Asking the Right Questions in Crowd Data Sourcing

Rubi Boim [1] Ohad Greenshpan [1] Tova Milo [1] Slava Novgorodov [1] Neoklis Polyzotis [2] Wang-Chiew Tan [3,2]
[1]*Tel-Aviv University* [2]*University of California, Santa Cruz* [3]*IBM Research - Almaden*

***Abstract*—Crowd-based data sourcing is a new and powerful data procurement paradigm that engages Web users to collectively contribute information. In this work we target the problem of gathering data from the crowd in an economical and principled fashion. We present AskIt! , a system that allows interactive data sourcing applications to effectively determine which questions should be directed to which users for reducing the uncertainty about the collected data. AskIt! uses a set of novel algorithms for minimizing the number of probing (questions) required from the different users. We demonstrate the challenge and our solution in the context of a multiple-choice question game played by the ICDE'12 attendees, targeted to gather information on the conference's publications, authors and colleagues.**

I. Introduction

In this work, we target the problem of effective *crowd data sourcing*, namely gathering data from the crowd in an *economical* and *principled* fashion. Specifically, we present AskIt! , a system that allows interactive data sourcing applications to effectively determine which questions should be directed to which users in order to minimize the uncertainty about the collected data. This is essential in a variety of applications, as illustrated by the following three examples.

Consider an online marketplace where customers can rate the sellers from whom they purchased goods. (A concrete example is Amazon's service at http://www.amazon.com/gp/seller/sell-your-stuff.html .) Clearly, all parties can benefit from having representative ratings for every seller. But since dissatisfied customers are more likely to voluntarily provide their ratings, the obtained distribution of ratings for a given seller may be skewed, and hence there is a strong incentive to contact customers and explicitly request ratings for their recent transactions. In this example, the crowd comprises the customers, the data (and correspondingly the questions for customers) comprises the sellers' ratings, and the problem is to understand which customers to contact in order to maximize the information gain from the obtained user answers. As another example, consider an image-tagging service such as tagasauris (http://www.tagasauris.com/). In this scenario, the crowd comprises paid workers, the data (and corresponding questions) comprises the tags associated with images in a specific corpus, and the goal is to intelligently assign images to workers for tagging. The underlying optimization problem is similar to the above but in addition every request to the crowd involves a monetary reward. Hence, it may be desirable to select only a limited number of the most useful requests based on a total monetary budget. Finally, one more example is the GUESS system from IBM Research, which engages users in a Trivia-like game and uses their answers for Trivia questions to maintain and curate a knowledge base. The nature of the game introduces an interesting constraint: each user typically plays only for a limited time hence cannot be asked too many questions. This further complicates the optimization problem.

The above examples involve an optimization across several dimensions. First, not all customers are equal: in the sellers rating example, each customer can provide ratings solely for the sellers with whom there was a transaction; different customers may be more or less likely to provide ratings; and, there may be constraints on how often a customer can be contacted (e.g., a customer may be more likely to reply to a rating request if they are not contacted too often). Second, not all questions/answers (ratings in this example) are equal. As a simplified example, assume that each seller is rated as either "GOOD" or "BAD", and consider two sellers A and B with the following ratings: seller A has three "GOOD" ratings and one "BAD" rating, whereas seller B has two "GOOD" ratings and two "BAD" ratings. One more rating for seller B cannot shift the overall distribution much, whereas one more rating for A can make a larger difference (it will lead to either a 4/1 split, i.e., strong positive majority, or a 3/2 split, i.e., almost equal split of opinions). In this sense, an additional rating for A is more beneficial. Overall, these observations reveal a complex optimization problem that is not likely to be solved effectively by requesting ratings at random.

To address and solve the above issues, we developed AskIt! , a real time system that carefully selects which questions to ask in order to reduce the uncertainty of the current data and also minimize the number of probes (questions) at different users. The optimization problems handled by AskIt! address a wide range of real-world applications in our target domain (including the three examples mentioned above). It is intuitively formalized as follows. We consider a universe U of users (the crowd) and a universe Q of questions. The information that is gathered from the crowd is represented by a $|U| \times |Q|$ matrix M, where an entry $M[u, q]$ contains the answer of user u to question q or the special value $\emptyset$ if u has not answered q. For instance, in the seller ratings application, q represents a seller, u represents a customer and $M[u, q]$ is the rating given by u to q (or $\emptyset$ if u has not rated q). Hence, asking the crowd for data means resolving some of the $\emptyset$ cells to concrete values. We develop a metric (described in the following section) to characterize the quality of the information in M based on the uncertainty caused by $\emptyset$ cells. Uncertainty as we define it can only decrease as more $\emptyset$ cells are resolved, and the decrease depends on the choice of which cells are resolved. (This is in accordance to the example we presented earlier with the two sellers A and B.) Using this foundation, we can formalize the optimization problem as follows: Given M and a set of constraints on which $\emptyset$ cells can be selected, identify the cells that satisfy the constraints and provide the highest reduction in uncertainty.

Outline of the demonstration: We demonstrate the problem described above along with our solution in the context of a multiple-choice question game played by the ICDE'12 attendees. The game is used to collect qualitative information about the conference's publications and authors. We show how AskIt! chooses what questions to ask and to which person (player), such that the overall uncertainty about the answers' distribution (or, alternatively, the information gain) is minimized (maximized). To make the game more entertaining and whimsical (and thereby motivate users to participate in our demo), AskIt! will also ask questions about humorous characteristics of the conference attendees, such as which celebrity they resemble or how funny they are. Participating players will then be allowed to view the crowd's answers about themselves and their colleagues, but only after they themselves answer a minimal set of questions.

II. Technical Background

We start with a formal statement of the optimization problem solved by AskIt! and a brief description of the principles underlying the algorithms used to solve it, and then consider some practical aspects regarding the input data (users, questions and answers).

Problem statement and algorithms: We are given a set of users U and a set of questions Q. We define a $|U| \times |Q|$ matrix M that holds the answers of users to specific questions. Specifically, $M[u, q]$ takes any value in the set $A_q \cup \{\emptyset\}$, where $A_q = \{a_{q1}, \dots, a_{q|A_q|}\}$ is a set of possible answers for question q, and $\emptyset$ denotes an unknown answer, i.e., the fact that user u has not provided an answer for question q. For simplicity, we henceforth assume that $A_q = A = \{a_1, \dots, a_n\}$ for all $q \in Q$, i.e., all questions have answers in the same domain. We will use (u, q) and c interchangeably to denote a cell in the matrix.

We use a measure of entropy to quantify the uncertainty in the current answers of some question q. Formally, let $p_1, \dots, p_n$ denote the probability distribution over the current answers of q. The entropy for q is computed as $-\sum p_i \log p_i$. We are *not* interested in minimizing the entropy of q, given that entropy depends solely on the answers provided by the crowd and this is something that we cannot control. Instead, *we are interested in minimizing the uncertainty that we have in the entropy of q due to the existence of $\emptyset$ cells.* We define this formally next. We define t as a matrix-specific threshold that denotes the desired number of users to answer question q. For instance, if we use a majority vote to derive an overall answer for each q, then t may express the size of the quorum (e.g., majority out of at least 5). Let v denote the number of users who answered q. Let v_i denote the number of users among them who gave the answer $a_i \in A$, and w.l.o.g., assume that $v_1 \geq v_2 \geq \dots \geq v_n$. A set of t answers for q may be achieved by asking $t - v$ users to provide an answer. Depending on these answer, q may have different entropy. The following is easy to verify.

- The minimum entropy for q is achieved if the remaining entries are all resolved as answer a_1 (the current majority).
- In contrast, the maximum entropy is achieved if we allocate x_i hypothetical votes for answer a_i so that the constraint $v_j + x_j + 1 \geq v_{j-1} + x_{j-1}$ holds for $j \in [k, n]$ and k is minimized.

We define the uncertainty of q as the difference between these two extreme entropies: $\textit{uncertainty}(q) = \textit{maxEnt}(q) - \textit{minEnt}(q)$.

Example 2.1: Consider a question q such that $n = 3$ (number of possible answers), $t = 5$ (number of desired users to answer q), and the distribution of current answers is $v_1 = 2$, $v_2 = 1$, $v_3 = 1$, i.e., one more $\emptyset$ entry needs to be resolved. The minimum entropy $\textit{minEnt}(q)$ is achieved when the remaining entry goes to the majority answer a_1, and hence the probability distribution is $p_1 = 3/5$, $p_2 = 1/5$ and $p_3 = 1/5$. The maximum entropy is achieved when the remaining answer is assigned to a_2, in which case the distribution is $p_1 = 2/5$, $p_2 = 2/5$ and $p_3 = 1/5$. The uncertainty of q is equal to $\textit{uncertainty}(q) = 0.21$.

Assume now that the current votes are distributed as follows: $v_1 = 2$, $v_2 = 2$ and $v_3 = 0$. In this case, the uncertainty in q is equal to $\textit{uncertainty}(q) = 0.08$. The reason for this reduction is that the one remaining $\emptyset$ entry cannot change the distribution too much. However, in the first case, there can be either a clear majority for a_1 or a near-split vote.

Having defined an uncertainty metric for each question q, we introduce an aggregate metric for the uncertainty of M. We consider two cases:

$$\textit{uncertainty}_{\max}(M) = \max_{q \in Q} \textit{uncertainty}(q) \quad (1)$$

$$\textit{uncertainty}_{\text{sum}}(M) = \sum_{q \in Q} \textit{uncertainty}(q) \quad (2)$$

The first metric bounds the uncertainty of every question in the matrix, whereas the second measures the overall uncertainty of all questions. The choice of metric depends on the target application and our system supports both. We can now define formally the problem we want to solve.

Definition 2.2 (Problem Statement): Given a matrix M, a choice $X \in \{\max, \text{sum}\}$ and a set of constraints, identify a set C of matrix cells such that:

- $\forall c \in C : M[c] = \emptyset$ (only $\emptyset$ cells are selected).
- C satisfies the constraints.
- $\max_{M' \in M_C} \textit{uncertainty}_X(M')$ is minimized, where M_C contains all possible matrices that we can derive from M by resolving solely the cells in C.

We will define shortly the classes of constraints that we consider. It is important to stress that the problem definition involves a worst-case analysis: we want to identify the unknown cells, such that their resolution will minimize the overall uncertainty in the worst case, i.e., under any possible assignment of answers to cells. We resort to this approach since we do not make any assumptions about the distribution of answers or how the users behave.

We distinguish four classes of constraints. Intuitively, they capture parameters such as how many questions overall we wish to ask (e.g., in the case of budgetary constraints), which users can answer which questions (e.g., only users who bought items from a given seller can provide rating for her), how many questions a given user can be asked (e.g., when game players play only for a limited amount of time) and how many times a given question can be asked (e.g., when there are budgetary constraints for individual questions).

A The constraints are defined through a set S of unresolved matrix cells and a positive integer k. Set C satisfies the constraints iff $C \subseteq S$ and $|C| = k$, i.e., we wish to select exactly k unresolved cells.

B Similar to the above except that k now bounds the number of probes allocated for each user rather than the overall number of probes, i.e., $C \subseteq S$ and $|\{q : (u, q) \in C\}| \leq k$ for all $u \in U$,

C Similar to the previous case, except that k applies to the number of probes per question, i.e., $C \subseteq S$ and $|\{u : (u, q) \in C\}| \leq k$ for all $q \in Q$.

D This is a combination of B and C. We are given a set S and two positive integers k_1 and k_2, and we enforce $C \subseteq S$ and $|\{u : (u, q) \in C\}| \leq k_1$ for all $q \in Q$ and $|\{q : (u, q) \in C\}| \leq k_2$ for all $u \in U$.

We have developed a set of algorithms to solve the optimization problem, for the two uncertainty metrics and four types of constraints mentioned above. The efficient computation of candidate queries is important in our setting, in order to maintain short interaction times with the user. As it turns out, the complexity of the problem differs for the two uncertainty metrics. For $uncertainty_{\max}$ we were able to design a PTIME algorithm that operates in a greedy manner and employs at each step a Max-Flow algorithm (on a network that captures the users/queries constraints) to ensure satisfaction of the constraints. For $uncertainty_{\text{sum}}$ a PTIME solution (based on dynamic programming) is possible for constraint classes [A] and [C]. However, the problem becomes NP-complete for constraint classes [B] and [D] and we thus employ a greedy heuristic. We omit details due to space constraints, but we note that part of the demonstration will be focusing on the algorithms.

Enriching the initial data set: It is important to understand that a typical data set (matrix) in our setting is often rather sparse, since each user can typically answer only a small number of questions. For example, if we look at the Netflix data set [1] that contains movie ratings, the number of movies (questions) is more than 17,000 but the average ratings (answers) per user is only 200. Thus, the number of possible questions the user can theoretically be still asked is huge. To reduce sparsity, we preprocess the data with an enrichment phase that comprises two complementary actions:

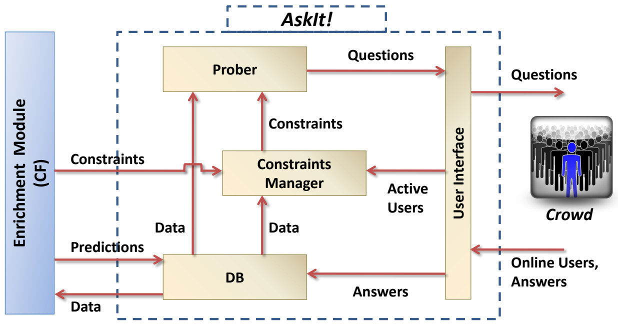

Fig. 1. AskIt! architecture

- *Predictions.* We apply a *prediction algorithm*, for each user, to predict her missing answers. Intuitively, the questions to which we can predict the answer with high certainty need not be asked as the predicted value may be used instead, i.e., inserted directly into the data set to enrich it (treated as "true" answers).
- *Pruning.* For the unknown entries for which we cannot predict the user answers, we apply an analogous prediction algorithm that estimates which questions the user is unlikely to answer (e.g., would press the *skip* button in a game). This information can be added as constraints.

In principle, any prediction algorithms can be applied for these two tasks. In our implementation, we employ Collaborative Filtering (CF) in both cases, as it has been shown to provide high-quality predictions in practice, especially for sparse data sets [2].

Related Work: Involvement of humans in generating data sets in a systematic manner has become popular in the past few years. A prominent example is Amazon's Mechanical Turk [3] service. Some related work [4] deals with the development of a unified language and model enabling data collection from both humans and machines while other [5] deals with ways to process this data [6], clean it and extract insights from it. Machine Learning techniques were also proposed as methods for identifying and pruning out low-quality teachers that are available for an online learning system that is trained on a data set and thus improve its quality [7]. Predictions generated by the CF method have been proven to improve recommendations for user actions [8].

III. System Overview

AskIt! is implemented in Java and PHP and uses a MySQL database. Figure 1 illustrates the system architecture, divided into operating modules. The *DB* includes the actual query answers given by users, and the generated predictions made by the *Enrichment Module*. The *Enrichment Module* employs the Collaborative Filtering algorithms from [8] to predict user answers and determine which users are likely to answer which questions. Note, however, that the independence between the module and the rest of the systems allows to plug-in any alternative prediction technique. Given the ids of the current active users and the input from the *Enrichment Module*, the *Constraints Manager* generates corresponding constraints for which questions can currently be posed. The *Prober* module uses these constraints and, depending on the chosen

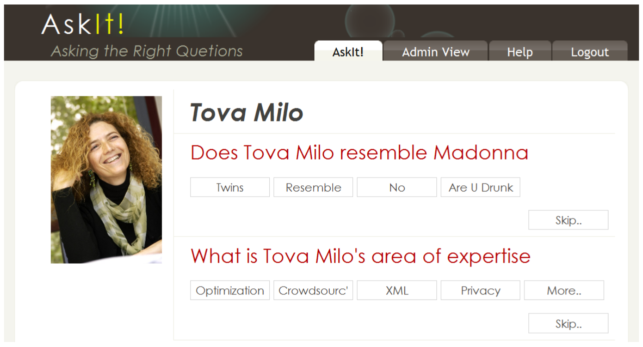

Fig. 2. AskIt! user interface

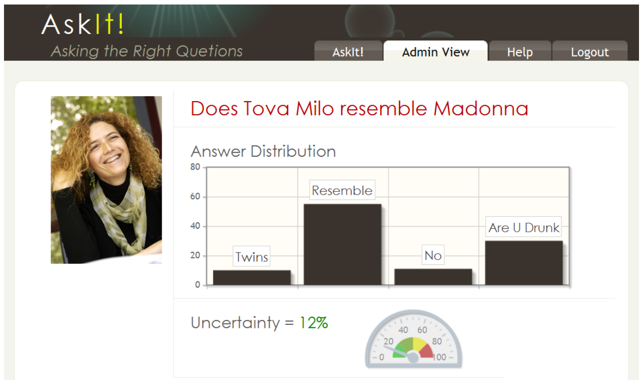

Fig. 3. The Administrator View: Data Set Snapshot

uncertainty matrix, employs the corresponding algorithm for determining which questions should be posed to each user to reduce the uncertainty about data distribution. In the demo itself (see next section) we use $uncertainty_{\max}$ with constraint class [B], but also explain how other settings similarly operate. The questions are posed to the users via the *User Interface* (UI). Figure 2 shows a screenshot of this operation. As mentioned earlier, in our demo scenario the collected data deals with ICDE'12 papers and authors, including both technical and humorous information. The user here is asked some multiple-choice questions about one of the participants. When relevant, AskIt! groups and presents related questions together, to extract more information in a single probe. We finally note that AskIt! may also be run in a "behind the scenes" mode which disables the UI and allows it to be invoked as a service. This mode is used by applications that wish to leverage its query selection facility but keep their own UI (e.g., in our sellers rating example, email the questions to users).

IV. Demonstration

In this demonstration, AskIt! will engage the ICDE'12 attendees to build a comprehensive crowdsourced database on researchers in the database community and their ICDE'12 publications. This will be done via a multiple-choice questions game, including both technical and humorous questions (the latter intending to attract as many participants as possible). The collected data consists, correspondingly, of technical and qualitative information about papers and authors as well as humorous information about the conference attendees. Examples for the first type include the subdomains to which each paper is most relevant to, whether it is more theoretical or systems-oriented, the interest that the paper stimulated in the conference, the presentation quality, the areas of expertise of the authors, etc. Examples for the second type include information about the celebrity to whom various researchers most resemble to, how serious or tall they are, etc. Each user plays only for a limited time hence cannot be asked too many questions. AskIt! is thus invoked to choose what questions should be posed to which of the current players, so that the overall uncertainty about the answers' distribution (or, alternatively, the information gain) is minimized (maximized). It is important to note that there is no single "correct" answer to questions and thus the goal is to approximate as much as possible the true distribution of answers, given the limited availability of users. To encourage participation, players will be allowed to view the distribution of the crowd's answers to questions (e.g., about their papers and colleagues), but only after they themselves answer a minimal set of questions per such view.

The data set used in the demo will be initialized with a set of questions by extracting the information from the ICDE 2012 program. The initial set of users will include all the paper authors. In order to not start the demo with an empty answers matrix we will partially fill it by fake (yet realistic) answers, using information extracted from DBLP.

We start the demonstration by explaining the game, its goal and rules. Then, we let our audience play it on several laptops allocated for that. In parallel, we will explain how AskIt! works (repeatedly, so that players who finished playing can join, allowing others to play). We will first show the current state of the database, i.e., the questions set, the answers provided so far and the level of uncertainty the system has about the actual "real" distribution of answers to these questions. We then request one of the attendees that played the game earlier to log-in again into the system (users log-in with their real name). We will first view her previous answers (if she agrees) then start the game again and follow its course. We will examine the questions that AskIt! poses to the user and her answers, and reveal, in parallel, on an Administrator screen, what is happening "under the hood". We will show the uncertainty level that the chosen questions have, the expected worst-case decrease of uncertainty and the actual decrease when the question is answered. We will explain the algorithms operation and why the specific questions were chosen. An example for one of the Administrator views that we will show is depicted in Figure 3, showing the current answers distribution of the question "Does Tova Milo resemble Madonna ?" as well as the uncertainty value. We will also describe how our system periodically updates its predictions, thereby improving its future selection of questions.

References

[1] J. Bennet and S. Lanning, "The netflix prize," *KDD Cup*, 2007.
[2] X. Su and T. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, 2009.
[3] "Amazon's mechanical turk," https://www.mturk.com/.
[4] A. G. Parameswaran and N. Polyzotis, "Answering queries using humans, algorithms and databases," in *CIDR*, 2011, pp. 160–166.
[5] D. Deutch, O. Greenshpan, B. Kostenko, and T. Milo, "Using markov chain monte carlo to play trivia," in *ICDE*, 2011, pp. 1308–1311.
[6] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "Crowddb: answering queries with crowdsourcing," in *SIGMOD*, 2011.
[7] O. Dekel and O. Shamir, "Vox populi: Collecting high-quality labels from a crowd," in *COLT*, 2009.
[8] R. Boim, H. Kaplan, T. Milo, and R. Rubinfeld, "Improved recommendations via (more) collaboration," *WebDB*, 2010.