Crowdsourcing for Multiple-Choice Question Answering

Bahadir Ismail Aydin, Yavuz Selim Yilmaz, Yaliang Li, Qi Li, Jing Gao and Murat Demirbas
Department of Computer Science and Engineering
University at Buffalo, SUNY, Buffalo, NY 14260 USA
Email: {bahadiri | yavuzsel | yaliangl | qli22 | jing | demirbas}@buffalo.edu

Abstract

We leverage crowd wisdom for multiple-choice question answering, and employ lightweight machine learning techniques to improve the aggregation accuracy of crowdsourced answers to these questions. In order to develop more effective aggregation methods and evaluate them empirically, we developed and deployed a crowdsourced system for playing the "Who wants to be a millionaire?" quiz show. Analyzing our data (which consist of more than 200,000 answers), we find that by just going with the most selected answer in the aggregation, we can answer over 90% of the questions correctly, but the success rate of this technique plunges to 60% for the later/harder questions in the quiz show. To improve the success rates of these later/harder questions, we investigate novel weighted aggregation schemes for aggregating the answers obtained from the crowd. By using weights optimized for reliability of participants (derived from the participants' confidence), we show that we can pull up the accuracy rate for the harder questions by 15%, and to overall 95% average accuracy. Our results provide a good case for the benefits of applying machine learning techniques for building more accurate crowdsourced question answering systems.

1 Introduction

Question answering (QA) is studied as a fundamental problem by the artificial intelligence (AI) and the machine learning (ML) communities. While search engines can answer well-formed factual queries successfully, their accuracy drops significantly for non-factual and natural language questions. There has been a substantial research effort on answering this kind of questions by incorporating the information retrieval (IR) and the natural language processing (NLP) techniques. There are some encouraging real-life examples (now ; sir ; Gunning et al. 2010; Ferrucci et al. 2010) of such systems, however, achieving acceptable accuracy for QA still remains a hard problem because both understanding and answering of the questions require solving complex AI problems.

In our work, we leverage human intelligence to help AI for more accurate QA. To this end, we consider crowdsourcing of QA and employ lightweight ML techniques, i.e.optimization with methods such as block coordinate descent. In order to simplify the problem further, we focus our attention to multiple-choice question answering (MCQA), because we notice that for aggregation purposes it is more productive to ask multiple choice questions than asking open-domain questions. In contrast to open-domain questions which result in a large set of possible answers, MCQA limits the answers to predefined set of multiple choices, and this facilitates participation of the crowd, improves the answer ratio and latency, and simplifies the aggregation process. We contend that providing multiple choice questions is feasible for many applications, and it is also possible to automate the process of adding multiple choices to an open-domain question (Lin, Mausam, and Weld 2012).

In order to develop more effective aggregation methods and evaluate them empirically, we developed and deployed a crowdsourced system for playing the "Who wants to be a millionaire?" (WWTBAM) quiz show. Our work was inspired by IBM Watson's success at Jeopardy and aims to utilize the crowd to answer WWTBAM questions accurately. We developed an Android app to let the crowd (i.e. the participants) to play the game with their mobile devices while they watch the quiz show on TV simultaneously[1]. Our WWTBAM app has been downloaded and installed more than 300,000 times and it has enabled us to collect large-scale real data about MCQA dynamics. Over the period of 9 months, we have collected over 3 GB of MCQA data using our Android app. In our dataset, there are 1908 live quiz-show questions and 214,658 answers to those questions from the participants.

Studying the data we collected, we find that by just going with the most selected answer in the aggregation (we call this the majority voting - *MV* scheme), we can answer 92% of the questions correctly, however, these are mostly entry-level questions. In the show, there is a safe zone between 7th and 8th level questions, after which the questions become much harder. After question 7, the success rates of *MV* plunge quickly down to 60%. In other words, the correct answer is half of the time not the most popular answer for these harder questions. Ideally the final aggregated answer should lean towards the answers of capable rather than unreliable participants, but the ability of each participant is unknown a

[1]The app also has offline question answering option for the participants who are not present when the show is on air.

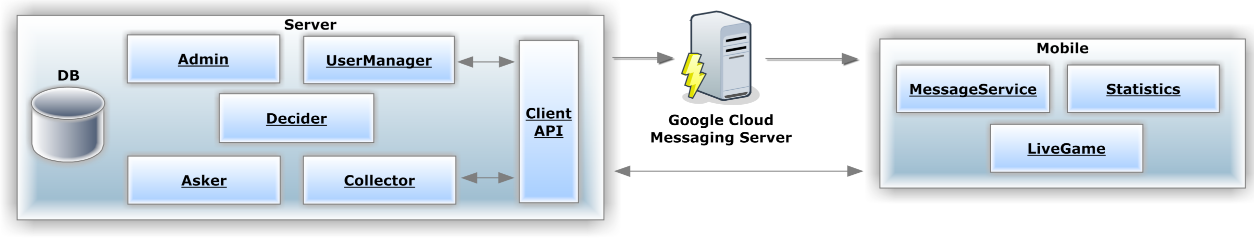


Figure 1: The system architecture

priori.

In this paper, we investigate how we can improve the success rates of these later/harder questions. To this end, we investigate novel weighted aggregation schemes for aggregating the answers obtained from the crowd. We propose to integrate the process of estimating participant abilities and deriving correct answers in a unified model. The basic idea in our proposed scheme is that the correct answer is obtained by a weighted voting among multiple participants where more reliable participants have higher weights. The participants who provide correct answers more often and more confidently will be regarded more reliable.

To enable this effective aggregation technique, we introduce two improvements. First, we ask the participants how confident they are with their answers just after they choose an answer. Then, for optimizing crowd's results on harder questions, we use lightweight ML techniques(Dai, Weld, and others 2011; Kittur et al. 2013). We weight participants' answers according to their performance and the confidence of their answers. We propose a novel method to estimate both participant weights and aggregated answers simultaneously. Our method is inspired by HITS algorithm (Kleinberg 1999) for webpage authority and truth discovery (Yin, Han, and Yu 2007) for conflict resolving: The candidate answers from reliable participants should be weighted more in the aggregation. Then using the optimized co-variants we aggregate weighted answers in order to decide a final answer.

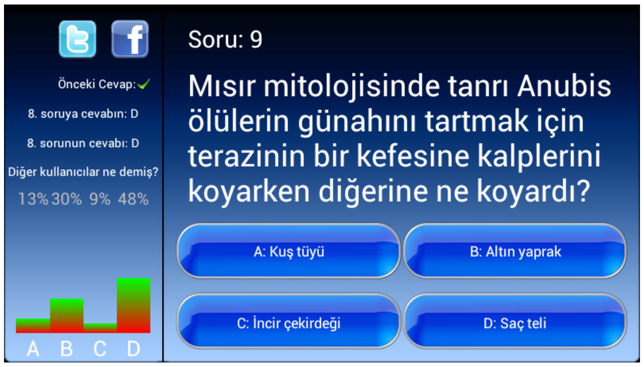
Our results present the effectiveness of using participants' confidence while answering the questions to solve this problem. We are able to pull up the accuracy rate for the harder questions by 15% over that of *MV* and to over 90% average accuracy, by using optimized weights for answers derived from participants' confidence. These findings suggest that it is feasible to build the superplayer by aggregating answers from many players effectively using lightweight ML techniques. Our WWTBAM application provides a good case for the benefits of applying AI techniques for building more accurate crowdsourced QA systems. In future work, we will investigate adapting lessons learned from the WWTBAM application to general/location-based crowdsourcing applications and recommendation systems.

The rest of the paper is organized as follows. We start with summarizing the design, development, deployment, maintenance and use of our application in Section 2. Then we provide the details about our dataset in Section 3. In Section 4 we detail our methods to use this dataset for MCQA , starting with the basic majority voting. We present our results in Section 5. We mention the related work in Section 6 and we conclude our discussions in Section 7.

2 *CrowdMillionaire* Architecture

In this section, we detail the architecture of our crowdsourced WWTBAM application: *CrowdMillionaire* . *CrowdMillionaire* enables the audience to play WWTBAM with their smartphones while watching the game show on TV simultaneously. We targeted the Turkish audience due to high Android penetration and popularity of the show in Turkey. Our mobile app's first version for the public use was released on November 08, 2012. Until now, the mobile app has been installed more than 300,000 times (app) and it has enabled us to collect large-scale real data about MCQA dynamics.

The overall architecture of *CrowdMillionaire* is shown in the Figure 1. *CrowdMillionaire* consists of three main parts, (1) an admin part for entering the questions & multiple choices while the game show is live on the TV, (2) a mobile side for presenting the questions to the users and letting them answer the questions, and (3) a server side for dispatching the questions, collecting the answers, providing statistics and running our MCQA algorithms.



Mısır mitolojisinde tanrı Anubis ölülerin günahını tartmak için terazinin bir kefesine kalplerini koyarken diğerine ne koyardı?

Figure 2: A screenshot of the *CrowdMillionaire* mobile app during an actual live game. The game and the questions are in Turkish. The question and choices translate as "Question:9 In Egyptian mythology for weighing dead people's sins, what does the god Anubis put on the pan of the scale while there is their hearts on the other salver? A) Feather B) Golden leaf C) Fig seed D) Strand"

When the show is on air, *CrowdMillionaire* sends an audible notification to alert the participants. They pick up their phones or tablets and start playing by answering the questions which they see both on the TV and *CrowdMillionaire* simultaneously, and enjoy the realtime gameplay. Figure 2 shows a screenshot of the *LiveGame* user interface (UI). During the live game, the participants see the current question & four choices in big and clear fonts. On a smaller frame, the answer of previous question is shown along with statistics about it.

On the admin side, the project members type the questions and multiple choices as they appear on the TV. We design the system the way that enables the team members to enter text as fast as possible using the admin UI while the game show is on air. For this purpose, we divide the input job into pieces: one of the project member types the question while another types the choices simultaneously. This web based admin UI, divide and distribute tasks dynamically if more team members are available. It also enables the typing task done on a single admin machine, when only one team member is available. As soon as the typing is completed by the team members, the server aggregates the inputs to form the question & four choices and send them to the participants. As a future work, we plan to extend dynamic admin UI to a crowdsourced question entering mechanism (Lasecki et al. 2012), and make the *CrowdMillionaire* a self-sustaining system.

The server collects input from admin machines, packages the question & the choices with some statistics and forward them to all participants. Then, the participants select their answers along with their confidence levels for their answers, using their phones. Finally, the data is sent to the servers through the *Client API*s and saved in the database.

When the game is not live on the TV, our server machine on the Amazon cloud (AWS EC2)(aws), serve the participants for the registration, the updates, the offline game playing and some other participant queries such as the game statistics. Furthermore when the show is on air, we run additional AWS EC2 instances to prevent race conditions and bypass buffering queues. So that, we provide realtime gameplay experience by fast question arrival and accurate live game statistics. We leverage Google Cloud Messaging (GCM) (gcm) push service to send the questions and live statistics to the mobile devices of the participants. Elasticity in our server design, dynamic admin UI and delegating the question forwarding to GCM help us to overcome the scalability issues we might encounter at any stage.

A last note about mobile side: the participants are incentivized by the joy and the desire of the gameplay itself. To this end, they are forwarded to *Statistics* screen when the live game is over. This component provides the list of the most successful 100 participants of the game, ranked based on the correct answer count and the response time. Note that, this ranking algorithm also take the participants' confidence choices into account: they gain/lose more points if they are right/wrong on their answers on which they were more confident.

3 Dataset

In this section we give details about our MCQA related data. Over the period of 9 months, we collected over 3 GB of data using our *CrowdMillionaire* app. In our dataset, there are 1908 questions and 214,658 unique answers to those questions from *CrowdMillionaire* participants. In addition, we have more than 5 million offline answers for archived live questions.

Our dataset includes detailed information on the game play. For example, our exhaustive timestamps show (1) how much time it took for a question to arrive to a participant, (2) when the question is actually presented to the participant on her device, and (3) when exactly the participant answered the question. After we clean and anonymize the data, we will share this dataset with the research community in order to advance the understanding of the MCQA dynamics. We consider this as another contribution of our work.

Below we present some basic statistics about our data:

Program statistics: Our dataset includes live game data from December 2012 to September 2013 (excluding the TV show's summer break). During this period, 2 or 3 programs were aired on TV per week which adds up to 80 TV programs. For each program the number of the contestants, who are actually answering the questions on TV, changes depending on the success of the contestants. The more the contestants are successful, the less the number of contestants present during a single TV show. We have 3.5 mean number of contestants for each program.

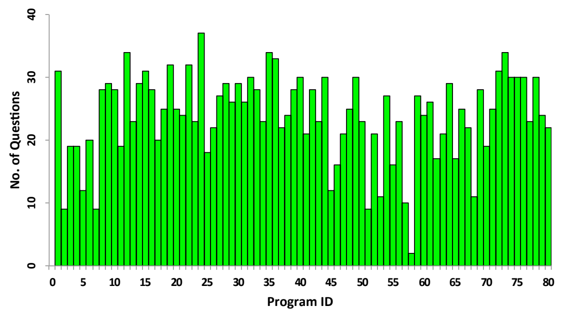
Figure 4 shows the cumulative distribution for the number of program played per participant. As the Figure indicates, we strive to keep the same group of participants for each program. Nearly, 98% of our participants played less than 5 games. On the other hand, we have a good number of participants for each program regardless of their continuity. Figure 3(c) shows the number of participants who played WWTBAM using *CrowdMillionaire* . Average number of participants per game is 733. We had more participants for earlier programs[2]. It arises up to 3000 on the most popular time of our app. *CrowdMillionaire* continues to attract a few hundreds of people every program.

Question statistics: Figure 3(a) shows the number of questions per program. As the figure indicates, the number of questions per program ranges from 9 to 37[3]. Mean number of questions per program is 23.85. Figure 5 shows number of questions by question level. Apparently, the earlier questions are more, because the later questions are presented if only the contestant is not eliminated. The number of questions decreases in an exponential style because the difficulty of the questions increases by the level.

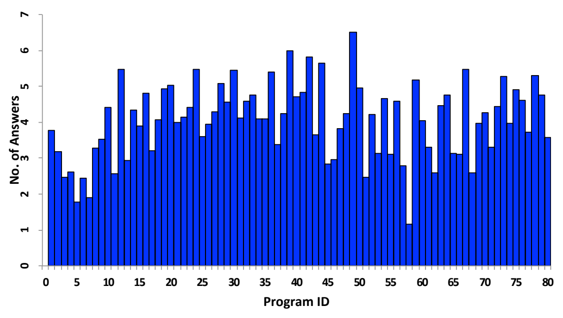
Answer statistics: Figure 3(b) shows the average number of the questions answered per participant for each program. The mean number of questions answered per participant in programs ranges from 2 to 7 questions. As the Figure 3(b) indicates, the average number of questions answered per participant per program stabilized with 4 to 5 questions. On the other hand, Figure 3(d) shows average number of answers

[2]This is due to the removal of other similar apps from the Google Play Store because of the copyright claims–Since we comply with the copyright of the game, we haven't had any issues.

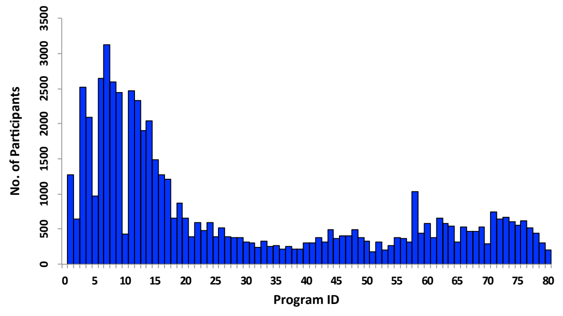
[3]The only exception is the TV show in which the contestant was able to see the final question that is worth 1 million in Turkish currency. In that special show, there were only 2 questions.



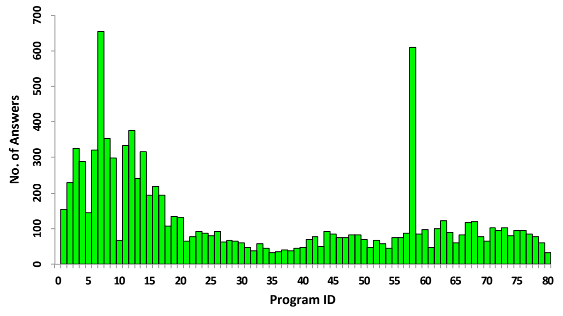
(a) Number of questions for each program.



(b) Average number of answers per participant for each program.



(c) Number of participants for each program.



(d) Average number of answers per question for each program.

Figure 3: Program Statistics

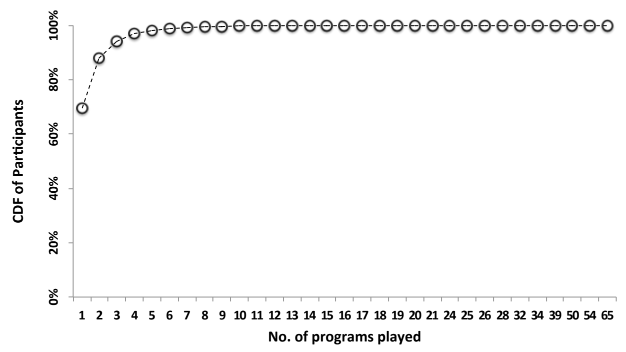


Figure 4: CDF for number of program played per participants.

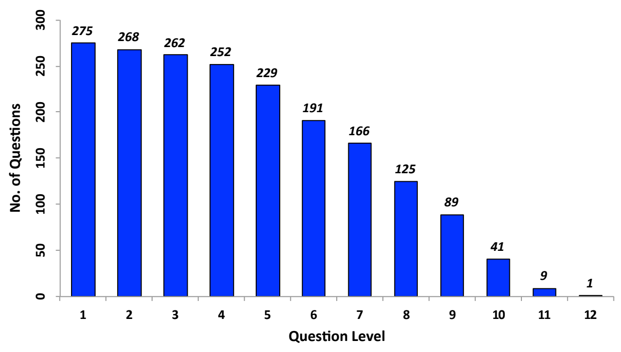


Figure 5: Number of questions by question level.

per question for each program. Although the average answer per question is stable around 100, it changes according to the popularity of a WWTBAM program (i.e. episode on TV), or the popularity of our *CrowdMillionaire* app at the time. The peak number for average answers was above 600. This happened once *CrowdMillionaire* was the only game on the Google Play to play WWTBAM [2], and once again when the final question appeared [3].

4 Methodology

In this section, we present our methods to incorporate crowdsourcing and ML techniques in order to build a crowdsourced WWTBAM player. Our objective is to collect the answers from the crowd, and then aggregate them using ML techniques to accurately answer the questions. Below we formulate the problem and then introduce our solutions.

Problem Formulation: We have a question set $\mathcal{Q}$, and each question in this set $q \in \mathcal{Q}$ is answered by a set of participants $\mathcal{P}_q$. For a given question q, each participant in the set $p \in \mathcal{P}_q$ gives a candidate answer x_q^p, and it can be one of the choices in set $\mathcal{S} = \{A, B, C, D\}$. Our objective is to aggregate candidate answers $\{x_q^p\}_{p \in \mathcal{P}_q}$ among participants, and get an aggregated answer x_q^* for each question q.

4.1 Majority Voting (*MV*)

A naive solution for aggregating the candidate answers is to select the choice which is claimed by the majority of the participants. Recall that, each candidate answer x_q^p can be one of the choices in set $\mathcal{S}$. Therefore in majority voting, for each question q, the highest voted choice by the participants is selected as the final answer x_q^*:

$$x_q^* = \arg\max_{x \in \mathcal{S}} \sum_{p \in \mathcal{P}_q} \mathbb{1}(x_q^p = x), \tag{1}$$

where $\mathbb{1}(\cdot)$ is an indicator function, which outputs 1 when x_q^p is the same as x; otherwise, outputs 0.

4.2 Confident Only Voting (*CO*)

Each candidate answer x_q^p has a corresponding confidence label c_q^p attached. Namely, when a participant provides an answer x_q^p, she is asked to indicate how confident her answer is, and this confidence label c_q^p is attached to her answer. A participant can define her confidence level using one of our predefined confidence labels which are {"certain", "guessing", "no idea"}.

We propose to filter the collected data by only choosing x_q^p which has a confidence label "certain". To this end, for each question q we define a participant set $\mathcal{P}_q^{certain}$ in which each candidate answer x_q^p has a confidence label c_q^p equals to "certain". Then, we run the majority voting on this subset to decide the aggregated final answer x_q^* for each question q:

$$x_q^* = \arg\max_{x \in \mathcal{S}} \sum_{p \in \mathcal{P}_q^{certain}} \mathbb{1}(x_q^p = x). \tag{2}$$

4.3 Confidence Weighted Voting (*CW*)

The aforementioned two algorithms are unweighted in the sense that participants have equal weights when aggregating their answers. In the following, we propose several weighted aggregation strategies. As the basic weight covariant, we use the answer confidence value c_q^p defined above. Instead of eliminating the candidate answers which are not labeled with "certain" as in the above method, here we set weights w_q^p according to the confidence labels c_q^p. Our insight here is to give higher weights to the answers of participants who are more confident, and lower weights otherwise as follows:

$$x_q^* = \arg\max_{x \in \mathcal{S}} \sum_{p \in \mathcal{P}_q} w_q^p \cdot \mathbb{1}(x_q^p = x), \tag{3}$$

where w_q^p is set according to the confidence label c_q^u by this rule: for confidence labels "certain", "guessing" and "no idea", we set the weights to be 3, 2 and 1 respectively.

4.4 Participant-Mine Voting (*PM*)

In our WWTBAM game scenario, it is natural to assume that some participants perform better than others. In order to leverage this fact, we introduce the concept of participants' weights, denoted as w_p. Note that w_p is different than w_q^p as it is not specific to a question. In this algorithm, if a participant's weight w_p is high, the candidate answers from her will be more trustful. We can incorporate such weights into the aggregation using weighted combination. Thus in a weighted aggregation scheme, we trust more in the candidate answers x_q^p that are provided by the participants with better performance.

In this scheme, we should figure out how to link participant weights with their performance. To tackle this challenge, we propose a novel optimization method to estimate both participant weights and aggregated answers simultaneously. The basic principle is inspired by HITS algorithm (Kleinberg 1999) for webpage authority and truth finding (Yin, Han, and Yu 2007) for conflict resolving: The candidate answers from reliable participants should be weighted more in the aggregation, and the participants who provide correct answers should be assigned higher weights. Based on this principle, we formulate the task as the following optimization problem:

$$\begin{aligned} \min_{\{x\},\{w_p\}} \quad & \sum_{q \in \mathcal{Q}} \sum_{p \in \mathcal{P}_q} w_p \cdot d(x_q^p, x) \\ \text{s.t.} \quad & w_p \geq 0, \quad \sum_p w_p = 1, \end{aligned} \tag{4}$$

where $d(\cdot)$ is a distance function which measures the difference between x_q^p and x. In order to regularize the participants' weights and prevent them to be infinity we set $\sum_p w_p = 1$ as a constraint function .

From Eq. (4), we can see that there are two sets of unknown variables, i.e., $\{x\}$ and $\{w_p\}$. Therefore, a natural way to solve this optimization problem is to use block coordinate descent techniques (Bertsekas 1999). Specifically, we adopt a two-step procedure which iteratively updates one set of variables to minimize the objective function while keeping the other set of variables unchanged.

As our proposed solution is an iterative procedure, it may lose some information if we assume that at each step only one candidate answer is true. To avoid this disadvantage, we represent candidate answers and aggregated answers by index vectors, which allow us to consider all possible values during the iterative procedure. For example, candidate answer "A" can be coded as $(1, 0, 0, 0)$, "B" is $(0, 1, 0, 0)$, and so on. In this case, we can use L^2-norm function for the distance function $d(\cdot)$ to measure the distance between two index vectors.

4.5 Confidence-Mine Voting (*CM*)

Here we proposed another method to combine Confidence Weighted Voting (*CW*) and Participant Mine Voting (*PM*) to utilize the power of weighted aggregation. The idea is simple: on top of the *PM* method, we incorporate the confidence labels into the index vectors. For example, consider a participant who provides a candidate answer "A". Then, based on the confidence label attached to her answer, the index vector is defined as follows:

If her answer has a confidence label "certain", then the index vector will be $(1, 0, 0, 0)$. If the confidence label is "guessing", then the index vector will be $(\frac{1}{2}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$. Finally, if the confidence label is "no idea", then the index vector will be $(\frac{1}{3}, \frac{2}{9}, \frac{2}{9}, \frac{2}{9})$. For the candidate answers "B", "C" and "D", they will be transformed into the index vectors in similar way. Then running our optimization algorithm defined in Eq. (4)

on these vector values, we calculate a weight $\{w_p\}$ for every participant $p \in P$. Finally based on these calculated weights, we choose an aggregated answer x_q^* for every question $q \in Q$ similar to the previous *PM* method. Thus, the only difference between *PM* and *CM* is the input vectors that we use to initiate the optimization.

4.6 Bing Fight

In order to compare the performance of our methods with the search engine based QA systems, we implemented the algorithm defined by (Lam et al. 2002). Despite the authors state Google performs better, we build the system based on Bing, because Google started to limit the number of queries for automated search requests.[4].

In the naive counting method, we produce the search engine query string by concatenating each choice with the question. Therefore, for each question in our dataset, we have four different search engine queries. We search these four queries on Bing, and calculate the total *number of result pages*, i.e. *number of hits* , for each of them. The final answer of this method is the choice with the maximum *number of hits* . Note that, we do not make any query modifications suggested in *DistanceScore* (Lam et al. 2002) algorithm such as ignoring some buzz words or decoupling phrases of the question and choices, because they are not in the scope of our work.

We define $\mathbb{B}(\cdot)$ as the function that returns number of result pages i.e. *number of hits* for a query string. In the following equation, $string_q$ denotes the question text as $string_x$ denotes a choice string, which is different for each of the four $x \in \mathcal{S}$:

$$x_q^* = \arg\max_{x \in \mathcal{S}} \mathbb{B}(string_x || string_q). \quad (5)$$

As an alternative, we also tested the backward keyword association approach defined in another work (Tonoike, Utsuro, and Sato 2004). It can answer only the 24.93% of the questions, which is the random success rate in a 4-choice classification. Thus, we do not discuss its results in Section 5.

$$x_q^* = \arg\max_{x \in \mathcal{S}} \mathbb{B}(string_x || string_q)/\mathbb{B}(string_x). \quad (6)$$

5 Evaluation

In this section, we evaluate our methods described in Section 4 using our WWtBaM data. Table 1 shows how each of these methods perform by question level. As shown in Figure 5, we do not have statistically significant number of questions from level 11 and 12 in our dataset. Thus, we do not include the results for those questions as they are not enough to accurately measure the performance of our algorithms. Note that, for any of the algorithms, if more than one choice has the maximum vote, then we count it as undecided.When calculating the accuracy of the algorithms, we count the undecided questions as failure (i.e. not as success).

We first run the Bing Fight on our question set. We directly queried via Turkish questions, so there is no translation effect. It can answer 30.50% of all questions successfully, that is a little less than the expected performance according to the previous works (Awadallah and Rauber 2006; Lam et al. 2002). This performance loss is due to several reasons such as: (1) the undecided questions, (2) using Bing instead of Google, which might perform better according to works mentioned in Section 6 and (3) not making the small query optimizations which are mentioned in the related work too. Ignoring the undecided questions, overall success rate increases to 37.06% which is closer to the results in previous works (Awadallah and Rauber 2006; Lam et al. 2002).

[4]At the time we write this paper, this limit is 100 queries per day

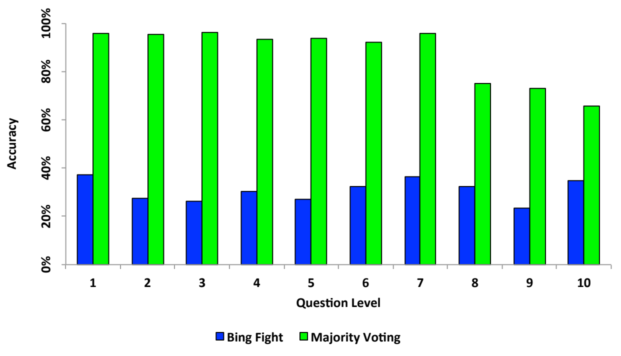
Table 1: Success Rates

Level	Bing	*MV*	*CW*	*CO*	*PM*	*CM*
1	37.09	96.00	97.68	**98.43**	97.69	96.00
2	26.87	95.89	**97.65**	97.60	97.28	95.52
3	27.10	96.17	97.20	96.77	**97.24**	96.17
4	30.16	93.25	94.96	95.76	**97.46**	95.24
5	26.64	93.86	94.04	95.83	**96.04**	95.61
6	31.41	92.14	91.80	92.22	94.95	**96.33**
7	34.94	95.78	94.34	96.82	95.18	**96.98**
8	31.20	75.20	73.50	79.82	90.58	**93.60**
9	24.72	73.03	70.59	77.87	90.91	**92.13**
10	34.15	65.85	60.00	72.22	84.10	**90.24**

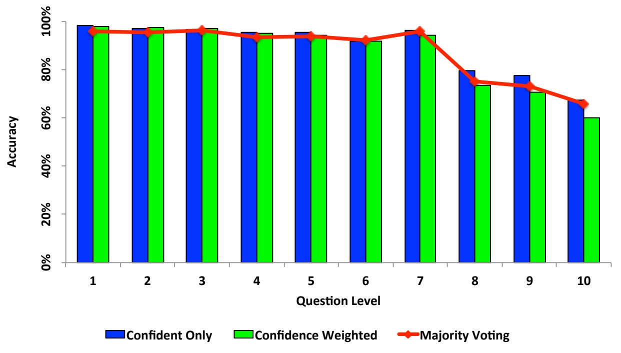
Using the *MV* algorithm, i.e. the elemental crowdsourcing algorithm, the crowd is able to answer 95% of the earlier/easier questions correctly. Although the *MV* is successful enough on those easier questions, its success rate plummets on the higher level questions as seen in the second column of Table 1 and in Figure 6(a). These results also represent an interesting fact about the format of the TV show: the success rates of the *MV* decrease more than 20% between the 7th and the 8th questions, because the hardness of the questions increase a lot after the safe zone of the TV show (i.e. 7th question). This distinctive change in the performance of crowd enables us to clearly categorize easier and harder questions based on question level. Figure 6(a) shows how the *MV* algorithm and Bing Fight performs by question level. It is clear form the graph that even the elemental crowdsourcing algorithm significantly outperforms the Bing Fight.

Regardless of whether they use the confidence data or not, all of our crowdsourced methods are able to answer roughly 95% of the lower level questions correctly. Namely, failure is a rare case. On those lower level questions, the *CW* or *CO* methods are able to answer some of the questions which the *MV* fails on, but on the other hand the *CW* or *CO* methods fail on some other questions which the *MV* answers correctly. However, when it comes to the higher level questions (i.e. 8th question and above), even the basic confidence-based methods seem to be more successful than the *MV* as seen on the Figure 6(b). Henceforth, our discussions will focus on the harder questions.

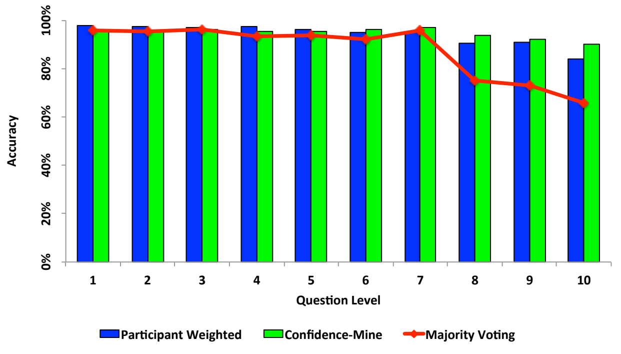
As it is clear from the Figure 6(b), the *CO* method slightly outperforms the *CW* method on the higher level questions. On the other hand, *CW* is a more robust algorithm as it is less dependent on the size of data. Although our data is big enough and we did not observe any problems caused by small data size while running our tests, picking only the "certain"



(a) Bing Fight vs Crowdsourced Majority Voting



(b) Majority Voting vs Confident Only Voting and Confidence Weighted Voting



(c) Majority Voting vs Participant Weighted Voting and Confidence-Mine Voting

Figure 6: Comparative performance results of our Methods

labeled answers can reduce the data size significantly. In some other domains, this reduced data size may cause problems, such as fluctuation and speculation on final answers as they may be dominated by a small number of faulty participants. Therefore, using the *CW* and the *CO* together in a hybrid fashion would be more dependable when the data size is not big enough. In that hybrid approach, *CO* function can lead the final decisions and *CW* can be used when *CO* cannot decide comfortably. That way, first only the "certain" answers will be counted, and in case of a close result then the hybrid algorithm will decide considering other answers too.

Figure 6(c) reveals that our optimized weight based methods perform significantly better than all the other methods, especially on the harder questions. On those harder questions, the *PM* method can successfully find the correct answers, for half of the situations where majority of the crowd fail. Then by incorporating the confidence data to the optimization, our *CM* method performs above 90% for all the question levels. This great improvement indicates using a lightweight training on our crowdsourced data, we are able to build a super-player for the WWTBAM game.

6 Related Work

6.1 MCQA studies on WWTBAM

In recent years, ML and IR communities have been producing large and growing literature on question answering(Gunning, Chaudhri, and Welty 2010). Today, there are many studies on question answering in the form of game playing. IBM Watson's (Ferrucci et al. 2010) Jeopardy challenge is the best known example of such systems. In order to outperform the human contestants, IBM Watson's Jeopardy system leverages NLP, ML and IR techniques. MCQA is drawing researchers' attention and there are several recent works on MCQA. WWTBAM has been regarded as a challenge for MCQA. Lam et al. (Lam et al. 2002) describe how search engines can be utilized to play WWTBAM game. Their basic method is to race the hit numbers of *question+choice* pairs by searching the possible combinations of such pairs on the search engines. With this naive strategy, out of 635 questions, their system is able to answer 44.4% of them using *MSN.com* and 55.6% of them using *Google*. In their advanced approach, they use the proximity of the question and the choice keywords as they appear in the search results. With the help of this strategy, their system is able to answer 68.9% of the questions. In (Tonoike, Utsuro, and Sato 2004), authors work on Japanese WWTBAM questions, and they try to detect the question phrase to decouple the choices before performing the hit race. With this keyword association, their system is able to answer the questions with 79% overall success rate. However, in another work (Awadallah and Rauber 2006), the same method performs 45% on Arabic WWTBAM dataset. The authors of the latter work, on the other hand, are able to pull the performance up to 55% with an improved method on Arabic questions. Their system also performs 62% on English questions. In a recent work (Molino et al. 2013) on WWTBAM, authors build a virtual player to play the game using Italian questions. Similar to IBM Watson's Jeopardy system, their virtual player leverages the information retrieved from Wikipedia. The overall success rate of their system is 76.33%. However, in none of the works above, the performance is proportional to the difficulty of the questions, because they are based on IR techniques as opposed to our system that leverages the crowd's intelligence.

6.2 Crowdsourced Question Answering

Crowdsourcing has been employed for answering subjective, relative, or multidimensional location-based queries for which the traditional search engines perform poorly. Two examples of location-based question answering systems are our previous work (Bulut, Yilmaz, and Demirbas 2011; Demirbas et al. 2010). AskMSR (Brill, Dumais, and Banko

2002) leans on the redundant data on the web instead of the complex NLP techniques. SMSFind (Chen, Subramanian, and Brewer 2010) is an example of SMS-based question answering systems. It leverages the search engines to answer the questions automatically. It tries to find the best matching answer by using both IR and NLP techniques. Although SMSFind is not a truly crowdsourced answering system, it is remarkable in the sense that the authors point out what type of questions are ambiguous to their system and where the human intelligence is needed. In another work, CrowdDB (Franklin et al. 2011), authors present an SQL-like query processing system to collect large amount of data with the help of microtask-based crowdsourcing. CrowdDB is piggybacked to Amazon Mechanical Turk (amt).

7 Conclusion

In this paper, we present crowdsourced methods using lightweight ML techniques to build an accurate MCQA system. The accuracy of our MCQA methods are promising. By using optimized weights for answers derived from participants' confidence, we are able to build a super player for the WWtBAM game that can answer the questions from all difficulty levels with an accuracy of above 90%. In future work, we will investigate adapting lessons learned from the WWtBAM application to general/location-based crowdsourcing applications and recommendation systems.

8 Acknowledgments

This work was supported in part by the National Science Foundation under Grant NSF IIS-1319973. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Amazon mechanical turk: Artificial artificial intelligence. `https://www.mturk.com/mturk`.

Google play page of application. `https://play.google.com/store/apps/details?id=edu.buffalo.cse.ubicomp.crowdonline`.

Awadallah, R., and Rauber, A. 2006. Web-based multiple choice question answering for english and arabic questions. In *Advances in Information Retrieval*. Springer. 515–518.

Amazon web services elastic compute cloud. `http://aws.amazon.com/ec2`.

Bertsekas, D. P. 1999. *Non-linear Programming*. Athena Scientific, 2 edition.

Brill, E.; Dumais, S.; and Banko, M. 2002. An analysis of the askmsr question-answering system. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP '02, 257–264. Stroudsburg, PA, USA: Association for Computational Linguistics.

Bulut, M. F.; Yilmaz, Y. S.; and Demirbas, M. 2011. Crowdsourcing location-based queries. In *PerCom Workshops*, 513–518.

Chen, J.; Subramanian, L.; and Brewer, E. 2010. Sms-based web search for low-end mobile devices. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, MobiCom '10, 125–136. New York, NY, USA: ACM.

Dai, P.; Weld, D. S.; et al. 2011. Artificial intelligence for artificial artificial intelligence. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Demirbas, M.; Bayir, M. A.; Akcora, C. G.; Yilmaz, Y. S.; and Ferhatosmanoglu, H. 2010. Crowd-sourced sensing and collaboration using twitter. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, 1–9. IEEE.

Ferrucci, D.; Brown, E.; Chu-Carroll, J.; Fan, J.; Gondek, D.; Kalyanpur, A. A.; Lally, A.; Murdock, J. W.; Nyberg, E.; Prager, J.; et al. 2010. Building watson: An overview of the deepqa project. *AI magazine* 31(3):59–79.

Franklin, M. J.; Kossmann, D.; Kraska, T.; Ramesh, S.; and Xin, R. 2011. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, 61–72. New York, NY, USA: ACM.

Google cloud messaging for android. `http://developer.android.com/google/gcm`.

Gunning, D.; Chaudhri, V. K.; Clark, P. E.; Barker, K.; Chaw, S.-Y.; Greaves, M.; Grosof, B.; Leung, A.; McDonald, D. D.; Mishra, S.; et al. 2010. Project halo updateprogress toward digital aristotle. *AI Magazine* 31(3):33–58.

Gunning, D.; Chaudhri, V. K.; and Welty, C. 2010. Introduction to the special issue on question answering. *AI Magazine* 31(3):11–12.

Kittur, A.; Nickerson, J. V.; Bernstein, M.; Gerber, E.; Shaw, A.; Zimmerman, J.; Lease, M.; and Horton, J. 2013. The future of crowd work. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, 1301–1318. New York, NY, USA: ACM.

Kleinberg, J. M. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46(5):604–632.

Lam, S. K.; Pennock, D. M.; Cosley, D.; and Lawrence, S. 2002. 1 billion pages= 1 million dollars? mining the web to play" who wants to be a millionaire?". In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, 337–345. Morgan Kaufmann Publishers Inc.

Lasecki, W.; Miller, C.; Sadilek, A.; Abumoussa, A.; Borrello, D.; Kushalnagar, R.; and Bigham, J. 2012. Real-time captioning by groups of non-experts. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, 23–34. ACM.

Lin, C. H.; Mausam; and Weld, D. S. 2012. Crowdsourcing control: Moving beyond multiple choice. In *UAI*, 491–500.

Molino, P.; Basile, P.; Santoro, C.; Lops, P.; de Gemmis, M.; and Semeraro, G. 2013. A virtual player for who wants to be a millionaire? based on question answering. In *AI* IA 2013: Advances in Artificial Intelligence*. Springer. 205–216.

Google now. `http://www.google.com/now`.

Apple siri. `http://www.apple.com/ios/siri`.

Tonoike, M.; Utsuro, T.; and Sato, S. 2004. Answer validation by keyword association. In *Proceedings of the 3rd Workshop on RObust Methods in Analysis of Natural Language Data*, 95–103. Association for Computational Linguistics.

Yin, X.; Han, J.; and Yu, P. S. 2007. Truth discovery with multiple conflicting information providers on the web. In *Proc. of the the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, 1048–1052.