt{ktd}(p_i^*, p_{ij})$ is the Kendall tau distance for p_i^* and p_{ij}. $\mathcal{P}_i$ denotes all the ranking answer of p_i. Usually, the size of the ranking question is smaller than 10, so we can enumerate all the permutations of the ranking question, and find the permutation with the largest value of Equation 11 as the aggregated ranking result. According to p_i^*, we can update the distribution weight for each ranking answer based on Equation 9, and update the ranking accuracy for workers that answering question p_i using Equation 3. At last, we can recalculate the confidence for each ranking answer via Equation 10. The above process will continue until the confidences λ of the ranking answers is finally converge.

For example, in Table 3, the ranking results for (o_2, o_3, o_4) are $\{\langle o_3, o_2, o_4\rangle, \langle o_2, o_3, o_4\rangle, \langle o_4, o_3, o_2\rangle, \langle o_2, o_3, o_4\rangle, \langle o_2, o_3, o_4\rangle\}$. They are respectively answered by w_3, w_4, w_5, w_7, w_8. Based on Equation 9, the distribution weights for the ranking answers are respectively 0.67, 1.0, 0, 1.0, 1.0. Besides, based on Equation 3, the ranking accuracies for workers w_3, w_4, w_5, w_7, w_8 are 0.83, 0.83, 0, 1.0, 0.89. So the confidences for $(w_3, \langle o_3, o_2, o_4\rangle)$, $(w_4, \langle o_2, o_3, o_4\rangle)$, $(w_5, \langle o_4, o_3, o_2\rangle)$, $(w_7, \langle o_2, o_3, o_4\rangle)$, $(w_8, \langle o_2, o_3, o_4\rangle)$ are 1.07, 1.30, 0, 1.41, 1.34. Based on the confidence of each ranking answer, the next aggregated ranking result for (o_2, o_3, o_4) is still $\langle o_2, o_3, o_4\rangle$. So the distribution weight and the worker's ranking accuracy are converged.

3.1.3 Iterative Confidence Estimation. Initially we need to equally initialize the confidence for each rating and ranking result. Then based on the initialization of the confidences, we can compute a weighted Gaussian distribution for each object using Equations 7 and 8. Besides, an aggregated ranking result p_i^* can also be computed based on Equation 11. Therefore, according to the current aggregated results, we can update the rating and ranking accuracy for each worker based on Equations 1 and 3. At last we will recalculate the confidences for all the answers. The above process will continue until all the confidences are finally converge.

3.2 Combing Rating and Ranking

We present how to combine rating and ranking answers to compute the average μ_i and variance σ_i. To achieve this goal, we first use the Plackett-Luce ranking model which can obtain a score for each object based on the ranking results and then propose a unified model to combine the ranking results and rating results.

3.2.1 Top-k Inference Algorithm. The Plackett-Luce model [15, 26] is a widely used ranking model that can be utilized to compute the probability for a permutation. Consider three objects o_i, o_j and o_l and a ranking sequence $p = \langle o_i, o_j, o_l\rangle$. The probability for permutation p by the general form of Plackett-Luce model is computed as

$$P_s(p) = \frac{\phi(s_i)}{\phi(s_i)+\phi(s_j)+\phi(s_l)} \cdot \frac{\phi(s_j)}{\phi(s_j)+\phi(s_l)} \cdot \frac{\phi(s_l)}{\phi(s_l)} \tag{12}$$

$P_s(p)$ is the likelihood for permutation p, and ϕ is an increasing and strictly positive function. And s_i, s_j, s_l are the scores for objects o_i, o_j, o_l. Obviously $C(p_x; \mu_i, \sigma_i)$ is an increasing and strictly positive function and we can use it to replace $\phi(s_i)$. Then we can update the scores s_i, s_j, s_l by maximizing $P_s(p)$ as follows.

Supposing we have a set of ranking answers $p_{11} = \langle o_i, o_j, o_l\rangle$, $p_{12} = \langle o_j, o_i, o_l\rangle$, $p_{13} = \langle o_i, o_l, o_j\rangle$. Then we want to compute (μ_i, σ_i), (μ_j, σ_j) and (μ_l, σ_l) by the following maximum likelihood estimation.

$$\mathcal{L}_1 = \sum_{t=1}^{3} \log\left(\frac{C_{p_{1t}(1)}}{C_{p_{1t}(1)}+C_{p_{1t}(2)}+C_{p_{1t}(3)}} \cdot \frac{C_{p_{1t}(1)}}{C_{p_{1t}(1)}+C_{p_{1t}(2)}} \cdot \frac{C_{p_{1t}(3)}}{C_{p_{1t}(3)}}\right) \tag{13}$$

where $p_{1t}(j)$ represents the j-th object in permutation p_{1t} and $C_{p_{1t}(j)}$ is the corresponding cumulative distribution for object $p_{1t}(j)$. For example, the first object in p_{11} is o_i, so $p_{11}(1) = o_i$. The second object in p_{11} is o_j, so $p_{11}(2) = o_j$. $C_{p_{11}(1)} = C(p_x; \mu_i, \sigma_i)$, $C_{p_{11}(2)} = C(p_x; \mu_j, \sigma_j)$. We take the general BFGS(Broyden-Fletcher-Goldfarb-Shanno) [13] optimizer to optimize $\mathcal{L}_1$ in Equation 13. Then $(\mu_i = -1.18, \sigma_i = 7.26)$, $(\mu_j = 4.48, \sigma_j = 6.25)$, $(\mu_l = 12.08, \sigma_l = 0.07)$.

Next we also want to add the rating answers into the likelihood function. Suppose o_i, o_j, o_l have three rating answers. And we can reformulate the likelihood function $\mathcal{L}_1$ to $\mathcal{L}_2$ as follows

$$\begin{aligned}\mathcal{L}_2 = &\sum_{t=1}^{3} \log\left(\frac{C_{p_{1t}(1)}}{C_{p_{1t}(1)}+C_{p_{1t}(2)}+C_{p_{1t}(3)}} \cdot \frac{C_{p_{1t}(1)}}{C_{p_{1t}(1)}+C_{p_{1t}(2)}} \cdot \frac{C_{p_{1t}(3)}}{C_{p_{1t}(3)}}\right) \\ &+ \sum_{y=1}^{3} \log(f(r_{iy}; \mu_i, \sigma_i)) + \sum_{y=1}^{3} \log(f(r_{jy}; \mu_j, \sigma_j)) + \sum_{y=1}^{3} \log(f(r_{ly}; \mu_l, \sigma_l))\end{aligned} \tag{14}$$

In Equation 14, r_{i*} represents the rating answer for object o_i. f is the probability density of the normal distribution. After optimizing the new likelihood function $\mathcal{L}_2$ by Maximum likelihood estimation, we can compute the value by combing both rating and ranking questions. However the rating and ranking have different weights, next we formally combine them together based on the following likelihood function.

$$\begin{aligned}\mathcal{L} = &\tau \sum_{i=1}^{|\mathcal{P}|} \sum_{p_{ij} \in \mathcal{P}_i} \lambda_{p_{ij}} \log\left[\prod_{l=1}^{|p_{ij}|} \frac{C_{p_{ij}(l)}}{\sum_{h=l}^{|p_{ij}|} C_{p_{ij}(h)}}\right] \\ &+ (1-\tau) \sum_{i=1}^{|\mathcal{R}|} \sum_{r_{ij} \in \mathcal{R}_i} \lambda_{r_{ij}} \log f(r_{ij}; \mu_i, \sigma_i)\end{aligned} \tag{15}$$

In Equation 15, $\mathcal{P}$ is the set of all the ranking questions, $\mathcal{P}_i$ is a ranking question, p_{ij} is the ranking answer for question $\mathcal{P}_i$ answered by worker w_j, $p_{ij}(l)$ is the l_{th} object in ranking answer p_{ij}, and $C_{p_{ij}(l)}$ is the cumulative distribution $C(p_x; \mu_{p_{ij}(l)}, \sigma_{p_{ij}(l)})$. $\mathcal{R}$ is the set of all the rating questions, $\mathcal{R}_i$ is a rating question, r_{ij} denotes rating answer for question $\mathcal{R}_i$ answered by worker w_j, and f is the probability density of the normal distribution. In addition, $\lambda_{p_{ij}}$ is the confidence for ranking answer p_{ij}, and $\lambda_{r_{ij}}$ is the confidence for ranking answer r_{ij}. τ is a tuning factor to control the weight of rating and ranking results. If the rating result is more confident, τ is smaller than 0.5; otherwise τ is larger than 0.5. Next we optimize $\mathcal{L}$ by maximum likelihood estimation to compute the average μ and variance σ for all objects. To optimize Equation 15, we use gradient descent to update the estimation for μ and σ.

For example, in Figure 1, if we only utilize the rating answers, then the top-2 objects will be o_2 and o_3. By introducing ranking answers, after the optimization of Equation 15, we get a new estimation for the gaussian parameters. $\mu_1 = 1998.5, \sigma_1 = 4.7$, $\mu_2 = 2000, \sigma_2 = 4.5$, $\mu_3 = 2003.5, \sigma_3 = 5.5$, $\mu_4 = 2009.6, \sigma_4 = 7.8$, $\mu_5 = 2007.6, \sigma_5 = 5.2$, $\mu_6 = 2003.5, \sigma_6 = 4.5$. After enumerating all 6 average values from μ_1 to μ_6, we find that μ_2 is most suitable for p_x. Then the overall top-k probabilities are $Pr_1 = 0.63, Pr_2 = 0.5, Pr_3 = 0.26, Pr_4 = 0.11, Pr_5 = 0.07, Pr_6 = 0.22$. Based on the refined estimation, we find that the top-2 results are o_1 and o_2.

Remark. In Equation 15 we use a linear combination to combine rating and ranking, and thus the order of rating and ranking will not affect the inference result. We can use some golden tasks (i.e., mixing some tasks with ground truth) to determine the weight τ. Based on the golden tasks, we can compute the ranking quality q_{rank} and rating quality q_{rate}. Then we can compute weight of rating as $\tau = \frac{q_{rate}}{q_{rate}+q_{rank}}$.

4 QUESTION SELECTION METHOD

Question selection is a key component in the crowdsourced top-k framework. Different ways of selecting questions will have different effects. Previous selection algorithms [5, 31, 43] focus on getting a total ranking order for all objects. However, we only care about the top-k results in this paper. To this end, we first formulate the question selection problem and propose an optimal algorithm in Section 4.1. As the complexity of the optimal algorithm is too high, we propose a heuristic algorithm in Section 4.2.

4.1 Optimal Question Selection

Figure 3 shows our motivation in the question selection. The blue curve D_G is the real overall top-k probability, where each element in the x-axis is an object sorted by the probability in the top-k answers in descending order and y-axis is the corresponding probability. D_R is the estimated top-k probability. Initially, the overall top-k probability for each object is $\frac{k}{n}$. After we collect some rating and ranking answers, we have a knowledge of the objects and the overall top-k probability for each object will be updated as D_R. So we want to select questions to narrow the gap between the estimated probability curve (e.g. D_R) and the real probability curve (D_G) as soon as possible. In addition, we assume the current probability curve is D_i. And the distance for D_i and D_G is computed by KL distance $KL(D_G||D_i)$.[2]

We first discuss the case that assigning one question to a worker and then extend it to assigning multiple questions. The algorithm contains the following steps.
(1) Enumeration. 1.1) Enumerate each possible rating question and possible ranking question. 1.2) Estimate the possible answer for the rating question and ranking question. 1.3) Update μ_i and σ_i based on the estimated answers using the top-k inference technique. 1.4) Compute the top-k probability of each object based on Equation 4 and get the updated probability. 1.5) Compute the KL distance.
(2) Selection. Select question with the minimal KL distance.

Next we discuss how to address step 1.2. For each rating question, e.g., o_j, we use the expectation for o_j as the estimated answer by the worker, i.e., μ_j. Thus $\mathcal{R}_j = \mathcal{R}_j + \{\mu_j\}$. Then we bring the new rating answer $\mathcal{R}_j$ and all the related ranking answers that involved object o_j into the inference component in Section 3. Next we recompute the confidence λ for every rating answer in $\mathcal{R}_j$. And we recalculate (μ_j, σ_j) by Equation 15 considering all rating answers of $\mathcal{R}_j$ and all the related ranking answers that involved o_j. Based on the new values of (μ_j, σ_j), we can recalculate the overall top-k probability Pr_j for o_j. Suppose the current curve is D_i. Then the probability curve will be changed to D_i'. We then compute the improvement Δ_{r_j} of rating question r_j as $\Delta_{r_j} = KL(D_G||D_i) - KL(D_G||D_i')$.

For ranking questions, taking $p_1 = (o_1, o_2, o_3)$ as an example, we enumerate every permutation $p_{11} = \langle o_1, o_2, o_3\rangle$, $p_{12} = \langle o_1, o_3, o_2\rangle$, $p_{13} = \langle o_2, o_1, o_3\rangle$, $p_{14} = \langle o_2, o_3, o_1\rangle$, $p_{15} = \langle o_3, o_1, o_2\rangle$, $p_{16} = \langle o_3, o_2, o_1\rangle$ of the three objects, and compute the probability for each permutation. For example, one possible ranking answer is $p_{11} = \langle o_1, o_2, o_3\rangle$. We assume each object is independent. Then the difference of two independent Gaussian distributions also follows the Gaussian distribution. For example, the difference of $o_1 - o_2$ follows the Gaussian distribution $\mathcal{N}(\mu_1 - \mu_2, \sqrt{\sigma_1^2 + \sigma_2^2})$. The probability for o_1 smaller than o_2 is $C(0; \mu_1 - \mu_2, \sqrt{\sigma_1^2 + \sigma_2^2})$. So the probability for ranking answer p_{11} is computed as

$$\mathcal{F}_{p_{11}} = \prod_{i=1}^{2} \prod_{j=i+1}^{3} C(0; \mu_i - \mu_j, \sqrt{\sigma_i^2 + \sigma_j^2}) \tag{16}$$

Based on this ranking answer $p_{11} = \langle o_1, o_2, o_3\rangle$, the ranking answer set $\mathcal{P}_1$ will become to $\mathcal{P}_1 = \mathcal{P}_1 + \{p_{11}\}$. According to the new value of $\mathcal{P}_1$, we calculate the new confidence λ for each ranking answer in $\mathcal{P}_1$. Then we utilize $\mathcal{P}_1$ and other rating and ranking answers that included at least one of $\{o_1, o_2, o_3\}$ to re-estimate the value of (μ_1, σ_1), (μ_2, σ_2) and (μ_3, σ_3) by Equation 15. Then the overall top-k probability of Pr_1, Pr_2, Pr_3 can be recomputed, and we get a new probability curve D_i'. So the improvement of p_{11} is computed as $\Delta_{p_{11}} = KL(D_G||D_i) - KL(D_G||D_i')$. Similarly, we can compute the improvement for $p_{12}, p_{13}, p_{14}, p_{15}, p_{16}$, as $\Delta_{p_{12}}, \Delta_{p_{13}}, \Delta_{p_{14}}, \Delta_{p_{15}}$, $\Delta_{p_{16}}$, and the probability for them can be computed as $\mathcal{F}_{p_{12}}, \mathcal{F}_{p_{13}}$, $\mathcal{F}_{p_{14}}, \mathcal{F}_{p_{15}}, \mathcal{F}_{p_{16}}$ that are similar to Equation 16. So the expected improvement by asking question p_1 is computed as $\sum_{i=1}^{6} \mathcal{F}_{p_{1i}} \cdot \Delta_{p_{1i}}$.

Assuming the size of ranking questions is y. We need to enumerate all the combinations of the objects O and calculate the expected improvement for $\binom{n}{y}$ candidate ranking questions. Afterwards, we combine all the rating and ranking questions together, and select questions in descending order of the expected improvement. As there are total $n + \binom{n}{y}$ questions, if we utilize Batch Gradient Descent in optimizing Equation 15, the complexity is $O(n^{2(y+1)})$. If n is large, the complexity of the enumerate algorithm is too high to be utilized in real online question selection scenarios. To address this issue, we propose a heuristic algorithm in Section 4.2.

4.2 Heuristic Selection Method

We propose a probability-based method to reduce the complexity of question selection from $\binom{n}{y}$ to $\binom{m}{y}$, where m is a constant number. If m is large, e.g., close to n, the performance improvement is limited but it keeps as high quality as the optimal method; if m is small, e.g., close to y, the performance improvement is significant, but it has low quality as it prunes many objects. Moreover, as we only consider m rather than n objects, we can also improve the performance of Gradient Descent for optimizing Equation 15.

Initially, we publish some rating questions to the crowdsourcing platform and collect the rating results. We sort the objects in decreasing order according to the μ_i of each object o_i and get the top-k result $\mathcal{A}$. Then we get two types of objects. The first is the top-k objects in ground truth but not be selected as top-k in $\mathcal{A}$. The second is top-k objects in $\mathcal{A}$ but not the real top-k in the ground truth. In the following iterations, we publish rating and ranking questions for these two types of objects rather than for the n objects.

It is hard to detect these two types of objects. We propose Algorithm 2 to detect these objects and generate new rating and ranking questions in each iteration (to improve line 5 in Algorithm 1). For each object o_i, we use the rating answer $\mathcal{R}$ and workers' Quality $\mathcal{E}$ to calculate an error probability EP_i for o_i. In each iteration, if we select m objects and the size of ranking question is y, there will be totally $m + \binom{m}{y}$ questions and the budget is b. So we can calculate

[2] For the convenience of computation of KL distance between D_G and D_i. We use a small value of $1e-6$ to replace 0 for the values greater than k in D_G.

Algorithm 2: QuestionSelection

Input: Objects O; Budget in each round b; Ranking answer $\mathcal{P}$; Rating answer $\mathcal{R}$; Worker Quality $\mathcal{E}$
Output: Selected questions Q

1 $EP = \phi$; // Error probability for all of the objects
2 **for** $i = 1$ *to* n **do**
3 $\mathcal{I} = \phi$;
4 **for** j *in* $\mathcal{W}_i$ **do**
5 $\mathcal{I} = \mathcal{I} \cup$ Interval($\mathcal{R}_j^i$,$\mathcal{E}_j^i$)
6 EP_i =ErrorProbability($\mathcal{I}$);
7 Compute m by $m + \binom{m}{y} = b$;
8 Q=Top(EP,m);
9 Return Q;

m while b is known in the line 7 in Algorithm 2. Then we select objects with top-m error probability and generate questions.

Next we focus on how to calculate the error probability EP_i for object o_i. For an object o_i, its rating results follow the Gaussian distribution. The mean μ_i of the distribution is the real rating answer of o_i. We assume the variance σ of each of the distribution of o_i is stable which is predefined. However, the results from different workers may have different bias. As we have the accuracy $\mathcal{E}_j^i$ for j-th worker who answers o_i, the rating answer of o_i answered by this worker is $\mathcal{R}_j^i$. We have known that:

$$Pr(\mathcal{R}_j^i | r_i = \mu_i) = \Phi(\mu_i, \sigma, l) = \int_{\mu_i - l}^{\mu_i + l} \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{(x-\mu_i)^2}{2\sigma^2}] = \mathcal{E}_j^i \quad (17)$$

$$s.t.\ \mu_i - l \le \mathcal{R}_j^i \le \mu_i + l.$$

For j-th worker who answer o_i, we have a high confidence that the value of μ_i will be in the interval $[\mu_{lb}^{i,j}, \mu_{ub}^{i,j}]$ where $\mu_{lb}^{i,j}$ is the lower bound for μ_i and $\mu_{ub}^{i,j}$ is an upper bound for μ_i as shown in Figure 5. For each worker w_j who answers rating answer of o_i, we can calculate such $[\mu_{lb}^{i,j}, \mu_{ub}^{i,j}]$. Then we get a lower bound μ_{lb}^i by considering all the minimal values, i.e. $\mu_{lb}^i = \min_{j=1}^{|\mathcal{W}_i|} \mu_{lb}^{i,j}$ and an upper bound $\mu_{ub}^i = \max_{j=1}^{|\mathcal{W}_i|} \mu_{ub}^{i,j}$ according to the results from $\mathcal{W}_i$ where $\mathcal{W}_i$ is the set of workers who answer rating question of o_i. In line 6 of Algorithm 2, we calculate an error probability for o_i. In the top-k result $\mathcal{R}$, let μ_{k-th} denote the mean of the k-th object. Then the interval $\mathcal{I}$ for o_i has two parts. $[\mu_{lb}^i, \mu_{k-th}]$ is outside the top-k in $\mathcal{R}$ and $[\mu_{k-th}, \mu_{ub}^i]$ is inside the top-k in $\mathcal{R}$ as shown in Figure 6. Then we calculate the error probability EP_i for o_i as:

$$EP_i = \begin{cases} \frac{\mu_{k-th} - \mu_{lb}^i}{\mu_{ub}^i - \mu_{lb}^i} & o_i \text{ is in the top-}k \text{ result} \\ \frac{\mu_{ub}^i - \mu_{k-th}}{\mu_{ub}^i - \mu_{lb}^i} & o_i \text{ is not in the top-}k \text{ result} \end{cases} \quad (18)$$

For example, o_2 is answered by workers w_1, w_2 and w_3 so $\mathcal{W}_2 = \{w_1, w_2, w_3\}$ and $\mathcal{R}_1^2$ = 2002, $\mathcal{R}_2^2$= 1997, $\mathcal{R}_3^2$ = 2005. It is easy to calculate that μ_2=2001.33. The variance is σ = 3.3 as calculated in Section 2.2. We also know that $\mathcal{E}_1^2$ = 0.61, $\mathcal{E}_2^2$ = 0.28, $\mathcal{E}_3^2$ = 0.30. We calculate interval for w_1 that Interval($\mathcal{R}_1^2$,$\mathcal{E}_1^2$) = $[\mu_{lb}^{2,1}, \mu_{ub}^{2,1}]$ = [1999.16, 2004.83]. We can also calculate $[\mu_{lb}^{2,2}, \mu_{ub}^{2,2}]$ = [1995.82, 1998.18] and $[\mu_{lb}^{2,3}, \mu_{ub}^{2,3}]$ = [2003.73, 2006.27]. Then we have $[\mu_{lb}^2, \mu_{ub}^2]$ = [1995.82, 2006.27]. If we sort the 6 objects according to current rates and select the top-3. The value of the third object is 2003.7. o_2 is not in the top-3. However, the possible range for o_2 is [1995.82, 2006.27]. So we may misplace o_2 with an error probability of $\frac{2006.27-2003.7}{2006.27-1995.82}$ = 0.246. Suppose we select m = 3 objects to generate new questions. We calculate error probability from o_1 to o_6 and select 3 objects with higher error probability.

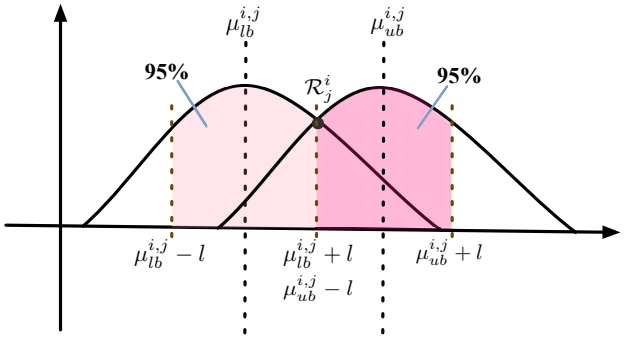


Figure 5: Interval for o_i of j-th worker.

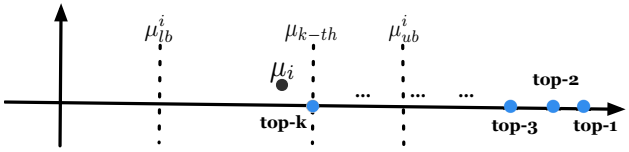


Figure 6: Error Probability for o_i.

Notice that we tune the parameters to get a proper m in our heuristic algorithm. When $m = y$, it is easy to utilize Algorithm 2 but we may lose many objects who are rated wrongly by workers. When the m is too big, it will be benefit for calculating the real answer but it will be expensive. In real application, we can always find a trade-off to get a proper m.

In each iteration, we will use the selection algorithm once. It will be verified by experiments that our heuristic selection algorithm has a higher potential to converge in high probability after each iteration. The summation of the error probabilities in EP for all the n objects will be smaller and smaller as we iterate to select the proper objects for generating questions.

Remark. When we consider all the n objects, the complexity of optimizing Equation 15 is $O(n^{2(y+1)})$. However, as we reduce n to a much smaller m above, the complexity of Equation 15 can be reduced to $O(m^{2(y+1)})$. As batch gradient descent iterates many times, we propose to use stochastic gradient descent to replace it which only considers a small sample of the whole dataset and uses the sample to iteratively compute the parameters. It converges more quickly than batch gradient descent. Thus, we can reduce the complexity of Equation 15 to $O(m^{(y+1)})$. In most of the case, it is very small as y is always less than 5 and m is always less than 10.

4.3 Extension to Assign Multiple Questions

Our selection algorithm is easy to adapt to assign multiple (e.g., b) questions to a worker. As we infer the top-k probability after getting the answers of these b assigned questions, the order of these b questions have no influence to the score μ and variance σ, because we will use these b questions together to update μ and σ. So we can see that whatever the order of questions, the likelihood function is the same. In this way, we still use our selection algorithm to iteratively select b questions.

5 EXPERIMENTS

5.1 Experimental Setting

Datasets. We used three real datasets to evaluate our method.

EventTime[3]**.** We collected 115 famous history events in modern history from Wikipedia and asked crowd to rank them by event time. Workers were asked to give the specific year for these events. For example, "Russian Revolution ends the Russian Empire" happened at 1917 and "Cuban Revolution" happened in 1959.

MovieTime. We collected 235 most famous movies from IMDB and asked the crowd to rank them by release time. We provided workers the title and poster of the movie, and workers were asked to give the release year for the movie. For example, "The Dark Knight" was released in 2008 and "Braveheart" was released in 1995.

IMDB-RANK. We asked the crowd to rank the above movies and took the ratings in IMDB as the ground truth. We aimed to check whether a small number of workers can recover the rankings in IMDB. We provided workers with the title of movies. In rating tasks, workers were asked to give a rating from 1 to 10. In ranking tasks, workers were asked to give a ranking of 5 given movies.

Questions. Each ranking question contained 5 objects. We collected 10 rating answers for each object and 5 ranking answers for each ranking question. The price of each rating question was \$0.01 and that of each ranking question was \$0.05. We conducted experiments on AMT.

Quality Metrics. We used the same evaluation metrics [44] to compare different algorithms. Assuming A^k was the top-k results returned by an algorithm, and T^k was the real top-k results. The first quality metric *Recall* was defined as $\frac{|A^k \cap T^k|}{k}$. The second metric was ACC^k that also measured the order for the top-k results.

$$ACC^k = \begin{cases} 1 & k = 1 \ \& \ A^k = T^k \\ 0 & k = 1 \ \& \ A^k \neq T^k \\ \frac{\sum_{o_i, o_j \in T^k \cap A^k} \mathtt{I}(i,j)}{k(k-1)/2} & k \geq 2 \end{cases} \quad (19)$$

where t_i was the real ranking position for object o_i, and a_i was the ranking position returned by an algorithm. $\mathtt{I}(i,j) = 1$ if $A_i^k < A_j^k \wedge T_i^k < T_j^k$; otherwise $\mathtt{I}(i,j) = 0$.

Setting. We implemented all the algorithms on a Macbook pro with 2.5 GHz Intel Core i7 CPU and 16GB RAM.

5.2 Evaluation of Inference Methods

Competitors. We compared with six existing methods.

SPR [18] used statistical tools to improve the quality by confidence level. Based on the pairwise comparisons, it attempted to minimize the total monetary cost of the top-k processing within a Select-Partition-Rank framework.

CrowdGauss [31] used pairwise comparisons as questions. It first computed a score for each object using maximum likelihood estimation (MLE) and Thurstone model [35]. Then it computed top-k results based on the scores.

CrowdBT [5] also utilized pairwise questions. It considered worker quality and computed the score based on BTL model [2].

Combine [43] used both ratings and pairwise comparisons as questions. Rating answers were modeled by Gaussian model, comparison answers were modeled by Thurstone model, and MLE was utilized to compute the score for each object.

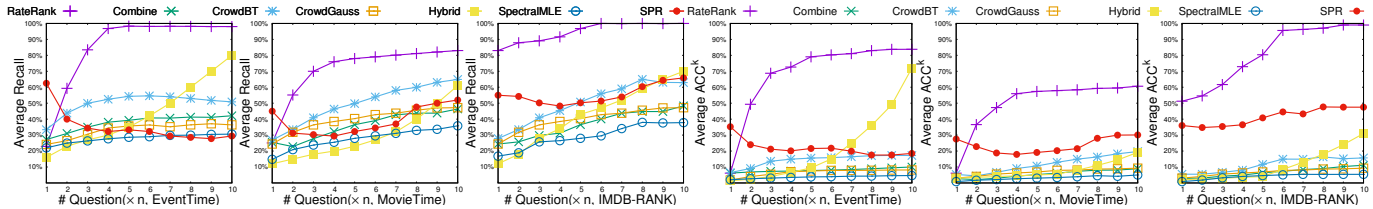
[3]https://en.wikipedia.org/wiki/Timeline_of_modern_history

Hybrid [17] also used ratings and pairwise comparisons. It first used rating answers to filter objects, and then used pairwise comparisons to further compare the objects. Then based on all the rating and comparison answers, a modified PageRank was utilized by Hybrid to compute the score.

SpectralMLE [6] contained two stages. The first stage initialized the score for each object by RankCentrality [27]. The second step refined the scores by using MLE.

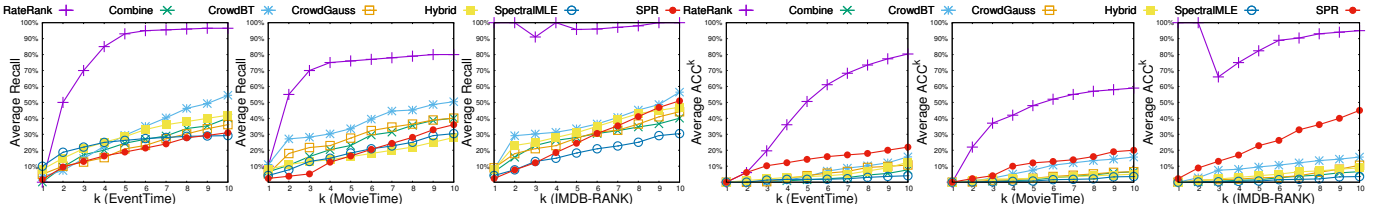
5.2.1 Real Experiments – Varying #Questions. Existing studies used either rating or comparison questions. To make a fair comparison, we set the rating and comparison question as a unit and the ranking question took 5 units. For example, given a budget of 5000 questions, we could ask either 5000 rating question, 5000 comparison questions, 1000 ranking questions, or 1000 rating questions and 800 ranking questions. We varied the question numbers and Figure 7(a) shows the *Recall* results.

We had the following observations. Firstly, RateRank significantly outperformed other inference algorithms, because RateRank used rating answers to remove many objects not in the top-k results and used ranking answers to refine the results around top-k results. Secondly, CrowdBT [5] was better than other existing competitors because it considered both the structure of the graph composed by the pairwise comparisons and the workers' quality. Thirdly, Combine [43] and CrowdGauss [31] were machine learning methods, which used Gaussian model to model the noise in the pairwise comparisons. The only difference was that Combine also utilized the rating answers. However, in MovieTime dataset, the average accuracy of rating answers was low, and the rating answers with bad quality would affect the performance of Combine. So the *Recall* of Combine was worse than CrowdGauss in MovieTime dataset. Fourthly, SpectralMLE [6] also had two stages. The first stage was an initialization to the score of the objects by RankCentrality [27], and the second stage refined the initial scores by the maximum likelihood estimation. However, we found that the performance of SpectralMLE was worse than the other methods on both datasets. The reason was that we had a limited pairwise comparisons, so in the initialization of RankCentrality, the transfer matrix was rather sparse. In this situation, the initial score for each object was not accurate. Even with the second stage, the final result was still worse than the other methods. Fifthly, for Hybrid method, initially, the performance was worse than other methods. This was because rating answers had many noises, since Hybrid algorithm completely depended on the rating answers in the first stage, and it had no any quality control method for rating answers. So the *Recall* of Hybrid was worse than other algorithms. However, with the increase of budget, we found that Hybrid had a rapid improvement of *Recall* in the EventTime dataset. This was because with redundant rating answers, Hybrid could gradually recover the real value of objects in EventTime dataset. Then Hybrid could select objects with the smallest values to participate in the second stage to find top-k objects.However, this algorithm did not have any quality control methods, so in MovieTime and IMDB-RANK dataset, even with the increase of budget, the improvement of Hybrid was slower than in EventTime dataset. Sixthly, most of methods performed better in the IMDB-RANK and EventTime datasets than the MovieTime dataset, because the MovieTime tasks were harder than EventTime



(a) *Recall* (b) ACC^k

Figure 7: Inference Performance by Varying #Questions ($k = 10$)



(a) *Recall* (b) ACC^k

Figure 8: Inference Performance by Varying k ($6n$ Questions).

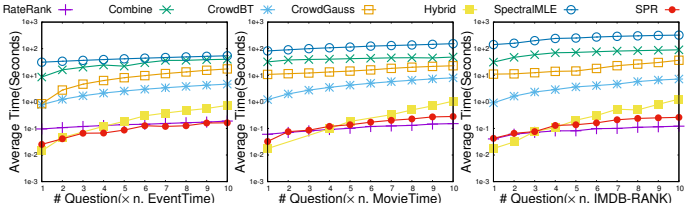


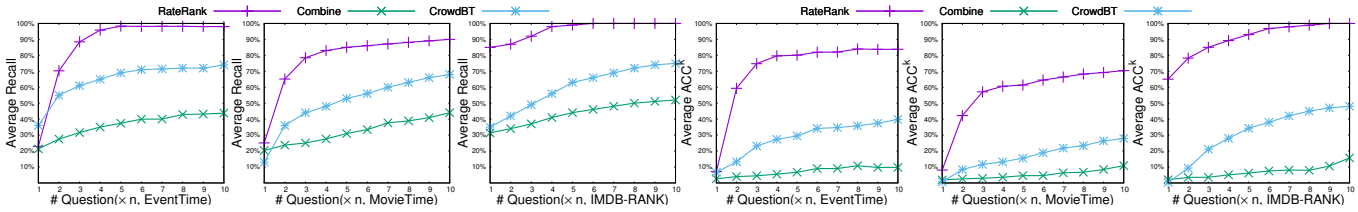
Figure 9: Efficiency for Inference Methods ($k = 10$)

and IMDB-RANK tasks. For example, the well-known top-k movies were so popular that workers always gave high ratings and ranking; history event time was closely related with daily life so that workers could give the ratings and ranking accurately; however workers might not know the release years of movies exactly.

Figure 7(b) presented the ACC^k result for all seven inference methods. ACC^k reflected the order of the top-k results. The results were consistent with those for *Recall* and RateRank was much better than other methods, leading by 20-60%. The reasons were three-fold. (1) Our framework considered the confidence for each rating and ranking answers. The computation of confidence for each answer not only took advantage of worker's quality, but also utilized the distribution of the rating and ranking answers. So the quality was guaranteed for the input answers. Thus we found that even in MovieTime dataset that the average rating and ranking accuracy was low, RateRank still had a good performance. (2) Our model successfully combined rating and ranking answers. Rating answers were used to get a rough score and ranking answers refined the scores. As RateRank had already had a quality control for all the rating and ranking answers. And RateRank also had a model to combine rating and ranking answers. (3) Our method could effectively compute the overall top-k probability for each object. Compared with CrowdBT, although CrowdBT and RateRank had quality control method, RateRank used rating answers and greatly saved cost in top-k computation. Compared with Hybrid, RateRank proposed a quality control method to guarantee the accuracy of rating and ranking answers, thus avoided the impact of the wrong answers. Besides, we also found that RateRank performed better on ACC^k in the IMDB-RANK dataset than the EventTime and MovieTime datasets, because the ratings of the top-k candidates in IMDB-RANK were not close and we had more opportunity to use some ranking questions to refine the top-k answers.

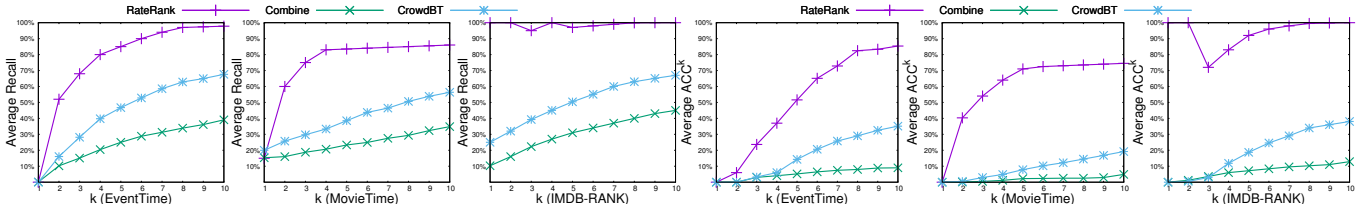
5.2.2 Real Experiments – Varying k. We varied k and evaluated the quality. Figure 8(a) and 8(b) showed the results on *Recall* and ACC^k respectively. The number of asked questions was $6n$ units. We had the following observation. Firstly, RateRank still outperformed other inference algorithms due to our ranking-rating framework, which could utilize different questions to identity top-k results based on different granularities. Besides, we found that even with a small k, e.g., 4, the *Recall* for RateRank was higher than 80% in EventTime dataset, and the *Recall* for RateRank was higher than 70% in MovieTime dataset. For other inference algorithms, CrowdBT still had an advantage in the three datasets, because it could estimate the scores more accurately. Secondly, with the increase of k, the average *Recall* and ACC^k also gradually increased for different algorithms in the EventTime and MovieTime datasets. This was because with a larger value of k, the top-k problem became easier. For example it was easier to find top-100 results from 1000 objects than finding top-10 results. Our method increased much faster than existing algorithms due to our hybrid ranking-rating model. Thirdly, RateRank could find the exact top-k answers in many cases in IMDB-RANK, because these movies were very famous and we could obtain higher ratings from workers. For example, most of the workers would give a rating more than 9 for "The Shawshank Redemption" which had significantly larger scores than other movies. Thus, we could exactly find the top-k movies. Based on high accuracy of ratings and ranking answers, RateRank had more than 90% accuracy on *Recall*. Due to similar reasons, RateRank performed much better on the IMDB-RANK dataset than the EventTime and MovieTime datasets in terms of ACC^k.

5.2.3 Efficiency For Inference Methods. We evaluated efficiency of different methods and Figure 9 showed the results. We had the following observations. Firstly, the machine-learning algorithms (CrowdGauss, CrowdBT, SpectralMLE, Combine) took more time than other methods Hybrid, SPR and RateRank, because the machine-learning algorithms required to spend much time to learn the parameters. SpectralMLE took most of the time in finding top-k objects, because SpectralMLE had two stages and the second stage took much time in optimizing the scores. Combine was slower than



(a) *Recall* (b) ACC^k

Figure 10: Selection Performance by Varying #Questions ($k = 10$).



(a) *Recall* (b) ACC^k

Figure 11: Selection Performance by Varying k ($6n$ Questions).

CrowdGauss, because Combine also needed to consider the rating answers. CrowdBT was better than Combine and CrowdGauss, because the optimization function of CrowdBT utilized the Bradley-Terry model [2] instead of the Thurstone model [35] that used by Combine and CrowdGauss. In the optimization of CrowdBT, Combine and CrowdGauss, the corresponding target function for them need to be evaluated many times. The efficiency in computing pairwise comparison results based on the Bradley-Terry model was faster than comparison results according to Thurstone model. Secondly, for RateRank, the time spent in computing the confidence for each answer was small. The main part of the time spent was in optimizing Equation 15. So we utilized stochastic gradient descent method in optimizing Equation 15. Thus the time spent for RateRank would be small in expectation. The experiments in Figure 9 validated our point of view for RateRank algorithm. Thirdly, Hybrid was faster than RateRank initially, because Hybrid only considered a smaller number of objects initially. And the time spent on Hybrid increased rapidly with the increase of the number of questions. Our method achieved higher efficiency on every datasets and could infer the results within 0.1 second, thus our method was rather efficient.

Remark. To summarize, our inference algorithms achieved much higher quality than existing approaches, even by 20-50%. Moreover, our method could achieve high efficiency and meet the performance requirement of fast result inference.

5.3 Evaluation on Selection Methods

Only CrowdBT, Combine and CrowdGauss studied the selection problems. As CrowdGauss involved many matrix computations, it was too slow. Thus we only compared with CrowdBT and Combine. CrowdBT [5] selected objects with the largest expected change of scores. Combine [43] selected questions with largest uncertainty.

5.3.1 Real Datasets – Varying #Questions. We varied the number of questions to compare the selection algorithms. Similar to the inference experiments in Section 5.2.1, the budget varied from n units to $10n$ units. We had the following observations. Firstly, RateRank had great advantage to CrowdBT and Combine, leading 20-30% on *Recall* and 20-50% on ACC^k. This was because RateRank could judiciously select the questions by minimize the difference between the estimated distribution and real distribution. Secondly, with the increase of number of questions, the quality increased because they could utilize more questions to infer the results. Thirdly, the performance of RateRank, CrowdBT and Combine in Figure 10(a) and 10(b) were all had improvements compared to random question selection in Figure 7(a) and 7(b). For CrowdBT, the selection algorithm in CrowdBT preferred to select pairwise comparison that would make the most change to the previous estimation of the object. Although CrowdBT mainly focused on the permutation of all the objects, the permutation of the top-k objects would still be refined. So we found that the *Recall* and ACC^k were all had improvements compared to randomly selecting questions. The selection algorithm in Combine preferred to select rating or pairwise comparisons that were most uncertain. This question selection method was useful in the permutation of all the objects. But it was inefficient to find the top-k objects. As soon as we had a knowledge of the objects, the objects that ranked in front usually had an obvious difference compared to the other objects. In this case, Combine would not select pairwise comparisons to further refine the permutation of the top-k objects. There was only a slightly improvement for Combine compared with randomly selecting questions in Figure 10(a) and 10(b). For our proposed selection method, we found that the *Recall* and ACC^k both had improvements on the EventTime and MovieTime datasets when there were few budgets. For example, when the budget was $2n$ or $3n$, there was an improvement of *Recall* about 10% in EventTime and MovieTime datasets. And an improvement of ACC^k about 5% in both datasets. Compared with randomly selecting questions, our proposed method preferred to select objects close to top-k results. Fourthly, RateRank had significant improvements on IMDB-RANK even there were few budgets for both *Recall* and ACC^k. For example, when the budgets was $3n$ on the IMDB-RANK dataset, the *Recall* of RateRank was more than 90% and the ACC^k was more than 80%. The reasons were two-folds. Firstly, the ratings and ranking from workers had high accuracy. Secondly, our selection techniques improved the performance of RateRank comparing with the inference only method. RateRank could nearly find the exact answer on the IMDB-RANK dataset by judiciously selecting the questions to ask.

5.3.2 Real Datasets – Varying k. We varied k from 1 to 10 and compared different algorithms. In this experiment, the budget was

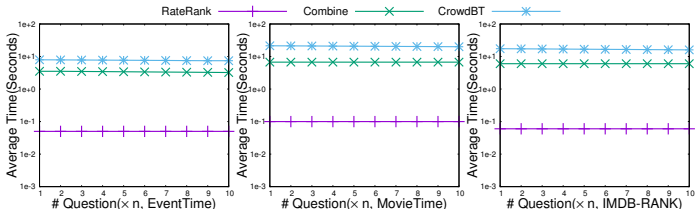


Figure 12: Efficiency for Selection ($k = 10$).

fixed to $6n$ units. Figures 11(a) and 11(b) showed the results. We found that RateRank still had great advantage compared to CrowdBT and Combine. With the increase of k, the *Recall* and ACC^k both increased. This was because the smaller of k, the harder of the top-k problem. With the increase of k, CrowdBT and Combine had a slow increase of the *Recall* and ACC^k. However, our proposed RateRank had a rapid increase of both *Recall* and ACC^k. The *Recall* for RateRank was stable when k increased to 4. The ACC^k was stable when k increased to 6. This validated that the performance of RateRank was stable even with a small value of k, because RateRank could judiciously select rating or ranking questions to get high-quality results by making a full use of the given budget.

5.3.3 Efficiency For Selection Methods. We evaluated the efficiency by varying #questions. Figure 12 showed the efficiency for RateRank, CrowdBT and Combine. We evaluated the average time for selecting one question. We found that CrowdBT took much more time than Combine and RateRank, because in selecting each pairwise comparison of CrowdBT, it had to enumerate all $\binom{n}{2}$ combinations of pairs and select one pair that could make the greatest change to the previous estimation of the objects. So the efficiency for CrowdBT selection algorithm was low. Combine had to solve an integer programming problem in selecting rating and comparison questions in each time. So the efficiency of Combine was also worse than RateRank. Our proposed heuristic selection algorithm was very efficient, because it could prune many unnecessary objects that could not be in the top-k results. Thus our method can be easily extended to support large datasets.

Remark. In all, our selection method outperformed state-of-the-art approaches by 10-40% in quality, and also was very efficient. Thus our method could meet the requirement of online task assignment to improve the quality effectively. We would evaluate the results by varying m in Appendix C.

6 RELATED WORK

Crowdsourcing has been extensively studied by the database community recently and Li et al. [21, 22] provide a detailed survey in crowdsourced data management.

Crowdsourced top-k Algorithm. We classify the crowdsourced top-k algorithms into five categories, local heuristic, global heuristic, heap-based method, machine-learning method, and hybrid method. (1) Based on the pairs, they construct a directed graph, where a vertex is an object and a directed edge is the comparison result, and utilize local information of this graph to compute a score for each object. In other words, they only consider the neighbors of each object. For example, BordaCount [1] count the number of winnings for each object. Copeland [32] is similar to BordaCount. The only difference is that Copeland also consider the number of losing times. So Copeland count the total winning times minus the total losing times as the final score for each object. BRE [32] count the number of total losing times minus the total winning time. URE [32] count the number of total losing times. (2) SSCO [3] and SSSE [3] gradually select objects with high probability in the top-k set, and prune objects with large probability not in the top-k set. PathRank [8] utilize the global information of the graph. It conduct "reverse" depth first search (DFS) for each object. If there exists a path longer than k, it means there are at least k objects better than this object, and thus this object will not belong to the top-k set. AdaptiveReduce [8] firstly selects a set of informative objects, and then uses this set of objects to remove objects with small probability in the top-k set. (3) Heap-based method [7] divides the objects into many small heaps, and finds the largest object in each heap to find the final top-k objects[11]. (4) CrowdGauss [31], CrowdBT [5] and SpectralMLE [6] utilize machine-learning-based techniques to learn a score for each object. (5) Combine [43] and Hybrid [17] use hybrid rating and pairwise comparison questions to find top-k results.

Crowdsourced Sort Algorithm. RankCentrality [27] is designed to sort all the objects based on random walk. It first constructs the transfer matrix and then repeatedly computes the distribution for the objects until converge. CrowdBT [5] and CrowdGauss [31] focus on sorting but can be used for top-k computation.

Crowdsourced Max Algorithm. Guo et al. [16] propose the maximal object finding problem in crowdsourcing. This work focuses on max object inference and selection algorithms. All the algorithms are based on pairwise comparisons. Both inference and selection algorithms utilize the structure of the graph. Venetis et al. [37] also focus on the crowdsourced max problem. It proposes two inference algorithms, bubble sort and tournament max algorithm. [17] is a two-stage algorithm to find the maximal object. The first stage is based on ratings. Objects with high rating results will participate in the second pairwise comparison stage.

Crowd-Powered System and Operators. There are several crowd-powered database systems, such as CrowdDB [14], Qurk [25], Deco [29], CrowdOP [10], and CDB [19, 20]. Moreover, there are also techniques that focus on designing individual crowdsourced operators, including selection [28, 33, 41, 42], join [4, 38, 39, 48], top-k/sort [7, 24], aggregation [23], and collect [30, 36].

Quality-Control in Crowdsourcing. There are some quality-control methods to improve the quality [9, 12, 34, 45–47], which design worker model and task model to improve the result quality.

7 CONCLUSION

We studied the crowdsourced top-k problem. We proposed a rating-ranking-based framework that used rating questions and ranking questions to compute top-k results. We devised effective model to compute confidence for rating questions and ranking questions. We designed a unified model to combine rating answers and ranking answers to infer the top-k results. We proposed an effective question selection framework to judiciously assign questions to workers. Experimental results on real datasets showed that our method significantly outperform existing approaches.

Acknowledgement. Guoliang Li was supported by the 973 Program of China (2015CB358700), NSF of China (61632016,61472198, 61521002,61661166012), and TAL education.

REFERENCES

[1] R. M. Adelsman and A. B. Whinston. Sophisticated voting with information for two voting functions. *Journal of Economic Theory*, 15(1):145–159, 1977.
[2] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, pages 324–345, 1952.
[3] R. Busa-Fekete, B. Szorenyi, W. Cheng, P. Weng, and E. Hullermeier. Top-k selection based on adaptive sampling of noisy preferences. In *ICML*, pages 1094–1102, 2013.
[4] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD*, pages 969–984, 2016.
[5] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM*, pages 193–202, 2013.
[6] Y. Chen and C. Suh. Spectral MLE: top-k rank aggregation from pairwise comparisons. In *ICML*, pages 371–380, 2015.
[7] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.
[8] B. Eriksson. Learning to top-k search using pairwise comparisons. In *AISTATS*, pages 265–273, 2013.
[9] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.
[10] J. Fan, M. Zhang, S. Kok, M. Lu, and B. C. Ooi. Crowdop: Query optimization for declarative crowdsourcing systems. *IEEE Trans. Knowl. Data Eng.*, 27(8):2078–2092, 2015.
[11] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. pages 1001–1018, 1994.
[12] J. Feng, G. Li, H. Wang, and J. Feng. Incremental quality inference in crowdsourcing. In *DASFAA*, pages 453–467, 2014.
[13] R. Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.
[14] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.
[15] J. Guiver and E. Snelson. Bayesian inference for plackett-luce ranking models. In *ICML*, pages 377–384, 2009.
[16] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, pages 385–396, 2012.
[17] A. R. Khan and H. Garcia-Molina. Hybrid strategies for finding the max with the crowd: Technical report. Technical report, Stanford University, February 2014.
[18] N. M. Kou, Y. Li, H. Wang, L. H. U, and Z. Gong. Crowdsourced top-k queries by confidence-aware pairwise judgments. In *SIGMOD*, pages 1415–1430, 2017.
[19] G. Li. Human-in-the-loop data integration. *PVLDB*, 10(12):2006–2017, 2017.
[20] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: optimizing queries with crowd-based selections and joins. In *SIGMOD*, pages 1463–1478, 2017.
[21] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. Knowl. Data Eng.*, 28(9):2296–2319, 2016.
[22] G. Li, Y. Zheng, J. Fan, J. Wang, and R. Cheng. Crowdsourced data management: Overview and challenges. In *SIGMOD*, pages 1711–1716, 2017.
[23] A. Marcus, D. R. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.
[24] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.
[25] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.
[26] J. I. Marden. *Analyzing and modeling rank data*. CRC Press, 1996.
[27] S. Negahban, S. Oh, and D. Shah. Iterative ranking from pair-wise comparisons. In *NIPS*, pages 2483–2491, 2012.
[28] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012.
[29] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5(12):1990–1993, 2012.
[30] H. Park and J. Widom. Crowdfill: collecting structured data from the crowd. In *SIGMOD*, pages 577–588, 2014.
[31] T. Pfeiffer, X. A. Gao, Y. Chen, A. Mao, and D. G. Rand. Adaptive polling for information aggregation. In *AAAI*, 2012.
[32] J.-C. Pomerol and S. Barba-Romero. *Multicriterion decision in management: principles and practice*, volume 25. Springer, 2000.
[33] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and A. Y. Halevy. Crowd-powered find algorithms. In *ICDE*, pages 964–975, 2014.
[34] C. Shan, N. Mamoulis, G. Li, R. Cheng, Z. Huang, and Y. Zheng. T-crowd: Effective crowdsourcing for tabular data. *ICDE*, abs/1708.02125, 2018.
[35] L. L. Thurstone. The method of paired comparisons for social values. *The Journal of Abnormal and Social Psychology*, 21(4):384, 1927.
[36] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, pages 673–684, 2013.
[37] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, 2012.

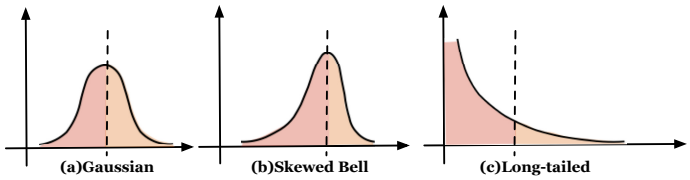


Figure 13: Three Distributions of Ratings.

[38] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.
[39] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.
[40] F. L. Wauthier, M. I. Jordan, and N. Jojic. Efficient ranking from pairwise comparisons. In *ICML*, pages 109–117, 2013.
[41] X. Weng, G. Li, H. Hu, and J. Feng. Crowdsourced selection on multi-attribute data. In *CIKM*, pages 307–316, 2017.
[42] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *MobiSys*.
[43] P. Ye, U. EDU, and D. Doermann. Combining preference and absolute judgements in a crowd-sourced setting. In *ICML '13 Workshop*, 2013.
[44] X. Zhang, G. Li, and J. Feng. Crowdsourced top-k algorithms: An experimental evaluation. *PVLDB*, 9(8):612–623, 2016.
[45] Y. Zheng, G. Li, and R. Cheng. DOCS: domain-aware crowdsourcing system. *PVLDB*, 10(4):361–372, 2016.
[46] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng. Truth inference in crowdsourcing: Is the problem solved? *PVLDB*, 10(5):541–552, 2017.
[47] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. QASCA: A quality-aware task assignment system for crowdsourcing applications. In *SIGMOD*, pages 1031–1046, 2015.
[48] Y. Zhuang, G. Li, Z. Zhong, and J. Feng. Hike: A hybrid human-machine method for entity alignment in large-scale knowledge bases. In *CIKM*, pages 1917–1926, 2017.

A DISCUSSION ON DATA DISTRIBUTION

The ratings from workers may follow different data distributions as shown in Figure 13. Besides the Gaussian distribution, we take the following two distributions as examples to show how our techniques can be extended to support them.

Note that the techniques of computing accuracy for each worker in Section 2.2, computing confidence for ranking results in Section 3.1.2, combing rating and ranking in Section 3.2 and selecting questions in Section 4 are orthogonal to the data distributions, and these techniques can be utilized in any data distributions. So we only need to consider the case of computing confidence for rating results when the distribution changes, i.e., changing Equations 5, 7 and 8 to support new data distributions.

Skewed Bell Curve. In some crowdsourcing tasks, workers may have a positive skewness or negative skewness on a specific rating, so the distribution may follow a skewed Bell Curve. Note that the Gaussian distribution is also a Bell Curve distribution with no skewness. For example, in the ratings of a famous movie (e.g., "The Shawshank Redemption") in IMDB, most of workers provide higher scores (e.g., ratings close to 9) while few workers give low ratings. The ratings of such objects will follow a positive skewed Bell Curve as shown in Figure 13(b).

There are many typical skewed Bell Curves such as Gamma distribution and Logarithmic Normal distribution. We can use statistical tool to estimate parameters of this type of function. We take Logarithmic Normal distribution as an example. We can use maximum-likelihood function to estimate the parameters.

The probability density function of Logarithmic Normal distribution is defined as

$$f(x,\mu,\sigma) = \frac{exp[-\frac{(\ln x-\mu)^2}{2\sigma^2}]}{\sqrt{2\pi}\sigma x},$$

where $x > 0$. Then the mean and variance in Section 2.3.1 can be replaced by

$$\mu_i = \frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\ln r_{ij}$$

and

$$\sigma_i = \sqrt{\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}(\ln r_{ij}-\mu_i)^2}.$$

Then, we replace Equation 5 with

$$e_{ij}^r = f(r_{ij},\mu_i,\sigma_i) = \frac{exp[-\frac{(\ln r_{ij}-\mu_i)^2}{2\sigma_i^2}]}{\sqrt{2\pi}\sigma_i r_{ij}}, \tag{20}$$

and replace Equations 7 and 8 with

$$\mu_i = \frac{\sum_{j=1}^{|\mathcal{R}_i|}\lambda_{r_{ij}}\ln r_{ij}}{\sum_{j=1}^{|\mathcal{R}_i|}\lambda_{r_{ij}}} \tag{21}$$

$$\sigma_i^2 = \frac{\sum_{j=1}^{|\mathcal{R}_i|}\lambda_{r_{ij}}(\ln r_{ij}-\mu_i)^2}{\sum_{j=1}^{|\mathcal{R}_i|}\lambda_{r_{ij}}} \tag{22}$$

Thus we can easily extend our framework to support Logarithmic Normal distribution.

Long-tailed Distribution. In some crowdsourcing tasks, most of the ratings fall into a small interval and few ratings fall into a extremely large interval. For example, considering the ratings of popular products (e.g., Apple Macbook Pro) in shopping website (e.g., Amazon), most people give high ratings while few people give low ratings. The ratings of such object will follow a long-tailed distribution as shown in Figure 13(c). There are many long-tailed distributions such as Zipf distribution and Pareto distribution. We use a typical Zipf distribution as an example where the probability density function is defined as below:

$$y = c_i x^{-b_i} \text{ (i.e., } \ln y = \ln c_i - b_i \ln x).$$

We can use answers in $\mathcal{R}_i$ to compute a proper curve for object o_i. For example, if we simply use "Least squares", we can find an approximate answer of c_i and b_i in linear time. Then we can compute the two parameters using the following equations

$$\begin{cases} b_i = -\frac{\sum_{j=1}^{|\mathcal{R}_i|}\ln r_{ij}\ln Pr(r_{ij}) - \frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\ln r_{ij}\sum_{j=1}^{|\mathcal{R}_i|}\ln Pr(r_{ij})}{\sum_{j=1}^{|\mathcal{R}_i|}(\ln r_{ij})^2 - |\mathcal{R}_i|(\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\ln r_{ij})^2} \\ c_i = exp[\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\ln(Pr(r_{ij})) + b_i\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\ln r_{ij}] \end{cases}$$

where $Pr(r_{ij})$ is the probability of the occurrence of rating answer r_{ij} among all rating answers of o_i, i.e., $Pr(r_{ij}) = \frac{count(r_{ij})}{\sum_{r\in\mathcal{R}_i} count(r)}$ (where $count(r_{ij})$ is the number of occurrences of r_{ij} among all the rating answers of o_i). Then the rating score of object o_i is computed by $exp[\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\ln r_{ij}]$.

Next Equation 5 can be replaced by

$$e_{ij}^r = f(r_{ij};b_i,c_i) = c_i r_{ij}^{-b_i} \tag{23}$$

and Equations 7 and 8 can be replaced by:

$$b_i = -\frac{\sum_{j=1}^{|\mathcal{R}_i|}\lambda_{r_{ij}}^2\ln r_{ij}\ln Pr(r_{ij}) - \frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\lambda_{r_{ij}}\ln r_{ij}\sum_{j=1}^{|\mathcal{R}_i|}\lambda_{r_{ij}}\ln Pr(r_{ij})}{\sum_{j=1}^{|\mathcal{R}_i|}(\lambda_{r_{ij}}\ln r_{ij})^2 - |\mathcal{R}_i|(\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\lambda_{r_{ij}}\ln r_{ij})^2} \tag{24}$$

$$c_i = exp[\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\ln(Pr(r_{ij})) + b_i\frac{1}{|\mathcal{R}_i|}\sum_{j=1}^{|\mathcal{R}_i|}\ln(r_{ij})] \tag{25}$$

In this way, we can easily extend our framework to support long-tailed distribution.

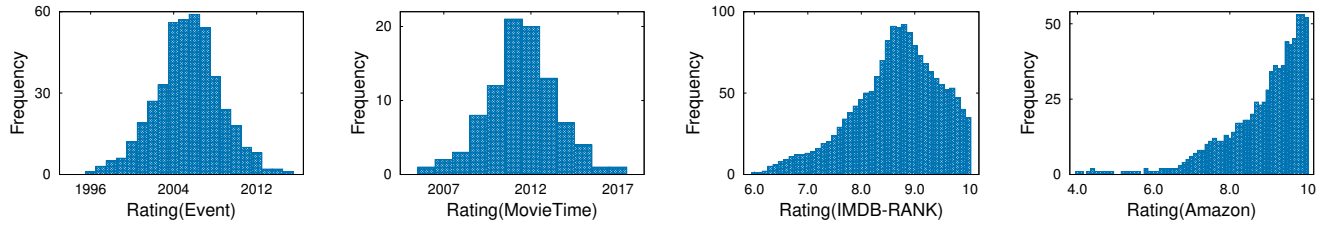
B EXPERIMENTS ON DATA DISTRIBUTIONS

Data Distribution of Rating Results. We collected the ratings of objects from our `EventTime`, `MovieTime` and `IMDB-RANK` datasets. Besides, we found that there were many ratings of products in shopping website, e.g., Amazon, so we collected the ratings of the well-known "Apple Macbook Pro" from crowdsourcing platforms. We showed the distribution of the ratings in Figures 14(a), 14(b), 14(c) and 14(d). We had the following observations. First, the ratings of `EventTime` and `MovieTime` followed Gaussian distribution well because these tasks are objective and workers have no skewed bias on the ratings. Second, the distribution of `IMDB-RANK` followed a skewed Bell Curve distribution, because most of the workers would had a positive skewness on the judgement of a good movie. Third, the distribution of a well-known product on Amazon followed a long-tailed distribution, because most people would be more likely to give a high rating for a good product.

In summary, we found that Gaussian distribution was the most popular distribution of the ratings of objects. The ratings of some objects with high reputation will follow a Skewed Bell Curve distribution, and the ratings of some popular products on shopping websites follow a long-tailed distribution.

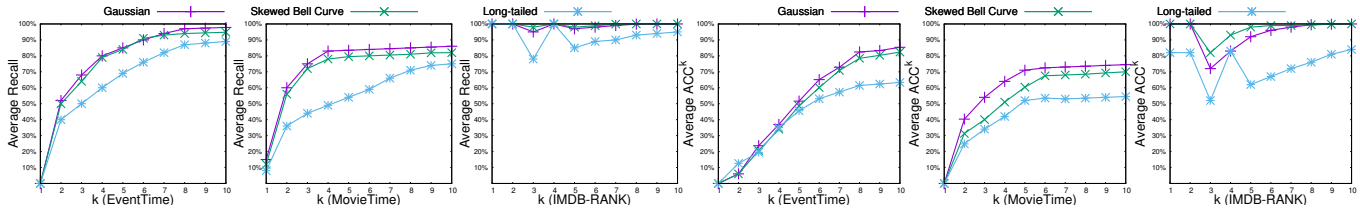
Comparison on Data Distribution of Rating Results. We evaluated different data distributions and showed whether different distributions affected the accuracy. We used the three data distributions on the three datasets and Figure 15 showed the results.

We had the the following observations. First, different distributions had minor effect on the crowdsourcing top-k results. This was because we could always compute the approximate ratings and ranking for the objects whatever the distribution was. In other words, our framework had robustness when the distribution of objects changed. Second, on the `EventTime` and `MovieTime` datasets, the Gaussian distribution and skewed Bell Curve distribution achieved similar results, which were better than long-tailed distribution on small k values. This was because long-tailed distribution might not adapt to other distributions. Third, we could use Gaussian distribution for most of datasets to aggregate the rating results, because it could adapt to most of data distributions and achieved high quality. Fourth, with increase of k, the accuracy improved as a larger k made the problem easier. For example, finding top-100 results was easier than finding top-10 results from 1000 objects.



(a) Event (b) Movie (c) IMDB (d) Amazon

Figure 14: Distribution of Objects on Real Datasets.



(a) *Recall* (b) ACC^k

Figure 15: Inference Performance by Varying k on Different Distributions($6n$ Questions).

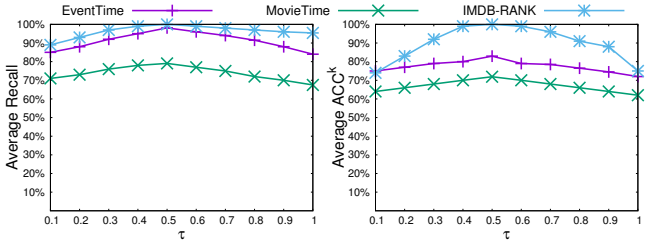


Figure 16: Influence of Parameter τ ($k = 10$)

C ADDITIONAL EXPERIMENTS

C.1 Evalutation Of τ

In this experiment, we evaluated the performance of RateRank inference method by varying the value of τ. The value of k was fixed to 10, and the budget was fixed to $6n$. We varied the value of τ from 0.1 to 1. Figure 16 showed the results. We found that the performance of RateRank was affected by τ. When τ was small, the value of μ and σ mainly depended on rating answers. Otherwise, when τ was large, the value of μ and σ mainly decided by ranking answers. With the increase of τ, the *Recall* and ACC^k gradually increased, and then slowly decreased. We found that when $\tau = 0.5$, RateRank achieved the best performance on both datasets.

In all, our inference method outperformed state-of-the-art approaches by 20-50% in quality, and also was very efficient.

C.2 Real Experiments – Varying Object Number

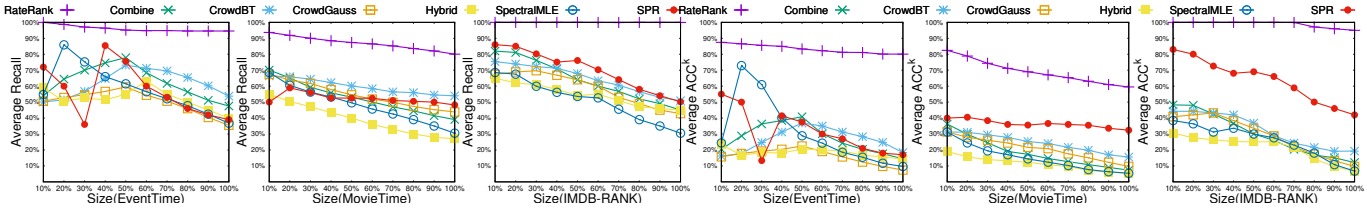
C.2.1 Inference – Varying Object Number. We varied the number of objects and compared different algorithms. Figures 17(a) and 17(b) showed the results on *Recall* and ACC^k. We still assigned $6n$ and set $k = 10$. We varied the size of the dataset from $10\% \cdot n$, $20\% \cdot n$, to $100\% \cdot n$. In each scale of the dataset, we randomly sample objects from all the objects. We had the following observations. Firstly, RateRank had obvious advantage to other algorithms on each dataset size, because our framework could utilize ranking and rating questions to improve the quality. Secondly, with the increase of the data size, the *Recall* and ACC^k both gradually decreased. This was because with a larger dataset, the top-k problem became harder, which was consistent with the results of increasing k while keeping the dataset size. With the increase of the data size, the speed of decreasing for RateRank was smaller than the other inference algorithms.

C.2.2 Selection – Varying Object Number. We varied the object number and compared the quality of different algorithms. Figures 18(a) and 18(b) showed the results. In this experiment, the budget was still fixed as $6n$ units, and k was fixed to 10. We varied the number of objects from $10\% \cdot n$, $20\% \cdot n$ to $100\% \cdot n$. From the results, we found that with the increase of the number of objects, the performance of *Recall* and ACC^k both decreased. It was obvious that with a larger dataset, the top-k problem naturally became harder. In addition, the decreased rates of RateRank was smaller than other two methods due to our effective selection techniques to consider the objects around the top-k results.

In the `IMDB-RANK` and `EventTime` datasets, RateRank could always perform much better than other methods in terms of both *Recall* and ACC^k. Besides, the *Recall* and ACC^k of RateRank in `IMDB-RANK` converged very quickly. In other words, even with a small number of judgements from crowdsourcing platform, our method could recover the ranking by millions of IMDB users, and thus our algorithm was rather effective.

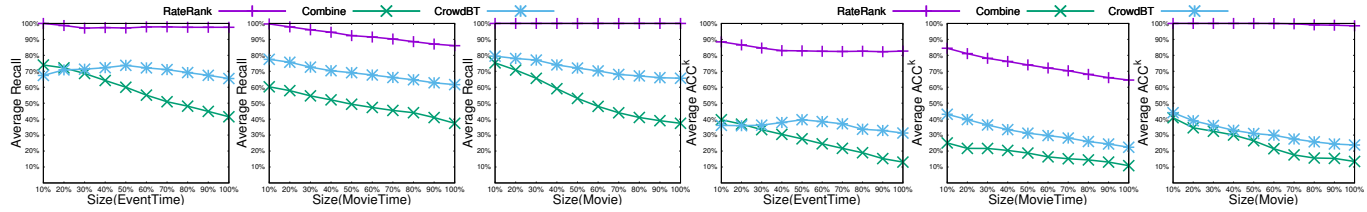
C.3 Evaluation of Tuning Selection Parameter

In Section 4.2, we proposed to use a small number m to replace n in order to improve the performance. In each iteration, we first chose m objects and then selected questions from these m objects rather than n objects. We compared two methods, n-Selection that selected questions from n objects and m-Selection that selected questions



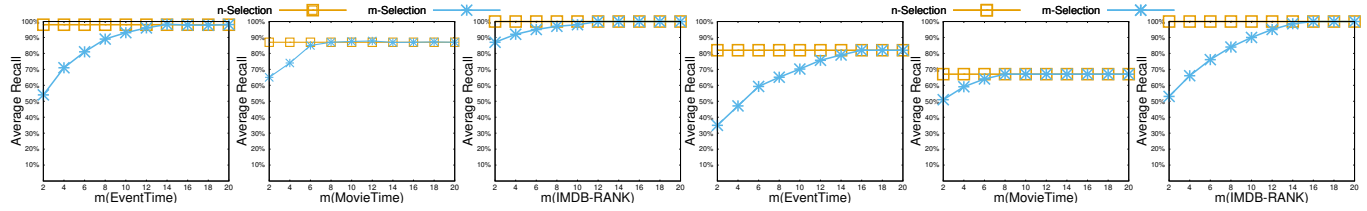
(a) *Recall* (b) ACC^k

Figure 17: Inference Performance by Varying Dataset Size($6n$ Questions).



(a) *Recall* (b) ACC^k

Figure 18: Selection Performance by Varying Dataset Size ($k = 10$, $6n$ Questions).



(a) *Recall* (b) ACC^k

Figure 19: Selection Performance by Tuning m ($k = 10$).

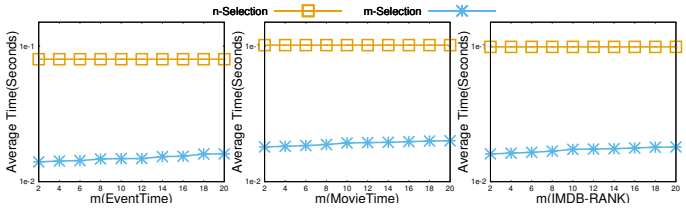


Figure 20: Selection Performance by Tuning m ($Time$, $k = 10$).

from m objects. The results on the *Recall*, ACC^k and efficiency were shown in Figures 19(a), 19(b) and Figure 20, where k was set to 10.

We had the following observations. First, m-Selection achieved similar results with n-Selection when $m \geq 10$. Thus we only needed to first choose 10 objects and used them to select questions. Second, m-Selection had much better performance than n-Selection because m-Selection used less objects to select the questions. Thus our method could significantly improve the performance while keeping high quality. Third, the average *Recall* of n-Selection kept stable as it selected questions from all objects while that of m-Selection first increased as the value of m increased because we could correct more misplaced objects by using a larger m. But after $m \geq 10$, the *Recall* kept stable, because all the high-quality questions could be selected from these m objects (other objects either had large probability in the top-k results or had small probability in the top-k results).