Crowdsourcing Reliable Ratings for Underexposed Items

Beatrice Valeri[1,*], Shady Elbassuoni[2,†] and Sihem Amer-Yahia[3]
[1]*Department of Information Engineering and Computer Science, University of Trento, Trento, Italy*
[2]*Department of Computer Science, American University of Beirut, Beirut City, Lebanon*
[3]*LIG, CNRS, University of Grenoble, Grenoble, France*

Keywords: Crowdsourcing, Task Assignment, Cheater Identification.

Abstract: We address the problem of acquiring reliable ratings of items such as restaurants or movies from the crowd. A reliable rating is a truthful rating from a worker that is knowledgeable enough about the item she is rating. We propose a crowdsourcing platform that considers workers' expertise with respect to the items being rated and assigns workers the best items to rate. In addition, our platform focuses on acquiring ratings for items that only have a few ratings. Traditional crowdsourcing platforms are not suitable for such a task for two reasons. First, ratings are subjective and there is no single correct rating for an item which makes most existing work on predicting the expertise of crowdsourcing workers inapplicable. Second, in traditional crowdsourcing platforms there is no control over task assignment by the requester. In our case, we are interested in providing workers with the best items to rate based on their estimated expertise for the items and the number of ratings the items have. We evaluate the effectiveness of our system using both synthetic and real-world data about restaurants.

1 INTRODUCTION

Rating websites have gained popularity as platforms to share experiences and acquire recommendations about items of interest. For example, Yelp[1] is a popular website for rating and recommending businesses like restaurants, bars, etc. In most rating websites, all ratings are treated equally. However, in many cases, people provide misleading ratings, either because they are cheating or because they are not knowledgeable enough about the items they are rating. Moreover, in most rating websites, the number of ratings are not balanced across the items being rated. Unreliable ratings and the imbalance of ratings across items can both deteriorate the accuracy of most recommendation algorithms.

It would thus be beneficial to build a rating platform that takes into consideration rating quality and sparsity. The platform should ideally acquire the best ratings possible from the currently active crowd of workers based on their expertise with respect to the items. The system should automatically identify cheaters, i.e. workers that are consistently providing misleading ratings. We distinguish two types of cheaters: i) *lazy workers*, who assign ratings randomly to complete the rating tasks as fast or as effortlessly as possible, and ii) *malign workers*, who give misleading ratings to particular items in order to reduce or raise their average ratings. The platform should also prioritize acquiring ratings for underexposed items (i.e., items that have fewer ratings).

While traditional crowdsourcing platforms such as Amazon Mechanical Turk[2] and CrowdFlower[3] have been successfully used in various scenarios to acquire human knowledge in a cheap and effective manner, they are not suitable for our scenario. In such platforms, there is no clear notion of a worker expertise apart from a single score reflecting how well a worker performed on previous tasks. This is not feasible in our scenario for various reasons. First, differently from other tasks, rating tasks are subjective as the workers are asked to express their personal tastes. This means that there is no single ground truth (i.e., correct ratings for items) that can be used to compute expertise scores for workers, based on the re-

*The first author was an intern at LIG CNRS, University of Grenoble, France.

†The second author was funded by the American University of Beirut Research Board.

[1]www.yelp.com

[2]www.mturk.com

[3]www.crowdflower.com

sults of previous tasks. Second, workers might be more skilled to rate certain types of items than others. For instance, students might be more knowledgeable about holes-in-the-walls or cheap restaurants whereas professionals might be more knowledgeable about fancier restaurants. It is thus crucial to associate workers with different scores representing their expertise with respect to the different types of items being rated.

Finally, it is crucial to ask workers to rate only the items for which they have higher skills. In traditional crowdsourcing platforms, workers self-appoint themselves to tasks and requesters have no control over how the task assignment is carried out. In our case, we would like the platform to automatically assign tasks to workers based on their estimated expertise.

In this paper, we present a novel crowdsourcing platform that acquires reliable ratings for a set of items from a set of workers. A reliable rating is a truthful rating provided by an *expert* worker. Our platform estimates worker expertise based on the agreement of the worker with other *similar expert* workers in the system. The platform makes use of a fine-grained utility function to present workers with the best items to rate based on the workers' expertise and the number of ratings the items have. Finally, the framework automatically identifies cheaters and we experiment with various ways of dealing with them. Our platform is described in Section 4.

With this data acquisition, we do not plan to replace traditional volunteering-based ratings collected in recommendation systems, but we want to provide a novel channel for high-quality rating collection. For example, a new recommendation system needs high-quality ratings to be able to build recommendations for the first users. Through our crowdsourcing platform, the service owner can ask the crowd to rate items, motivating them with a reward, and build a dataset of reliable ratings to use.

We evaluated our framework using a set of exhaustive experiments on both real and synthetic datasets about restaurant. Our experimental results, presented in Section 5, clearly highlight the effectiveness of our system in acquiring reliable ratings from expert workers, particularly for underexposed items. We also show that identifying cheaters has a great impact on the overall rating quality.

According to our knowledge, this is the first work that addresses the issue of acquiring reliable *ratings* from the crowd. Most related work studied how to estimate the skills of crowdsourcing workers assuming the existence of only one valid ground truth (Dawid and Skene, 1979; Ipeirotis et al., 2010; Joglekar et al., 2013; Wolley and Quafafou, 2013; Hirth et al., 2011), which is not the case in our setting as we have already explained. Moreover, most related methods for estimating worker skills are generally post-processing methods, so they are not applicable in our scenario where we make use of the worker skills during task assignment to improve rating quality as more tasks are being performed. This is also not the case for existing work where one-time pre-task qualification tests are run to estimate worker skills. We review all relevant related work in detail in Section 2.
Our main contributions are the followings:

- We build a scalable and realistic crowdsourcing platform to acquire reliable ratings of items such as restaurants, movies or hotels.
- Our platform automatically estimates worker expertise based on the agreement of the worker with similar expert workers in the system.
- Our platform uses a carefully designed utility function to present workers with the best items to rate based on the estimated expertise of the workers and the number of ratings for those items.
- Our platform automatically identifies cheating workers and can dampen their effect.

2 RELATED WORK

In literature the idea of collecting information about items to recommend form people through micro-tasks has been already presented by Felfernig et al. (Felfernig et al., 2015) and by Hacker and von Ahn (Hacker and von Ahn, 2009). In both papers people are asked to express their opinion about some items, while they miss the control of answer quality and assignment of the items the worker is more expert about.

Much work has been also done on improving the data quality for crowdsourcing. It mostly focused on the joint inference of true labels of items and worker reliabilities (Dawid and Skene, 1979; Ipeirotis et al., 2010; Joglekar et al., 2013; Wolley and Quafafou, 2013; Hirth et al., 2011). However, all these methods suffer from two drawbacks. First, they assume the presence of one ground truth, even if it is unknown, which is clearly not the case for subjective crowdsourcing tasks such as the rating of items. Second, they are generally post-processing methods, so they cannot be seamlessly used during task assignment.

To address the first issue, Tian and Zhu (Tian and Zhu, 2012) studied the problem of worker reliability in crowdsourcing tasks in the case where more than one answer could be valid. They started from two mild assumptions on grouping behaviour: 1) reliable workers tend to agree with other workers in many tasks; and 2) the answers to a clear task tend to form tight clusters. Following this idea, they developed a low-rank computational model to explicitly relate the grouping behavior of schools of thought, characterized by group sizes, to worker reliability and task clarity, without the need of gold standards.

There have been various attempts to integrate worker reliabilities or skills with task assignments in crowdsourcing platforms. Li et al (Li et al., 2014) proposed a crowdsourcing platform that can automatically discover, for a given task, if any group of workers based on their attributes have higher quality on average and target such groups, if they exist, for future work on the same task. Auctions have been presented as an alternative way for task assignment by Satzger et al. (Satzger et al., 2012), assigning the task to the more skilled workers which volunteer for it respecting the economic constraint fixed by the requester. The cost/quality ratio has high importance also in the work of Karger, Oh and Shah (Karger et al., 2011). They proposed a new algorithm for deciding which tasks to assign to which workers and for inferring correct answers from the workers' answers, solving the problem of minimizing the total price (i.e., number of task assignments) to achieve a target overall reliability.

Ho and Vaughan (Ho and Vaughan, 2012) explored the problem of assigning heterogeneous tasks to workers with different unknown skill sets in crowdsourcing markets. They presented a two-phase exploration-exploitation assignment algorithm to allocate workers to tasks in a way that maximizes the total benefit that the requester obtains from the completed work. The same authors together with Jabbari (Ho et al., 2013) investigated the problem of task assignment and label inference for heterogeneous classification tasks, deriving a provably near-optimal adaptive assignment algorithm and showing that adaptively assigning workers to tasks can lead to more accurate predictions at a lower cost when the available workers are diverse. Most of the above work addressing the issue of task assignment and worker reliabilities however assumes the presence of single ground truth which is not applicable in subjective tasks such as the rating of items.

Roy et al. (Roy et al., 2013) proposed in a vision paper to rethink crowdsourcing as an adaptive process that relies on an interactive dialogue between the workers and the system in order to build and refine worker skills while tasks are being completed. In parallel, as workers complete more tasks, the system learns their skills more accurately, and this adaptive learning is then used to dynamically assign tasks to workers in the next iteration. This dialogue between the system and workers resembles the dialogue between users in trust-based systems (Mui et al., 2002). In the same way in which in an e-commerce website buyers rely on the sellers' reputation and on the past interactions to understand how much they can trust them, in a crowdsourcing platform the task requesters use the learned skills of workers to understand how much they can trust them. However, the meaning of worker skill is different from trust, as it includes also the specific capabilities of a worker.

3 PROBLEM DEFINITION

Given a set of workers W and a set of items I, our goal is data acquisition. That is, we want to acquire *reliable* ratings for as *many* items as possible where the set of possible ratings R is $\{0$ (don't know), 1 (don't like), 3 (neutral), 5 (like)$\}$. We map ratings to values between 1 and 5 to stay compatible with the 5-star rating paradigm used by most recommender systems.

More specifically, we want to populate a database of tuples of the form $T = < w, i, r >$, where w is a worker, i is an item, and r is the rating provided by w for item i. Our data acquisition has the following two sub-goals: 1) worker w should not be a cheater, and 2) item i should currently be the best item for worker w to rate, meaning that i is the item w is most knowledgeable about and that has the fewest ratings (the platform parameters give more importance to one or the other aspect).

Note that the first sub-goal, identifying cheaters, is important for the realization of our main goal, acquiring reliable ratings for items. In the second sub-goal, given that w is not a cheater, we want her to rate the items that have fewer ratings she is most knowledgeable about and for which she is most likely to give reliable ratings. In the next section, we describe our framework that realizes the above sub-goals to achieve our main goal of acquiring reliable ratings for as many items as possible from a crowd of workers.

4 FRAMEWORK

In a nutshell, our framework works as follows. First, we cluster the items to be rated into n itemsets $I_1, ..., I_n$ according to characteristics of the items. The clusters can overlap and the characteristics are inherent properties of the items. For example, in the case of restaurants, the characteristics can be cuisine, price range, etc. We cluster items into itemsets so that we can associate each worker in our system with an expertise level for each itemset that can help us assess how likely the worker can provide reliable ratings for

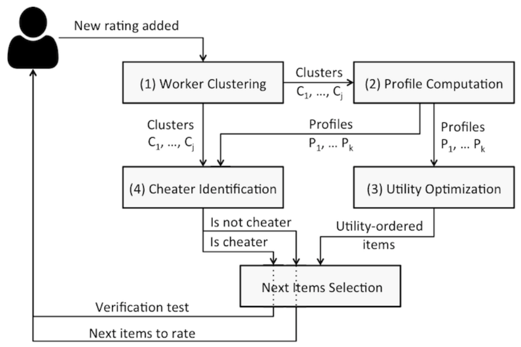


Figure 1: The proposed framework.

items in an itemset. A given worker can be an expert in one type of items and less expert or completely unknowledgeable for other types. Second, we constantly cluster active workers in the platform based on the ratings they provide: each cluster represents a group of workers with same tastes, different from the other ones. Clusters are used in our system for two reasons: 1) to compute the expertise of workers, and 2) to identify workers who provide misleading ratings which we refer to as *cheaters*.

A diagram of our framework is shown in Figure 1. Our platform works in task sessions: the worker enters a new task session with the first ratings she assigns and continues to rate the items the platform presents her till she leaves the session. Within a session, the reliability of her ratings is checked. Once a new worker joins the system, she is asked to rate the l items with the highest number of ratings in the system so far, to be able to compare her with as many workers in the system as possible. Based on her provided ratings, we assign the worker into one worker cluster. After a worker is assigned to a cluster, her profile is updated for each itemset she provided ratings for. Using the calculated profile, the worker is either suspected to be a cheater and is asked to pass a verification test, or she is asked to rate more items. A verification test is simply another set of rating tasks where a worker is asked to rate items she has rated before. A worker passes the verification test if she is relatively consistent with her previous ratings, otherwise she is banned from the system. The verification test is designed this way to disguise it from actual cheaters and to not turn off falsely-flagged workers. Other ways of dealing with cheaters are also applicable and we experiment with different methods in Section 5.

In case more items are to be rated (i.e., the worker was not flagged as a cheater or has passed the verification test), we apply a utility function that presents the worker with the items we anticipate that she has the most promise to rate reliably, prioritizing items that have fewer ratings. Our framework performs the above mentioned procedure for every worker that is currently active in the platform and the same procedure is repeated each time a new rating is provided until the worker leaves the system. In case a worker consistently fails to join a worker cluster after she has provided a given number of ratings, the worker is suspected to be a cheater and is asked to pass a verification test as before.

Our framework consists of four main components: 1) Worker Clustering, 2) Profile Computation, 3) Utility Optimization, and 4) Cheater Identification. We describe each component next.

4.1 Worker Clustering

Given the set of active workers W in the system, our goal is to cluster them into a number of clusters based on their ratings. Let w and w' be two workers in W and $I_{int} =< i_1, i_2, \dots, i_n >$ be the set of items that both w and w' rated. Also, let $R =< r_1, r_2, \dots, r_n >$ be the vector of ratings worker w provided for the items in I_{int} where r_k is the rating of item i_k. Similarly, let $R' =< r'_1, r'_2, \dots, r'_n >$ be the vector of ratings worker w' provided for the items in I_{int} where r'_k is the rating of item i_k. We cluster the workers based on an adjusted Euclidean-distance-based similarity measure which is computed as follows:

$$sim(w,w') = 2 * \left(\frac{1}{1 + \frac{\Sigma_{k=1}^{n}\left(\frac{r_k - r'_k}{r_{max} - r_{min}}\right)^2}{n}} - \frac{1}{2}\right)$$

where r_{max} is the highest rating possible (5 in our case) and r_{min} is the lowest rating possible (1 in our case). In the computation of similarity we are ignoring the “don’t know” ratings with value 0.

Note that using a standard Euclidean-distance-based similarity or a Cosine Vector similarity will not work in our setting, where the neutral rating (3) should be considered very similar to both “like” (5) and “don’t like” (1). In standard similarity metrics, instead, rating 3 is considered quite different from both 1 and 5. To overcome this, we started from the standard Euclidean-distance-based similarity metric

$$EuclideanDistanceSim(w,w') = \frac{1}{1 + \frac{\sqrt{\Sigma_{k=1}^{n}(r_k - r'_k)^2}}{\sqrt{n}}}$$

and substituted the simple distance ($\sqrt{\Sigma_{k=1}^{n}(r_k - r'_k)^2}$) with a relative distance by dividing it by the maximum distance there could be between any two ratings (in our case 4). This results in a value in the range of $[-1,1]$ and squaring it gives a value in $[0,1]$ which is a diminished distance to accommodate for the closeness between the neutral rating and the other two ratings. Finally, the whole fraction gives a result in the

Algorithm 1: Cluster Workers.

```
Input: current set of clusters $H$, provoking worker $w$,
  closeness threshold $\tau_C$
Output: new set of clusters
  if w is an exisiting worker then
    $C_w \leftarrow getCluster(H, w)$
    $removeWorker(C_w, w)$
    if $C_w$ is empty then
      $removeCluster(H, C_w)$
    end if
  end if
  $new \leftarrow createNewCluster(w)$
  $addCluster(H, new)$
  while TRUE do
    $max \leftarrow -\infty; first \leftarrow null; second \leftarrow null$
    for $i = 1$ to $|H| - 1$ do
      for $j = i + 1$ to $|H|$ do
        $closeness \leftarrow closeness(C_i, C_j)$
        if $closeness > max$ then
          $max \leftarrow closeness$
          $first \leftarrow C_i; second \leftarrow C_j$
        end if
      end for
    end for
    if $max > \tau_C$ then
      $merged \leftarrow mergeClusters(first, second)$
      $replaceClusters(H, first, second, merged)$
    else
      break
    end if
  end while
  return H
```

range of $[0.5, 1]$ (after calculating the average, adding 1 and then taking the reciprocal). Thus, we subtract $1/2$ and multiply by 2 to map the results back to the interval [0,1]. Of course, any other suitable similarity measure can be seamlessly used instead depending on the context for which the framework is used.

Our incremental clustering algorithm is shown as Algorithm 1. The algorithm is called whenever a new rating is provided, since it is a new evidence about what the worker knows and the goal of the clustering is to group together workers who have similar tastes and experiences. This is also the case when a new worker joins the system and there already exists a set of worker clusters. We utilize an *incremental hierarchical* clustering algorithm (Johnson, 1967). We opted for a hierarchical clustering rather than a flat one since the number of clusters is not known a priori and it changes over time as workers rate more items or as new workers join the platform. Hierarchical clustering is also best suited for incremental clustering as it avoids recomputing the full hierarchy of clusters each time a new rating arrives. Note that we do not store the full hierarchy of clusters at the end, but only store the final level ending with a flat set of clusters.

Our clustering algorithm takes as input 1) the worker who provided the new rating (or a new worker) which we refer to as the provoking worker w, 2) the current set of worker clusters H, and 3) a cluster closeness threshold τ_C, and it returns a new set of clusters. In case the provoking worker w was an existing worker who has rated a new item, we remove w from its own cluster and assign it to a singleton cluster. In case w is a new worker who has just joined the system, she is also assigned to a singleton cluster after providing a predefined number of ratings for the most rated items. The algorithm then follows a bottom-up approach, merging the two closest clusters and reducing the number of clusters by one at each iteration.

The closeness of two clusters C_i and C_j is computed as the smallest similarity between their workers (i.e., complete-linkage clustering) as follows:

$$closeness(C_i, C_j) = \min_{w \in C_i, w' \in C_j} sim(w, w')$$

Finally, we merge the two clusters C_i and C_j with the *highest* closeness. We keep on merging clusters as long as the following condition holds:

$$\exists_{C_i, C_j} closeness(C_i, C_j) > \tau_C$$

where τ_C is a threshold on the closeness between any two clusters to be merged.

We use complete linkage to ensure that when we remove the provoking worker from her cluster, the intra-cluster similarity either stays the same or increases. This in turn means that the affected cluster stays compact, does not need to be split to improve the clustering quality, and we can start improving clusters from the current configuration of clusters, without recomputing the complete clustering hierarchy. Since we only add one cluster at each step, very few iterations are needed for updating the clusters and this is independent from the number of clusters and workers.

4.2 Profile Computation

Each worker w is associated with a profile vector $< w.p_1, \ldots, w.p_n >$ representing the worker's expertise for each itemset I_j, where $w.p_j$ is the ordered pair $(w.p_j.known, w.p_j.skill)$. To compute this profile, we measure two aspects of the worker: how many items in I_j she knows (i.e., did not rate 0), which we refer to as $w.p_j.known$ and, for the items she knows, how much she agrees with other expert workers from her cluster, which we refer to as $w.p_j.skill$. The first component of the worker profile $w.p_j.known$ is computed as follows:

$$w.p_j.known = \frac{\#known(w,I_j)}{\#ratings(w,I_j)}$$

where $known(w,I_j)$ is the number of items that worker w knows in I_j (i.e., did not rate as 0) and $\#ratings(w,I_j)$ is the number of items in I_j she has rated so far (including the 0 rating).

The second component measures how skillful the worker is for the items she knows. For the skill component, we utilize the agreement of the worker w with other workers from her cluster. The intuition behind this is that the worker is expected to behave similarly to the rest of the workers in her cluster. Before we dwell into the details of how we compute agreement between workers, we need to decide on who to compute agreement with. One alternative is to compute the agreement of worker w with all other workers from her cluster. This is however prune to some fundamental issues. First, in case some non-expert workers are still present in the current worker cluster, their effect on the agreement might deteriorate the profile values computed for other, possibly, expert workers. Moreover, if we compute the agreement with all the workers in the cluster of w, we would need a lot more ratings to have sufficient enough ratings to compute agreements between workers. Note that agreement depends solely on ratings provided by workers for items in the current itemset of interest. This is a problem since we ideally would like to acquire minimum number of ratings from non-expert workers. In order to overcome the aforementioned issues, we propose the following. To compute the skill value $w.p_j.skill$ for worker w, we measure the agreement of worker w with only the top-k most expert workers for the itemset I_j. We explain how to retrieve the top-k most expert workers in a given cluster later.

Regardless of whether we measure agreement with all workers in a cluster or with only expert workers, the rest of the computation procedure for the skill component of a worker's profile is the same. To measure agreement between two workers w_i and w_j, we use the same similarity metric used for building our clusters described in the previous subsection. Our similarity metric is well adapted to our setting of ratings and is applicable even when only few items have been rated by both users.

Once we have the agreement of the worker w_i with all the top-k expert workers in her cluster, we aggregate the agreements by taking the *average* and use this as the skill for worker w_i on itemset I_j as follows:

$$w.p_j.skill = \frac{\Sigma_{w' \in top-k} agreement(w,w')}{k}$$

where top-k is the top-k most expert workers in w's cluster.

Retrieving the Top-*k* Most expert Workers in a Cluster. Recall that in order to compute the skill component of the profile of a worker w, we need to measure the agreement between w and the top-k most expert workers in her cluster C_w with respect to an itemset I_j. In order to retrieve these top-k most expert workers, we rank all the workers $w \in C_w$ in decreasing order of their skill components $w.p_j.skill$. We then take the top-k workers with the highest $w.p_j.skill$ values. Ties are broken arbitrarily using $\#known(w,I_j)$ which is the number of items that the worker knows (i.e., did not rate as 0) from itemset I_j.

Initially, we bootstrap the system with a set of experts, for instance, restaurant or movie critics. These initial experts are clustered based on their ratings and their skills are computed based on the overall agreement between them. At step n, when worker skills need to be updated, the top-k most expert workers are selected based on their skills computed at step $n-1$ and those are used to update the worker skills.

4.3 Utility Optimization

The goal of the utility optimization component is to pick the best item for a given worker w to rate. More precisely, we want to pick the items with few ratings the worker most likely knows and will be able to reliably rate. To be able to do this, we use a utility function that is composed of two sub-components. The first component, $SetUtility(w,I_j)$, takes into consideration the worker profile and the number of ratings the worker has already provided for the itemset I_j. The second component, $ItemUtility(w,i)$, takes into consideration the number of ratings available for the item i and the closeness of the item to other items the worker knows.

More precisely, given a worker w and an itemset I_j, the $SetUtility$ component is defined as follows:

$$\begin{aligned} SetUtility(w,I_j) &= \beta_1 \cdot (1 - \frac{\#ratings(w,I_j)}{MAX_k \#ratings(w,I_k)}) \\ &+ \beta_2 \cdot (w.p_j.known * w.p_j.skill) \end{aligned}$$

where $\beta_1 + \beta_2 = 1$, $\#ratings(w,I_j)$ is the number of ratings the worker w provided for I_j and $w.p_j$ is the profile value of worker w for I_j. The first component of the $SetUtility$ favors itemsets for which the worker has provided fewer ratings. The second component measures how expert the worker is with respect to the itemset.

Similarly, given a worker w and an item i, the $ItemUtility$ component is defined as follows:

$$\begin{aligned} ItemUtility(w,i) &= \beta_3 \cdot (1 - \frac{\#ratings(i)}{MAX_j \#ratings(j)}) \\ &+ \beta_4 \cdot \frac{\Sigma_{j \in I_k^w} sim(i,j)}{|I_k^w|} \end{aligned}$$

where $\beta_3 + \beta_4 = 1$, $\#ratings(i)$ is the total number of ratings for item i and I_k^w is the set of items that worker w knows (i.e., has not rated as 0). The similarity $sim(i,j)$ is the similarity between two items i and j and it can be measured based on characteristics of the items (e.g. geographic distance between restaurants).

The final utility function $utility(w,i)$ of item i belonging to itemset I_j for worker w is then computed as the average of $ItemUtility(w,i)$ and $SetUtility(w,I_j)$ as follows:

$$utility(w,i) = \frac{ItemsetUtility(w,I_j) + ItemUtility(w,i)}{2}$$

Note that the utility of item i for worker w or $utility(w,i)$ is equal to 0 if worker w has already rated item i since we do not want to acquire more than one rating for an item by the same worker.

Once the utilities of every item for a given worker w are computed, we pick the item i for which $utility(w,i)$ is maximum and provide this item to the worker w to rate.

4.4 Cheaters Identification

One constant goal of our framework is to identify cheaters, that is, workers who are consistently providing misleading ratings. We distinguish two types of cheaters: i) *lazy workers*, which assign ratings randomly to complete the rating tasks as fast or as effortlessly as possible, and ii) *malign workers*, which provide misleading ratings to particular items in order to reduce or raise their average ratings. Our platform makes use of its different components to achieve this task. Given a threshold τ_S, the worker w is considered a cheater if the following condition holds:

$$\forall_j w.p_j.skill \leq \tau_S$$

In addition, a worker w is considered a cheater if she consistently remains in a singleton cluster after m number of ratings have been collected (other than don't know or 0). In either case, the flagged worker is asked to pass a verification test by asking her to rate items she previously rated. We then measure the agreement between the new ratings and the old ratings, and if the agreement is below a threshold value τ, the worker is verified to be a cheater and is banned from the system. Otherwise, the worker profile is updated based on the agreement between the worker's new and old ratings. In the next section, we experiment with other strategies to deal with cheaters such as weighting down their ratings when aggregating items' ratings.

5 EVALUATION

We evaluate the effectiveness of our framework for acquiring reliable ratings from expert workers using four different sets of experiments. The first set verifies the quality of ratings acquired for a real dataset of restaurants by assessing the performance of a recommendation system after identifying cheaters. This set of experiments clearly highlights the importance of the identification of cheaters. In these experiments, we also test the effect of worker expertise with respect to itemsets on the quality of the ratings acquired.

Next we perform parameter tuning to study the effect of the different parameters in our system such as the clustering algorithm parameters, cheaters identification threshold and the weights used in the utility function. Parameter tuning was performed on both synthetic and real datasets about restaurants.

The third set of experiments studies our utility function more closely and compares it with a number of alternative utility functions to test its effect on the overall performance of the system. In all these three experiments, we used the case of lazy workers to represent cheaters, as it was easier to simulate and since the results of the experiments hold regardless of the type of cheaters. In the fourth and final experiment, we focus on the other type of cheaters, namely malign workers, which are workers who intentionally give misleading ratings to particular items in order to reduce or raise their average ratings. In particular, we evaluate the effectiveness of our framework in identifying such cheaters.

5.1 Rating Quality Experiments

Effect of Filtering Out Lazy Workers. The main goal of our work is to build a crowdsourcing service for collecting reliable high-quality ratings. One major application that could benefit from our work is recommendation, as we assume that using reliable ratings we can improve recommendation accuracy. To validate this hypothesis, we measured the difference in prediction errors by an off-the-shelf recommender algorithm when using the full dataset and when only using the ratings of trusted workers, i.e. removing the identified lazy workers.

To run this evaluation, we built a real dataset by collecting ratings for 50 selected restaurants in Grenoble, France, from students and researchers using a custom website. We had a total of 57 workers, seven of which were experts and 10 were lazy workers (assigning random ratings) and acquired a total of 540 ratings. The experts were colleagues very familiar with the restaurants in Grenoble.

We analyse the recommendations built on different subsets of our ratings to identify which one have higher quality and let the algorithm build better recommendations. We used user-based collaborative filtering as a recommendation algorithm, with cosine similarity on rating vectors to define, for each user, a fixed-size neighborhood of 10 most similar users. Such a configuration has been shown to perform quite well and is very popular in many successful recommendation systems (Lee et al., 2012).

We ran a recommender algorithm evaluation based on a 70-30 training-test split of data and root mean squared error (RMSE) as evaluation metric. We ran the algorithm using the training set as known ratings and predicted the ratings (computed as similarity-wighted mean of neighbors ratings) for the worker-item pairs already present in the test set. In this way, we compared the predicted rating and the real rating assigned by the worker to the item and measured the error (according to RMSE) the algorithm made. The smaller the error the better the prediction, and hence the better the quality of the data in the training dataset. For evaluation, we considered only "known" ratings (i.e. ratings 1, 3 and 5) and we split the dataset by time, identifying a specific date such that 70% of the ratings in our dataset were provided before that date (i.e. the training set) and 30% of the ratings were provided after (i.e. the test set).

Using the full dataset, we obtained an RMSE of *2.202*. Removing lazy workers, the RMSE was *1.021*. We can therefore conclude that the ability to isolate cheaters in this dataset reduced recommendation error by *53.6%*. This result is quite promising and shows the utility of cheater identification for a popular recommendation algorithm.

Effect of Filtering Out Ratings of Non-expert Workers. Moreover, we know which itemsets the workers are deemed to be more experts for, and we can exploit this information to filter out lower quality ratings (i.e., those for which workers are not considered to be experts enough to rate). Recall that our utility function makes use of the worker profile $w.p_j$ to identify the itemsets for which the worker can give the best ratings. This means that ratings provided to items in itemsets for which the worker has higher profile values should be of higher quality since the worker is considered to be an expert for items they contain. For this reason, by filtering out the ratings workers provided to itemsets they are less experts for,

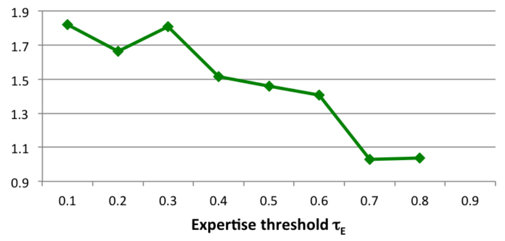


Figure 2: RMSE as ratings for itemsets on which workers are less expert about are filtered out.

i.e., with lower profile value, we will increase the quality of the ratings used to compute recommendations. Recall that the profile value of a worker w for an itemset I_j is composed of two components: 1) $w.p_j.skill$ which is measured as the agreement of worker w with the top-k most experts in her cluster with respect to itemset I_j, and 2) $w.p_j.known$ which is measured as the number of items in I_j the worker knows (i.e., did not rate as 0). Finally, the two components of the worker profile are then used in our utility function to represent the worker expertise as $expertise(w,I_j) = w.p_j.known * w.p_j.skill$.

The minimum expertise value we obtained in our dataset was zero, so we tried different expertise thresholds τ_E ranging from 0.1 to 0.9. After removing all ratings assigned by workers to itemsets for which they had expertise lower than the threshold, we split the dataset into training and test sets using a temporal cutoff as in the previous experiment (70% training set, 30% test set). We computed the RMSE for the same user-based collaborative filtering algorithm used previously (with a fixed-size neighborhood of 10 workers with the highest cosine similarity).

As can be seen in Figure 2, the RMSE decreases as the threshold τ_E increases, confirming that the ratings for which workers are more expert are of better quality and enable the recommendation algorithm to produce more precise predictions. The RMSE immediately falls to *1.819* with $\tau_E = 0.1$, and reaches the lowest value of *1.029* with $\tau_E = 0.7$. As the value of τ_E increases, we end up with too few remaining ratings and for $\tau_E = 0.9$ the recommendation algorithm is not able to compute any prediction. As can be seen, the final RMSE is always lower than the one we obtained before removing lazy workers (*2.202*).

Effect of Weighting Ratings by Worker Expertise. Filtering out the ratings of non-experts or flagged cheaters is a huge expense. In fact, they can still be of some value when aggregated. Another possibility is to weight the ratings by the expertise of the workers providing them when aggregating the items' ratings. To test the effect of this on the final aggregated ratings, we created two lists of aggregated ratings. In the first list, which we refer to the *unweighted* list, the ratings for each item were aggregated by taking the average over all the ratings provided for this item by all workers including lazy workers. In the second list, which we refer to as the *weighted* list, the ratings of each item were aggregated by taking a weighted average over all the ratings provided for this item by all workers (including lazy workers) such that each rating is weighted by the expertise of the worker that provided the rating at the time the rating was provided. To compare these two lists of aggregated ratings, we created a third list of aggregated ratings and used this list as a reference list. In this third list of aggregated ratings, the rating of each item was computed as the average of all ratings provided for the item by only "trusted" workers (excluding lazy workers).

We computed the RMSE (root mean squared error) for each of the two lists, the unweighted list and the weighted one, using the reference list as the "true" average ratings. We obtained an RMSE of *0.201* for the unweighted list and an RMSE of *0.077* for the weighted list, with a reduction of *62%* in average rating prediction. This clearly highlights the merits of weighting ratings by worker expertise when aggregating ratings. This can also be seen as another strategy for dealing with cheaters, instead of using a verification test and banning workers that do not pass it.

5.2 Parameter Tuning

In this set of experiments we study the effect of the various parameters of our framework on cheater identification accuracy. We generated a synthetic dataset and computed the accuracy of cheater identification for different values of the worker clustering threshold, the weights used in the utility function and the minimum skill threshold. We then tested the selected values on other larger datasets and a real-world one. Here we focused only on lazy workers as cheaters.

The synthetic dataset consisted of 100 fake restaurants, randomly placed in a 20-km diameter, divided into four non-overlapping itemsets I_1, I_2, I_3, and I_4, of the same size (i.e., 25 restaurants each). We generated 100 workers divided in the following way: 15 initial experts, 15 lazy workers and 70 trusted workers (i.e. providing truthful ratings). Each worker rated 40 restaurants, for a total of 4000 ratings with a value greater than zero (i.e., no don't know or 0 ratings). The 15 initial experts in our dataset were divided into three non-overlapping groups each consisting of five experts. The first group liked itemsets I_1, and I_2, the second group liked itemsets I_3 and I_4, and the third group liked I_1 and I_4. In order to make rating generation simpler, we assumed all expert workers either liked or disliked all the items they know in any given itemset for which they provided ratings. This is a simplification of a real-world scenario where it is more likely that workers will like some items in an itemset and dislike others in the same itemset. Similarly, our 70 trusted workers were divided into seven non-overlapping groups each consisting of 10 workers. The first three groups were similar to the three groups of experts, that is, they liked the same itemsets as the three groups of experts. The fourth group of trusted workers liked I_1 and I_3, the fifth group liked I_2 and I_4, the sixth group liked I_1 only, and the seventh and final group liked I_3 only. Rating generation for trusted workers was done as follows. All trusted workers gave a rating of 5 to items within the itemsets they liked with a 70% probability, and 3 (i.e., neutral) with a probability of 30% . The same happened for items they didn't like where a rating of 1 was generated with a 70% probability and a rating of 3 was generated with a 30% probability. Finally, the remaining 15 workers in our dataset were designed to be lazy workers with random ratings.

Using the above synthetic dataset, we tuned the different parameters in our framework. The first parameter is the similarity threshold for our clustering algorithm. Recall that our framework continuously re-clusters workers in the system as new ratings arrive. Our incremental hierarchical clustering algorithm merges clusters continuously until no two clusters can be merged (i.e., the closeness between any pair of clusters is lower than a threshold τ_C). We ran our algorithm with different values of τ_C keeping all other parameters fixed to some randomly selected values: $\beta_1, \beta_2, \beta_3$ and $\beta_4 = 0.5$ for the utility functions and $\tau_S = 0.3$ for the minimum skill threshold. We then computed the precision, recall and F2 measure for detecting cheaters, where

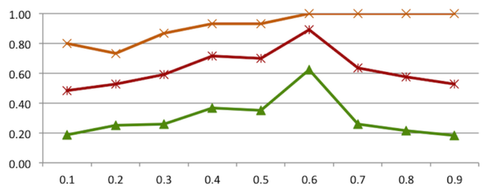
$$precision = \frac{\#true\_positives}{\#true\_positives + \#false\_positives}$$

$$recall = \frac{\#true\_positives}{\#true\_positives + \#false\_negatives}$$

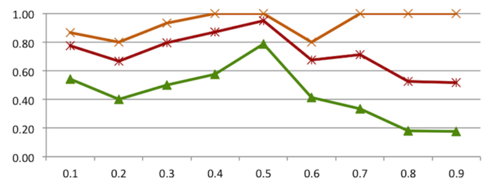
$$F2\_measure = (1+2^2) * (\frac{precision * recall}{(2^2 * precision) + recall})$$

We used the F2 measure since we wanted to sacrifice a bit of precision in favor of a higher recall. That is, we want to detect as many lazy workers as possible, even with the price of falsely flagging some trustful workers as cheaters. This is not a problem in practice, since trustful workers will eventually pass the verification test after being flagged as cheaters.

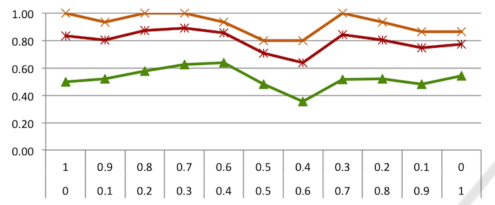
We obtained the highest F2 measure with $\tau_C = 0.6$ (see Figure 3(a)). We also measured the quality of the clusters obtained with our algorithm with respect to



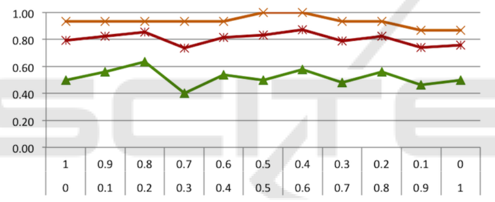
(a) *Best value for* τ_C.



(b) *Best value for* τ_S.



(c) *Best values for* β_1 *and* β_2*, with* β_2 *values on top and* β_1 *values on bottom.*



(d) *Best values for* β_3 *and* β_4*, with* β_4 *values on top and* β_3 *values on bottom.*

(e) *Legend.*

Figure 3: The best values for system parameters.

the ideal set of clusters computed once all the ratings were generated. On average, *76%* of workers were clustered correctly, with an *83%* precision when the selected similarity threshold was used. In a similar fashion, we identified the best values for the other parameters in our framework: $\tau_S = 0.5$ for the minimum skill threshold; $\beta_1 = 0.3$ and $\beta_2 = 0.7$ for itemset utility; and $\beta_3 = 0.6$ and $\beta_4 = 0.4$ for item utility. The results of this evaluation are shown in Figure 3.

To test the identification of cheaters on a larger dataset, we built four other synthetic datasets of bigger size, each one with 1000 workers and 300 items divided into six itemsets I_1 through I_6. The set of experts was composed of 100 workers equally divided into five groups, same for all datasets. The groups have tastes like the ones presented for the smaller dataset, with the addition of an itemset-independent taste: one group liked only items whose id was a multiple of three. This last group represents a more realistic group of workers who like and dislike items within the same itemset and across itemsets. Experts' ratings were generated with the correct rating (i.e., 1 for items the expert didn't like and 5 for items she liked) with 80% probability and the neutral rating (with value 3) with 20% probability.

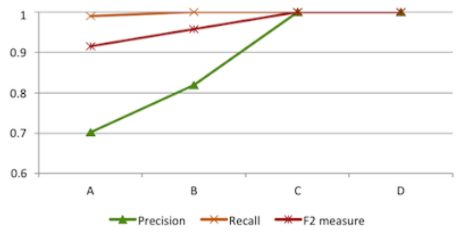


Figure 4: Results of the 4 synthetic datasets with 1000 workers.

The main difference between the four datasets in this evaluation is the number of lazy workers. Dataset A had 100 cheaters, dataset B had 200 cheaters, dataset C had 300 cheaters and dataset D had 400 cheaters. Trusted workers were equally divided into 10 different groups, five of which correspond to the ones of the experts while the rest are composed in the same way, but with different combinations of itemsets. Trusted workers gave correct ratings (either 1 or 5) with a 70% probability and neutral rating (i.e., 3) with a 30% probability. Each dataset contained around 120,000 ratings. As shown in Figure 4, our framework performs consistently well for all four datasets, reaching full precision and recall for dataset D (i.e., as the number of cheaters increase). Each task was run in about *2* seconds on average, from received rating to visualization of next item to rate, indicating the feasibility of our approach as a web service for crowdsourcing rating tasks.

Finally, we ran our framework with the best parameter values determined by the previous experiment on our real dataset. We obtained a precision of *0.727*, a recall of *0.800* and an F2 measure of *0.784*. These results confirm the ability of our framework to correctly identify cheaters in a real setting.

5.3 Utility Function Experiments

Our utility function has no influence on the identification of cheaters: cheaters will sooner or later reveal themselves as they rate more items, regardless of the order in which items are presented. The importance of the utility function is the time needed to identify cheaters. Clearly, the sooner they are identified, the better. If the framework needs to collect many ratings before identifying cheaters, this would be costly and it could happen that a cheater might stop giving ratings before the framework had had the chance to identify her as one. In this case, the ratings provided by this unidentified cheater would be considered reliable.

To test the effect of our utility function on the overall performance of the system, we compare it to two other baseline utility functions: i) a recommendation-based utility function, in which the next item shown to the worker is the one recommended to the worker according to the ratings she already gave using an off-the-shelf recommendation system, and ii) a random utility function, in which the next item is randomly selected. Recall that our proposed utility function is based on the number of ratings already assigned to an itemset, the profile of the worker for the itemset, the number of ratings already given for an item and the similarity of the item with other items rated by the worker. To test which utility function performs best, we analyzed the number of ratings the framework asked each cheater before identifying her, using our real dataset of restaurants. Using our real dataset, on average the random utility function needed to show *25* items and the recommender-based utility function needed to show *32* items, while our utility function needed to present only *17* items. These different results are statistically significant according to t-tests between the pairs (p-values: 0.0001, 0.008, 0.03, α-level: 0.05). This means that our utility function identified cheaters at least *32%* earlier than the other two functions.

Another important aspect of our utility function is that it keeps the number of ratings balanced over items which is a main goal of our data acquisition that distinguishes it from a recommendation system. When the framework chooses which item to show next, it gives higher priority to items that have fewer ratings, balancing in this way the number of ratings across items. To verify this, we analyzed how many ratings items had after *N* data acquisition rounds, for different values of *N*. We used standard deviation to compute rating distributions. The results are shown in Table 1. We can see that all utility functions have close values of standard deviation, with our utility function having smaller values for the first *150* ratings. A smaller deviation means that the number of ratings across items is quite balanced. When we consider a higher number of ratings, the standard deviation increases even with our utility function. This is mainly an effect of the initial items proposed to the worker when she arrives. Since the system aims to gather enough ratings per worker to be able to cluster them, the first items proposed to workers are those with the highest number of ratings. We thus end up with a small set of items that have more ratings than others, causing this problem of unbalanced ratings.

Table 1: Standard deviation of number of ratings per item.

N	Our utility	Recommendation-based utility	Random utility
150	0.2	0.53	0.59
300	0.99	0.53	0.57
600	1.2	0.67	0.57
1200	1.32	1.67	1.11
2000	1.94	1.81	1.75

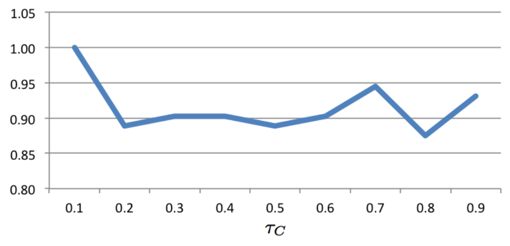


Figure 5: Average recall of malign workers identification.

5.4 Malign Workers Experiments

So far, we have only considered lazy workers as cheaters. There are other ways for cheating and a particularly appealing category of them are the malign ones. Malign workers are workers who intentionally give misleading ratings to particular items to reduce or raise their average ratings.

In this set of experiments, we added malign workers to our real and synthetic datasets and tested how many of them are correctly marked as cheaters. For the synthetic dataset, these malign workers provided a percentage of "truthful" ratings by following the same behavior of some of the experts in the system (as these behaviors are well defined). For the real dataset, the malign worker followed the behavior of the majority of the other workers (i.e., giving a positive rating when the majority gave a positive rating and vice versa). For the rest of the items, the malign workers provided opposite ratings to those provided by the experts or the majority depending on the dataset, reducing or raising the average ratings of these items.

We start by testing how many misleading ratings these workers had to give before being identified as cheaters. In the synthetic dataset used for parameter tuning (with 100 workers), we added 24 malign workers divided into four groups of six workers with different percentages of misleading ratings: 10%, 20%, 30% and 40%. The framework identified *87%* of the malign workers on average (21 of 24), and *71%* of the ones that it missed to identify had only *10%* of misleading ratings. Considering only the workers with a higher percentage of misleading ratings, the framework identified on average *95%* of the malign workers. We conclude then that the framework is able to correctly identify almost all malign workers when they give at least *20%* of misleading ratings.

We also computed the recall of malign workers' identification as we vary the clustering threshold (τ_C). Since workers are marked as cheaters when they fail to join clusters, a different value for this parameter could increase or decrease the amount of misleading ratings the workers should provide to be identified as cheaters by our framework. Figure 5 shows that the recall remains stable with varying τ_C values. On average, *75%* of malign workers that were not identified as cheaters had only *10%* of misleading ratings. This confirms the previous limit of *20%* of mialsedinag ratings as the minimum amount of biased ratings that a malign worker has to provide to be identified as a cheater by our system.

Finally, we confirmed the above results using our real dataset. We added 10 malign workers with 20% of misleading ratings and the rest of the ratings following the majority of the other workers, and the framework correctly identified *93%* of the malign workers on average. These results are quite promising, but marking malign workers as cheaters is only half of the work. To complete the work, malign workers should not be able pass the verification test. However, since malign workers are aware of their misleading ratings, they will be able to reproduce their ratings when the verification test is run. We leave the identification of a different verification test that is hard to pass for malign workers but not for trustful workers falsely flagged as cheaters to future work.

6 CONCLUSION

We presented a crowdsourcing platform to acquire reliable ratings of items. Our data acquisition platform differs from existing crowdsourcing systems and recommendation systems because it targets the most expert users to provide ratings for items with the fewest number of ratings. Our system relies on incremental clustering to identify cheaters and a carefully-designed utility function to assign items to rate to the most expert workers. Our experimental evaluation on both synthetic and real restaurant datasets showed that detecting cheaters, acquiring ratings from expert workers only, and automating the rating acquisition process all have a positive impact on both the cost of acquiring reliable ratings and on improving recommendation accuracy in popular recommendation systems. In the future, we plan to run more experiments on other datasets including movie datasets and to design other utility functions that are most adapted to such datasets.

REFERENCES

Dawid, A. P. and Skene, A. M. (1979). Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied Statistics*, 28(1).

Felfernig, A., Ulz, T., Haas, S., Schwarz, M., Reiterer, S., and Stettinger, M. (2015). Peopleviews: Human computation for constraint-based recommendation. In *CrowdRec*.

Hacker, S. and von Ahn, L. (2009). Matchin: Eliciting user preferences with an online game. In *CHI*.

Hirth, M., Hoßfeld, T., and Tran-Gia, P. (2011). Cost-Optimal Validation Mechanisms and Cheat-Detection for Crowdsourcing Platforms. In *FINGNet*.

Ho, C., Jabbari, S., and Vaughan, J. W. (2013). Adaptive task assignment for crowdsourced classification. In *ICML*.

Ho, C.-J. and Vaughan, J. W. (2012). Online task assignment in crowdsourcing markets. In *AAAI*.

Ipeirotis, P. G., Provost, F., and Wang, J. (2010). Quality management on amazon mechanical turk. In *KDD, Workshop on Human Computation*.

Joglekar, M., Garcia-Molina, H., and Parameswaran, A. (2013). Evaluating the crowd with confidence. In *KDD*.

Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, 2.

Karger, D. R., Oh, S., and Shah, D. (2011). Budget-optimal task allocation for reliable crowdsourcing systems. *CoRR*, abs/1110.3564.

Lee, J., Sun, M., and Lebanon, G. (2012). A comparative study of collaborative filtering algorithms. *arXiv preprint arXiv:1205.3193*.

Li, H., Zhao, B., and Fuxman, A. (2014). The wisdom of minority: Discovering and targeting the right group of workers for crowdsourcing. In *WWW*.

Mui, L., Mohtashemi, M., and Halberstadt, A. (2002). A computational model of trust and reputation. In *HICSS*.

Roy, S. B., Lykourentzou, I., Thirumuruganathan, S., Amer-Yahia, S., and Das, G. (2013). Crowds, not drones: Modeling human factors in interactive crowdsourcing. In *DBCrowd*.

Satzger, B., Psaier, H., Schall, D., and Dustdar, S. (2012). Auction-based Crowdsourcing Supporting Skill Management. *Information Systems, Elsevier*.

Tian, Y. and Zhu, J. (2012). Learning from crowds in the presence of schools of thought. In *KDD*.

Wolley, C. and Quafafou, M. (2013). Scalable expert selection when learning from noisy labelers. In *ICMLA*.