Learn or Earn? Intelligent Task Recommendations for Competitive Crowdsourced Software Development

Muhammad Rezaul Karim
University of Calgary
Calgary, AB, Canada
mrkarim@ucalgary.ca

Ye Yang
Stevens Inst. of Technology
Hoboken, NJ, USA
ye.yang@stevens.edu

David Messinger
Topcoder
Indianapolis, IN, USA
dmessinger@topcoder.com

Guenther Ruhe
University of Calgary
Calgary, AB, Canada
ruhe@ucalgary.ca

Abstract

Background: Competitive crowdsourced development encourages online software developers to register for tasks offered on the crowdsourcing platform and implement them in a competitive mode. As a large number of tasks are uploaded daily, the scenery of competition is changing continuously. Without appropriate decision support, online developers often make task decisions in an ad hoc and intuitive manner.
Aims: To provide dynamic decision support for crowd developers to select the task that fit best to their personal learning versus earning objectives, taking into account the actual competitiveness situation.
Method: We propose a recommendation system called EX^2 ("EX-Square") that combines both explorative ("learn") and exploitative ("earn") search for tasks, based on a systematic analysis of workers preference patterns, technologies hotness, and the projection of winning chances. The implemented prototype allows dynamic recommendations that reflect task updates and competition dynamics at any given time.
Results: Based on evaluation from 4007 tasks monitored over a period of 2 years, we show that EX^2 can explore and adjust task recommendations corresponding to context changes, and individual learning preferences of workers. A survey was also conducted with 14 actual crowd workers, showing that intelligent decision support from EX^2 is considered useful and valuable.
Conclusions: With support from EX^2, workers benefit from the tool from getting customized recommendations, and the platform provider gets a higher chance to better cover the breadth of technology needs in case recommendations are taken.

1. Introduction

As an emerging paradigm, crowdsourced software development (CSD) derives from general crowdsourcing by utilizing an open call format to recruit global online software workers to work on software mini-tasks [1, 2]. The success of competitive crowdsourced software development (CSD) depends on a large crowd of trustworthy software workers who are registering and submitting for their interested tasks in exchange for financial gains. A general competitive CSD process starts with task requesting companies distributing tasks with prizes online, and then crowd workers browsing and registering to work on selected tasks, and submitting work products once completion. Crowd submissions will be evaluated by experts and experienced developers, through a peer review process, to check the code quality and document quality [1, 3].

In crowdsourcing platforms, significant number of tasks are posted each week. Crowd workers need to spend significant amount of time (several hours per week) in reading individual task description and selecting appropriate tasks to work on. By the high complexity, the decisions to be made will be far from being best possible. Intelligent task recommendation can save developers time by recommending tasks based on developers existing technical skills, recent interests and current workload. Intelligent approaches are needed to not only matching developers expertise but also providing options for crowd workers to explore new technologies with high demand and buildup skills and knowledge in a more strategic manner. Software engineering is a knowledge intensive discipline [4]. Technologies are changing with quick intensity. Learning is a means to accommodate to all these changes. Learning is also a primary motivation factor for participation in CSD [3] and other forms of crowdsourcing [5, 6]. Baltassaare and Mirolli studied Intrinsically Motivated Learning Systems [7]. They found that greater exploration associates with higher performance. Here, exploration, more generally, refers to the act of searching for the purpose of discovery of information and resources.

In this study, we propose a recommendation approach that incorporates the individual Learn and Earn preferences. These targets correspond to exploration respectively exploitation search strategies

among all the tasks offered. By Exploration, we mean the selection of tasks requiring new technologies which the worker did not try before. An Exploration strategy can be employed by crowd workers to acquire skills in unexplored technologies and platforms. Exploitation refers to the selection of tasks requiring technologies where a crowd worker previously worked with or has significant expertise.

We focus on enabling workers decision making based on a systematic analysis of workers preference patterns in selecting tasks, relative hotness of technologies required in task requirements, and the projection of winning chances in consideration of competitiveness. In this paper, compared to former work [8], we have made new contributions in five directions:

- Conceptual: Task Recommendation is based on a balanced scoring of "Learn and Earn" preference which is a new concept and of practical interest for the case study company.
- Functionality: There is no longer focus just on predictions, but also with actionable insights on the ranking of recommendations in consideration of the dynamically changing competitiveness.
- Scope: Recommendations can be made for all tasks (not only ongoing ones) and for all workers (even for new ones, having no experience record yet).
- Implementation: The recommendation method EX^2 has a prototype implementation that was used to generate recommendations for the actual Topcoder workers that had signed up for our survey.
- Evaluation: Empirical evaluation is based on both a more comprehensive (more than two years of observations) and more recent data set (ranging up to March 31, 2017), and a user study with current TopCoder workers.

The paper is structured into eight sections. Section 2 introduces two motivating examples; Section 3 presents the modeling of the task recommendation problem; Section 4 details the experimental evaluation design; Section 5 reports empirical results; Section 6 discusses the results; Section 7 summarizes related work. Finally, Section 8 provides conclusions and an outlook to future work.

2. Motivating example

The motivation for our study came from a brief empirical analysis on task selection behaviors across crowd workers with a diverse background. The data analyzed for the motivational example include TopCoder data from January 1, 2016, to December 31, 2016. We compare two types of workers including expert workers and learning by doing workers.

2.1. How learning workers with no past success history select their tasks?

Unlike expert workers, learning-focused workers typically do not have any winning or submission success in the Topcoder platform. However, this constitutes an important pool of available (e.g. green, yellow belt workers in [9]) workers in crowdsourcing market, regarding experiencing competition, acquiring new skills, and expertise buildup. As an illustrative example, we look at the first-10 and last-10 tasks registered in 2016 by worker *testXuSanping*. She registered with tasks from different technologies and platforms but did not have any luck in succeeding a competition yet. To experience crowd competition and improve existing skills, she mostly registered with the tasks that required technologies and platforms tried before (exploitation behavior). She also explored new technologies in every few months (exploration behavior). For example, she started to register on tasks related to ReactJS technology from May 2016. She initially selected a few tasks where ReactJS was required with Node.js technology, which she previously worked with. We also noticed that since November 2016, she started to register on tasks requiring Predix, which was a completely new platform and technology to her.

2.2. How expert workers with significant success record select their tasks?

Expert workers are primarily driven by winning prizes in working on CSD platform. For example, the expert worker *seriyvolk83*, 90% of the time selected tasks from the web and mobile development domains where he had the most expertise. The rest of the time she selected tasks that required one or more of the unexplored technologies. Analyzing the 70 registered tasks of this particular worker, we noticed that he has significant expertise in Node.js, PostgreSQL, REST, Swift, Xcode, iOS, API, JSON and JavaScript technologies, with 80.9% winning rate and 87.3 % submission success rate in tasks requiring one or more of these technologies. To maximize his winning rate or to earn, he selected 63 tasks from these areas and seven from unexplored areas.

2.3. Discussion

Investigating a few more instance workers from the above two groups leads to similar observations in workers task selection preferences. In general, "learning"-focus or "earning"-focus corresponds to different search strategies for developers in looking for interested tasks to work on. In this study, we characterize and name such strategies as exploitation and exploration strategies, respectively. More specifically, we conceive that "earning"-driven workers mostly employ exploitation strategy to exploit their areas of expertise. With occasionally trying to acquire new skills by exploring new type of tasks; and "learning"-focused less experienced workers or newcomers frequently try to register tasks from multiple domains or platforms to minimize their loss chance.

We argue that the recommendations systems should take both the levels of exploration/exploitation preference into account due to the following four reasons. First, workers can benefit from more personalized recommendation based on their past task selection preference; Second, newcomers or workers with limited experience can benefit from exploration strategy; Third, as technologies often become obsolete and new platforms/technologies frequently arrive, crowd workers also need to spend time on acquiring new technical skills; and Fourth, crowdsourcing platform providers get a higher chance to better resource coverage on the breadth of technology needs in case recommendations are taken.

3. Recommendation system EX^2

In this part, we describe the main idea of the proposed task recommendation system EX^2, which searches for most relevant tasks for crowd workers, guided by individual workers preferences between Learn and Earn intention at any given time. While Earn represents the traditional view of looking for tasks that fit their existing expertise in the best way, the Learn portion is different as looking for new and highly requested technologies the worker is not yet familiar. The two components of the search are combined with a preference level of the worker towards the one or the other strategy. The top task recommendations are further supported by a projection of the winning chances. In what follows, we describe the basic notation and outline key parts of the recommendation system and its implementation.

Figure 1. Technology heat map from Topcoder [11] (image modified for better visibility)

3.1. Notation

For describing our proposed method, we need to introduce some basic notation[1]

- $tsim(s, s\prime)$ is the normalized (to [0,1]) textual cosine similarity between two tasks s and $s\prime$ using the title and description of the tasks. Here, the Apache Lucene implementation of Term Frequency-Inverse Document Frequency (TF-IDF) based similarity [10] is used. The normalized value [0,1] is obtained considering all task-task similarity scores.
- $\mathbf{T}(s)$ the set of technologies required for task s.
- $\mathbf{P}(s)$ is the set of platforms (e.g. Heroku, IBM Bluemix) required for task s.
- $h(q)$ is the hotness of a technology q required for the task s. Hotness is defined on a five-point scale from 1 (most popular) to 5 (least popular). Figure 1 shows an example technology heatmap from Topcoder [11]. The color-code reflects the relative level of demand and supply in a particular technology.
- $\mathbf{SS}(d, t)$ is the set of tasks for which worker d successfully submitted during recent M months.
- $\mathbf{SQ}(d, t)$ is the set of tasks for which worker d failed to submit (after registration) during recent M months.

3.2. Earn measurement

We have defined a measure for the degree of exploitation of worker d at time t when looking at task s. This measure is formulated as Equation 1. The set

[1] Bold characters refer to sets instead of variables.

$\mathbf{XS}(d) \cap \mathbf{T}(s)$ is the intersection of the technologies required for task s and the set of technologies that were needed in all the tasks where the worker d successfully submitted in the last M months.

$$EARN(d,s,t) = ps(d,s,t) * A(s) + (1 - ps(d,s,t)) * B(s) \quad (1)$$

In Equation 1, $ps(d,s,t)$ expresses the submission probability of task s for worker d at time t. The probability is defined as the ratio between number of submitted and number of registered tasks in the last M months considering tasks requiring at least $\mathbf{XS}(d) \cap \mathbf{T}(s)$ technologies. The value of $ps(d,s,t)$ lies in the range of 0 to 1.

$EARN(d,s,t)$ is a linear combination combing the technology usage in case of successful ($A(s)$) and unsuccessful ($B(s)$) submission of tasks s. The precise definition of both terms is given in Equations 2 and 3, respectively. Therein, $|\mathbf{Z}|$ stands for the number of elements of set Z. The higher the probability value $ps(d,s,t)$ (in Equation 1), the higher is the contribution of past success (submission), and the lower is the input of past failure.

What should be considered as recent information? In this paper, we take the results and information of the past M = 6 months tasks into account. If we would just consider one or two months of historical data, then we might miss essential information. Not all workers can submit and win every month.

$$A(s) = \sum_{s\prime \in SS(d,t)} \frac{|T(s\prime) \cap T(s)|}{|T(s)|} * \frac{|P(s\prime) \cap P(s)|}{|P(s)|} * tsim(s, s\prime) \quad (2)$$

$$B(s) = \sum_{s\prime \in SQ(d,t)} \frac{|T(s\prime) \cap T(s)|}{|T(s)|} * \frac{|P(s\prime) \cap P(s)|}{|P(s)|} * tsim(s, s\prime) \quad (3)$$

The newcomers will have $EARN(d,s,t)$= 0 for all new and ongoing tasks s, as the value of both $A(s)$ and $B(s)$ will always be zero. For workers having one or more registrations $A(s)$ or $B(s)$ can be greater than 0 only when each of $|T(s\prime) \cap T(s)|$, $|P(s\prime) \cap P(s)|$ and $tsim(s, s\prime)$ are greater than zero in the concerned equations. In this case, the value of $EARN(d,s,t)$ will be determined by $A(s)$, $B(s)$ and $ps(d,s,t)$.

For non-newcomer workers with no past success (no submission) at any of the registered tasks, $ps(d,s,t) = 0$ for all new and ongoing tasks s. What that means is that the failed tasks (tasks in $SQ(d,t)$ used in $B(s)$) will mainly contribute to the $EARN(d,s,t)$ scores for the new tasks. The higher the value of $B(s)$, the higher will be the value of $EARN(d,s,t)$. For workers with past success, both submitted (tasks in $SS(d,t)$) and failed tasks (tasks in $SQ(d,t)$) will contribute to earn measurement, depending on the value of the weighting factor $ps(d,s,t)$, $A(s)$ and $B(s)$.

3.3. Learn measurement

We also define a measure for the learning portion of the search function. The exploration score $LEARN(d,s,t)$ of worker d at time t for task s is the weighted sum of the hotness of the required technologies $T(s)$ for task s.

$$LEARN(d,s,t) = \sum_{i \in T(s)} w(i) * h(i) \quad (4)$$

For each technology i in $T(s)$, the weight $w(i)$ is calculated as $1/(1 + o(i,d))$, where $o(i,d)$ indicates the frequency of developer d participating in historical tasks that require technology i during the last M months. The more developer d used a required technology, the less the technology will contribute in the learn score of a task. For a task requiring unexplored technologies only (i.e. $o(i,d) = 0$ for all i in $T(s)$), the learn score is just the sum of the hotness scores of the required technologies. For two tasks requiring completely new technologies for a developer, the tasks with more hot technologies will have better learn score. For a newcomer worker, $o(i,d)$ is equal to 0 for all required technologies i in all ongoing tasks. Therefore, depending on the technological expertise of the non-newcomer workers, we may see $o(i,d) = 0$ for all required technologies i for a few ongoing tasks.

3.4. Preference-based recommendations of tasks

Computation of winning chance

We use $pw(d,s,t)$ for the winning chance of worker d on a new task s at time t. It is determined by applying machine learning based on former tasks. In this paper, at time t, we classified the winning chance using Random Forest machine learning [12]. Random Forest is an ensemble machine learning technique that applies bagging on decision trees. While building models, it samples the training records as well as the predictor variables. It can also automatically determine which predictor variables are important. The information related to all completed tasks before time t as well as the associated worker registrations, starting from 01 January 2015, are used to create training samples for learning.

Each training sample correctly represents a task-worker pair, where the worker registered for the task before time t and had an outcome (either won or not-won). Each training sample is labeled with either of the two values: Winner (if the associated worker won

the competition) and Non-Winner (if the related worker failed to win the competition). Thus model is built towards solving a two class classification problem. The dependent or class variable used in the model take two values: Winner and Non-Winner. Each test sample also represents a task-worker pair, where the associated task is an ongoing or a new task and the associated worker is a worker for which the class variable value (Winner or Non-Winner label) has to be predicted.

For each completed task i before time t, we create C_i training samples, where C_i is the number of registered workers for task i. If we have T_r tasks completed before time t, the total training samples in the training set will be $\sum_{i=1}^{T_r} C_i$. Without any filtering, training samples are extremely unbalanced and contains more samples from the non-winner class than the winner class. We followed a sampling process to make the training set balanced by reducing the samples from the non-winner class.

The non-winner training examples comes from two different types of task-worker pairs (prior registration information for the tasks completed before time t). In one case, the workers could submit in the registered tasks but failed to win the competition, while in the other case, the workers had to quit the registered tasks (no submission). The second case contributes majority samples for the non-winner class and also for the entire training data set. To make the training data set somewhat balanced, we randomly selected only 1% of the training samples from the second source. The samples from the first case added to the training samples without any filtering or selection. Once we have the training set ready, we build the model for time t and make prediction on the test samples.

Even though each training and test sample represents a task-worker pair, it consists of three type of features: task related features (the task represented in the current sample), worker related features (the worker represented in the current sample), and the competitor related features (features related to the other workers registered in the same task). Details can be found in [13].

To summarize, for each test sample, the built model predicts whether the developer d represented in the test sample will be the winner or non-winner in an ongoing or new task s associated with the sample. Thus, the built models work at the worker level and recommend tasks for the workers. Rather than directly using the predicted label for the worker on the task, we extract the predicted probability associated with the Winner class. This winning chance probability is used in Equation 5 as the value of $pw(d, s, t)$.

Ranking task recommendation

For performing search, we need to combine learning and earning based on specific preference $pref(d)$ of the worker d. The score $SCORE(d, s, t)$ of a task s for developer d at time t is determined based on weighted (and normalized) scores for exploration and exploitation as given in Equation 5:

$$\begin{aligned} SCORE(d,s,t) &= ((1 - pref(d)) * EARN'(d,s,t) \\ &+ pref(d) * LEARN'(d,s,t)) \\ &* pw(d,s,t) \end{aligned} \quad (5)$$

$EARN'(d, s, t)$ is the normalized score of $EARN(d, s, t)$, while $LEARN'(d, s, t)$ is the normalized score of $LEARN(d, s, t)$, The value of $EARN'(d, s, t)$, $LEARN'(d, s, t)$, $pw(d, s, t)$ and $pref(d)$ lies between [0,1]. The higher the value of $pref(d)$, the stronger is the preference towards learning (and vice versa). $pref(d)$ can be elicited from each worker with a recommendation tool. In this paper, we normalized each metric value (e.g., $EARN(d, s, t)$) score) to lie between 0 and 1 by using the following transformation:

$$Norm(x) = 1 - \frac{Max(x) - x}{Max(x) - Min(x)} \quad (6)$$

In Equation 6, $Min(x)$ and $Max(x)$ are the minimum and maximum of the observed values considering same type of metric values of all tasks currently available for recommendations for developer d. $EARN'(d, s, t)$ score will be zero for all ongoing and new tasks s for a newcomer worker. In this case, only $LEARN'(d, s, t))$ and $pw(d, s, t)$ will have impact in the ranking score $SCORE(d, s, t)$.

3.5. Problem formulation

Task recommendations are based on the computations described above. For each pair of $< d, t >$ of worker d and time t, we generate a set of up to 10 recommended tasks. These tasks are selected to be the best regarding the score defined in Equation 5 among all tasks available.

Our problem is dynamic in nature in the sense that new tasks are uploaded all the time, and some other tasks are getting finished. Consequently, a proper recommendation needs to be regularly updated. An illustrative example of this changing context is given in Figure 2. For a sample worker, recommended tasks are listed for three consecutive times t_1, t_2, and t_3. We observe that: (i) the number of recommended tasks are varying (ii) from one time to another, tasks might be added or deleted, and (iii) the preference order of the tasks might change as well between points in time.

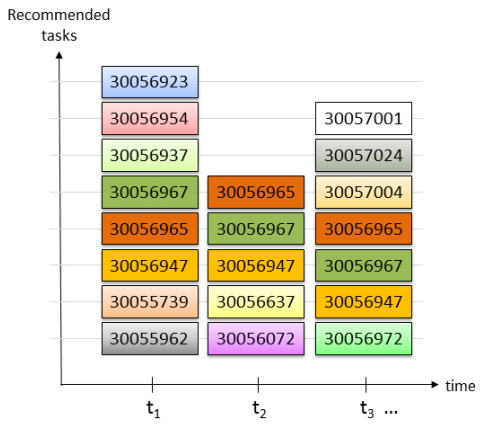

Figure 2. Illustrative example of changes in task recommendations in dependence of time (three sample days)

3.6. Solution approach

Our solution approach is based on two main assumptions that (i) for each worker, we know her preference values towards Learn and (ii) at any point in time t, we know the active workers. In total, the approach consists of six main steps:

Step 1: The system extracts all available completed tasks, associated worker registrations and results of the competitions from the crowd platform.
Step 2: The system identifies active crowd workers, ongoing and new tasks from the crowd platform.
Step 3: For each worker, the task recommendation system excludes all current and new tasks already registered by the worker. Unlike [8], the system only takes not yet registered tasks into account for recommendation.
Step 4: For each active worker, the system computes the learn score (Equation 4), earn score (Equation 1) for each ongoing and new task selected in Step 3. The machine learning component of the system also predicts the winning chance of the worker on each task (see Section 3.4 for further details).
Step 5: The system ranks the tasks for each worker based on Equation 5.
Step 6: The top 10 recommendations are presented to each worker.

In our current implementation of the prototype tool [13], the top 10 tasks can be emailed to each of the workers. This was also done for the developers who participated in our survey. The tool is dynamic in the way that it can re-run for recommendations after any reasonable time interval (e.g., every hour, every day, every week).

4. Empirical evaluation

4.1. Research questions

In the context of the dynamic task recommendation system, we investigate four research questions:

- RQ1 (Understanding): How do different groups of workers (defined by their tenure) behave regarding the Learn and Earn objectives?
  Why? This RQ serves as a justification for our research. From the analysis of real-world data, we want to understand the variation in the Learn and Earn behavior of different groups of workers
- RQ2 (Gain in technology coverage): Can the adoption of recommendation support the diversification of workers towards overall technology coverage?
  Why? This RQ measures the impact of the generated recommendations on improving participation in tasks requiring hot technologies.
- RQ3 (Usefulness of recommendations) How do actual crowd workers evaluate the recommendations?
  Why? External validation in an industrial context is a critical means to judge the applicability and usefulness of results.

4.2. Data collection

The whole empirical investigation done in this paper is based on real-world data from the Topcoder platform. Data collection was done for the period from January 01, 2015 to March 31, 2017. The crawler component of the developed prototype tool has retrieved the historical data from the Topcoder website with publicly available API. In total, we analyzed 4007 development type tasks and 14631 workers with 104222 registrations.

4.3. Study design

Analysis procedure to RQ1

In this analysis, we compare the Learn and Earn behavior of three group of workers separated based on their tenure. The analysis is conducted using 6-month data over October 1, 2016, to March 31, 2017. Out of the total 14631 workers, we initially considered only those who had at least one registration in one month of the 12-month period before March 31, 2017. This led to 5885 workers. To effectively demonstrate Earn behavior of workers, we need those workers who had

at least a few registrations in the last 6-month period. So, to answer RQ1, we considered only those who had at least one registration in three months of the 6-month period before March 31, 2017. This led to 357 workers. From our analysis, we noticed that the median and average number of task registrations per week (based on 357 workers) is 0.91 (almost 1) and 1.93, respectively (see [13] for more analysis). Significant number of development type tasks were posted on weekly basis (10 to 35 tasks) and monthly basis (65 to 110 tasks) over the analyzed six months.

Analysis procedure to RQ2

To answer this RQ, we generated recommendations for the same 5885 workers in the Topcoder platform on a weekly basis for the month of October 05, 2016 to March 28, 2017. For evaluation purpose, we used a metric called Technology Coverage gain which is defined as follows.

Definition: Technology coverage gain for a technology in a certain period is defined as the improvement in worker participation in that technology assuming each worker had adopted top ranked (rank 1) recommended task all the time. In other words, technology coverage gain is the percentage of workers in the platform working with tasks requiring that technology (following our task recommendation every week) minus the percentage of workers who worked with that. To compute this metric, we considered the actual worker registration for each task in a month and all the weekly generated recommendations in that month.

Analysis procedure to RQ3

To evaluate the usefulness of EX^2, we designed and conducted a user survey, consisting of four steps. First, in collaboration with TopCoder online developer community, an invitation letter was sent out to solicit participants. Second, our task recommendation system automatically generates Top-10 new recommendation for sign-up workers. Third, after reviewing the list of recommended tasks, each participant will complete a feedback survey about the usefulness of the guidance provided. Last, we analyzed the responses.

The user survey was conducted from March 25, 2017, to March 31, 2017. In this case, two Google forms (sign-up form and feedback form) were designed and used to collect data. For details of the forms, please refer to [13]. For the period of data collection, 14 workers have signed up to participate in the study, and we received 12 responses. Two of the workers were newcomers.

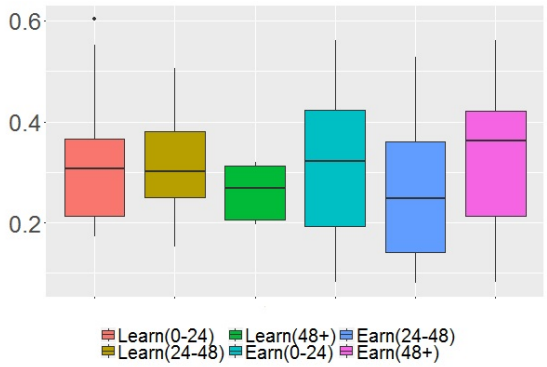

Figure 3. Tenure wise average monthly normalized Learn and Earn score for each group.

5. Empirical results

5.1. RQ1: Understanding the learning and earning behavior

In Figure 3, we show the monthly average Learn and Earn score for three groups of workers in box plots. Therein, grouping is based on the number of months (tenure) the workers are active in the Topcoder platform. More specifically, there are three worker groups, i.e. 1-24 months, 24-48 months, and more than 48 months, and each with 25, 25, and 307 workers respectively. The vertical axis show the calculated monthly average learn and earn scores ([0,1]).

It is evident from the figure that workers from all groups spend time on learning and exploration. Moreover, there is a decreasing trend in the average learning scores. This result is quite intuitive considering the different motivation patterns of the three groups. We noticed that even the workers who have been active for a long time (above 48 months) show both learning and earning behavior with huge variation in monthly average learn score. The monthly average Learn score ranges from 0.12 to 0.55 within the workers in that group, while monthly average Earn score ranges from 0.08 to 0.66. The monthly average Learn and Earn score for workers with 0-24-month tenure were in the range 0.17-0.60 and 0.08-0.56, respectively.

5.2. RQ2: Gain in technology coverage

In Figure 4, we report the overall hot technology coverage gain for a period of six months. This gain is defined as the sum of technology coverage gains for all defined level 1 hot technologies. In this experiment, we could not exactly measure who else were active in a particular month other than the workers registering on that month. To compute overall technology coverage,

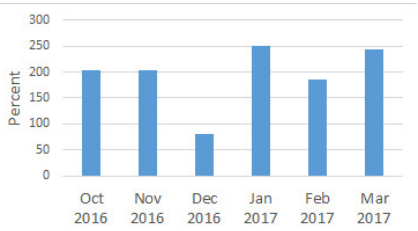

Figure 4. Total technology coverage gain after recommendation adoption for level 1 technologies.

Table 1. Percent of crowd workers having registered for hot technology tasks without (actual) and with (assumed following) recommendation

Technology	February 2017			March 2017		
	Without	With	Diff.	Without	With	Diff.
android	3.7	0.0	-3.7	8.5	8.0	-0.5
angular.js	26.8	69.9	43.1	18.2	76.2	58.0
api	14.0	12.1	-1.9	2.8	9.4	6.6
css	19.9	33.9	14.0	21.2	35.7	14.4
html	10.0	27.0	17.0	11.5	54.8	43.3
html5	15.9	29.8	13.9	16.0	27.2	11.2
ios	1.8	0.0	-1.8	0.0	0.0	0.0
java	27.6	49.1	21.5	20.9	54.6	33.7
javascript	29.3	45.6	16.2	24.8	68.0	43.1
jquery	5.2	1.6	-3.6	0.0	0.0	0.0
json	8.0	43.5	35.6	6.1	12.2	6.2
mongodb	0.2	12.2	12.1	0.0	0.0	0.0
node.js	23.2	47.6	24.4	16.3	42.8	26.6
Total			**186.8**			**242.6**

we made an assumption regarding active workers. We assume that a worker needs to have at least one registration history in the prior month or in the current month, to be considered as active. However, there can be more active workers in a platform than the workers registering in either of the two subsequent months.

Figure 4 shows that we achieved significant increase in the level 1 hot technologies regardless of the learning preference values of the workers. For November 2016, January 2017 and February 2017 when learning preference value was reasonably low (varied from 0.02 to 0.40), we observed 200% and above technology coverage gain for each month. Even when learning preference value was very low (0.02 or 0.05) for all workers in the platform (for December 2016 and March 2017), we had 80 to 250% gain, as through recommendations we could reach more workers than the workers who registered in tasks requiring those hot technologies.

In Table 1, we show the results regarding the increase or decrease in the number of workers after following recommendations for the very recent period of February and March, 2017. For most of the technologies, following recommendations results in a higher number of workers registering in these tasks, thereby reducing the probability of task cancellation due to qualified submissions.

When we performed the similar type of analysis assuming that 40% and 60% crowd workers (instead of all workers) accepted the top ranked (rank 1) recommended task, monthly average technology gain was around 20% and 90%, respectively. The more crowd worker accepts the top recommendation on a weekly basis, the higher the monthly average technology gain. In addition, the more active workers in a platform, the higher the chances of accepting the top recommendation. Interested readers can see [13] for more analysis on monthly technology gain.

5.3. RQ3: Usefulness of recommendations

According to feedback from the participated crowd workers, here is a summary of evaluation results:

1. 10 out of 12 (83.4%) believe that: It makes sense to employ the differentiation between exploitation and exploration in task recommendation. The other 2 responses are neutral. Average score: 4.25/5.
2. 7 out of 12 (58.3%) believe that: The Top-10 recommendations are useful for selecting up-coming tasks, with 2 neutral responses, and 2 negative responses. The average score is 3.41/5.
3. 9 out of 12 (75%) workers believe that: Projection of the winning chance supports my decision on task selection, with 1 neutral and 2 negative responses. The average score is 3.83/5.
4. 9 out of 12 (75%) believe that: A weekly update helps me to accommodate changes in the tasks available in the environment, with 2 neutrals and 1 negative responses. The average score is 3.9/5.
5. 7 out of 12 (58.3%) workers believe that: A daily update helps me to accommodate changes in the tasks available in the environment, with 2 neutral and 3 negative responses. Average score is 3.66/5.

The feedback to the two open ended questions provides much more informative suggestions on the numerical ratings. Here are some encouraging remarks regarding the usefulness of the recommendation system:

1. *"Suggested list matches my interests, so the recommended list is good."*
2. *"The recommended challenges help me keep updated with the technologies and the new trend."*
3. *"I have registered one challenge (the first one in the recommendation list) because I'm working on the Hercules XRE series challenges."*

4. *"Seems fit to me."*

Two workers provided negative responses for questions Q2 to Q4. In the two open end questions, one indicated that I register in almost every challenge to at least checkout whats going on even when not planning to submit, which suggests an additional task selecting pattern on gathering information. Additionally, few research directions were suggested by participants. such as reviewer position recommendation, and incorporating special community identity factor of TCO eligibility.

6. Discussion

6.1. Implications for workers and platform

A tremendous amount of variability accompanies individual performance on software crowdsourcing mini-tasks. To support dynamic worker decision making, it is essential to capture and benefit from appropriate manipulation of such variations. The proposed EX^2 method addresses the measurement of varying amount of technical requirements w.r.t. skill and abilities of individual workers using two scores, i.e. Earn score and Learn score.

Additionally, a common criticism faced by many recommendation systems is the bias towards experienced individuals because the lack of proper mechanism to handle newcomers with limited or no prior information in the learning history. Such difficulty can be inherently dealt with through the Learn (Exploration) dimension in the EX^2 method.

From platform perspectives, such preference-driven task recommendation is important for different reasons. In the short term, it ensures attraction of expert workers; and in the long run, it guides the healthy growth of resource pool w.r.t. the new and emerging technologies. Even though crowd workers try to acquire skills in new technologies, they may or may not pay attention to hot/ demanding technologies.

6.2. Threats to validity

Threats to Construct Validity: Learning preference value can take various values within the range 0 to 1. Also, different workers can also have different preferences over time. Thus, the chosen preference values might not be the representative values.

Technology heat map can vary over time. It usually does not drastically change over one year period, may take several years. In this study, rather than varying the map over time, we used the single technology heat map already determined by the Topcoder platform. However, in future, we will design metrics to periodically and dynamically re-evaluate hotness of various technologies based on market demand.

Threats to External Validity: The used data set is collected from Topcoder. The learning preference values, expertise and interest of the pool of crowd workers and tasks posted in the Topcoder platform can be quite different from those of other competitive platforms. Thus, the conclusion that we made might not generalize to other platforms, but the learn and earn based recommendation system and its impact generalize to other competitive crowdsourcing platforms.

7. Related work

7.1. Crowd worker motivation and behavior

Different studies on motivation patterns of crowdsourcing workers reported that monetary award is one of the top motivating factors to attract and involve potential workers in task competition in crowdsourcing market [1, 2, 14, 15, 6]. The award amount typically correlates the degree of task complexity and required competition levels as well as task priority in the project development [15, 6]. Learning is another primary motivation factor in CSD. For newcomers or learners, it takes the time to improve and turn into an active worker after their first arrival [6] Therefore, most of them focus on registering and gaining experience by competing with peers and learning from reviews. Kaufmann and Schulze [15] identified some additional motivational factors, such as community identity.

7.2. Resource optimization

Assigning human resource to software development tasks has been studied intensively [11]. The questions to be answered in general are "Who will work on what?" and "When to work on what?" A variety of techniques has been proposed for addressing these questions, primarily for proprietary software development. Finding the best assignment strategy in consideration of conflicting objectives have been studied in [16, 17, 18].

In CSD task assignment, existing studies focus on learning worker expertise and skills from history and match to factors or criteria extracted from task requirements. For example, Mao et al. proposed a content-based classification approach to recommend both winning workers and participatory workers for new tasks [19]. Zhu et al. employed conditional random field method to develop a learning to rank framework to recommend developers matching specific criteria such as skill and location in task requirements [20]. Abhinav et al. proposed a freelancer recommendation framework

for interdependent tasks based on multi-dimensional assessment metrics and various machine learning models [21]. The results of the above research provides decision support to competitive or hiring based platforms or task requesters in those platforms. Unlike former approaches, this work intends to assist crowd workers in decision making. The work in [8] suffers from starvation problem (workers with no past success likely not to receive any task recommendation) due to the employed ranking algorithm, while this work can recommend and rank tasks for any worker. Unlike [8], this approach employs binary classification with different set of features to predict winning chance of each worker on each task.

8. Conclusion and future work

Dynamic decision support in software crowdsourcing market is critical to meet individual worker preferences, ensure task success, as well as balance resource utilization. Existing methods overlook the role of workers learning preference in recommending which tasks to work on.

In this study, we proposed the EX^2 method that defines and utilizes Learn and Earn measures when quantifying and recommending target goals of developers on crowdsourcing tasks. The method was implemented as a dynamic task recommendation system, which can produce regular (e.g., near real-time, weekly, daily) recommendations. The proposed method and tool implementation are evaluated both on a historical dataset and through a user study with 14 crowd workers from the TopCoder online community. The evaluation results show very positive feedback.

Acknowledgment

This research was partially supported by the NSERC Discovery Grant 250343-12 and AITF scholarship.

References

[1] K. Lakhani, D. Garvin, and E. Lonstein, "Topcoder (a): Developing software through crowdsourcing," tech. rep., Harvard Business School, 2010.

[2] K. Mao, Y. Yang, M. Li, and M. Harman, "Pricing crowdsourcing-based software development tasks," in *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pp. 1205–1208, 2013.

[3] C. Bonner, *10 Burning Questions: Your Starting Guide to Open Innovation and Crowdsourcing Succes*. Topcoder, 2017.

[4] G. Ruhe and F. Bomarius, "Learning software organizations," in *Handbook of Software Engineering and Knowledge Engineering*, pp. 663–678, 2000.

[5] L. Muhdi and R. Boutellier, "Motivational factors affecting participation and contribution of members in two different swiss innovation communities," *International Journal of Innovation Management*, vol. 15, no. 03, pp. 543–562, 2011.

[6] S. Faridani, B. Hartmann, and P. G. Ipeirotis, "What's the right price? pricing tasks for finishing on time," in *Proceedings of AAAI 2011*, pp. 26–31, AAAI Press, 2011.

[7] M. M. Gianluca Baldassarre, *Intrinsically Motivated Learning in Natural and Artificial Systems*. 2013.

[8] Y. Yang, M. R. Karim, R. Saremi, and G. Ruhe, "Who should take this task?: Dynamic decision support for crowd workers," in *Proceedings of ESEM 2016*, pp. 8:1–8:10, ACM, 2016.

[9] R. Saremi, Y. Yang, G. Ruhe, and D. Messinger, "Leveraging crowdsourcing for team elasticity: An empirical evaluation at topcoder," in *Proceedings of ICSE-SEIP 2017*, pp. 103–112, 2017.

[10] "Apache lucene tfidf similarity." `https://lucene.apache.org/core/6_3_0/core/index.html?org/apache/lucene/search/similarities/TFIDFSimilarity.html`.

[11] "Topcoder technology heatmap." `http://crowdsourcing.topcoder.com/what-are-the-hottest-technologies-in-crowdsourcing/`.

[12] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, 2001.

[13] "Online access to data and instruments used in the study." `https://github.com/mrkcse/LearnEarnRecommendationTool`.

[14] Y. Yang and R. L. Saremi, "Award vs. worker behaviors in competitive crowdsourcing tasks.," in *Proceedings of ESEM 2015*, pp. 68–77, IEEE, 2015.

[15] N. Kaufmann, T. Schulze, and D. Veit, "More than fun and money. worker motivation in crowdsourcing: a study on mechanical turk," in *Proceedings of ACIS 2011*, pp. 1–11, 2011.

[16] C. Stylianou and A. Andreou, "Human resource allocation and scheduling for software project management," in *Software Project Management in a Changing World* (G. Ruhe and C. Wohlin, eds.), ch. 4, pp. 73–106, Springer, 2014.

[17] E. Alba and J. F. Chicano, "Software project management with gas," *Information Sciences*, vol. 177, no. 11, pp. 2380 – 2401, 2007.

[18] M. R. Karim, G. Ruhe, M. M. Rahman, V. Garousi, and T. Zimmermann, "An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs," *Journal of Software: Evolution and Process*, vol. 28, no. 12, pp. 1025–1060, 2016.

[19] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman, "Developer recommendation for crowdsourced software development tasks," in *Proceedings of SOSE 2015*, pp. 347–356, March 2015.

[20] J. Zhu, B. Shen, and F. Hu, "A learning to rank framework for developer recommendation in software crowdsourcing.," in *Proceedings of APSEC 2015*, pp. 285–292, IEEE Computer Society, 2015.

[21] K. Abhinav, A. Dubey, S. Jain, G. Virdi, A. Kass, and M. Mehta, "Crowdadvisor: A framework for freelancer assessment in online marketplace," in *Proceedings of ICSE-SEIP 2017*, pp. 93–102, IEEE Press, 2017.