Who Should Take This Task? – Dynamic Decision Support for Crowd Workers

Ye Yang
Stevens Inst. of Technology
1 Castle Point Ter
Hoboken, NJ 07030, USA
+1(201)216-8560
ye.yang@stevens.edu

Muhammad Rezaul Karim
University of Calgary
2500 University Drive NW
Calgary, Alberta T2N 1N4
+1 (403) 220 7692
mrkarim@ucalgary.ca

Razieh Saremi
Stevens Inst. of Technology
1 Castle Point Ter
Hoboken, NJ 07030, USA
+1(201)216-8560
rlotfali@stevens.edu

Guenther Ruhe
University of Calgary
2500 University Drive NW
Calgary, Alberta T2N 1N4
+1 (403) 220 7692
ruhe@ucalgary.ca

ABSTRACT

Context: The success of crowdsourced software development (CSD) depends on a large crowd of trustworthy software workers who are registering and submitting for their interested tasks in exchange of financial gains. Preliminary analysis on software worker behaviors reveals an alarming task-quitting rate of 82.9%.

Goal: The objective of this study is to empirically investigate worker decision factors and provide better decision support in order to improve the success and efficiency of CSD.

Method: We propose a novel problem formulation, DCW-DS, and an analytics-based decision support methodology to guide workers in acceptance of offered development tasks. DCS-DS is evaluated using more than one year's real-world data from TopCoder, the leading CSD platform.

Results: Applying Random Forest based machine learning with dynamic updates, we can predict a worker as being a likely quitter with 99% average precision and 99% average recall accuracy. Similarly, we achieved 78% average precision and 88% average recall for the worker winner class. For workers just following the top three task recommendations, we have shown that the average quitting rate goes down below 6%.

Conclusions: In total, the proposed method can be used to improve total success rate as well as reduce quitting rate of tasks performed.

CCS Concepts

• Software and its engineering → Software development process management • Information systems → Data analytics

Keywords

Crowdsourced software development; worker behaviors; dynamic decision making; submission rate; submission score; task-quitting.

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, 3]. The success of CSD relies heavily on a large crowd of trustworthy software workers who are registering and submitting for crowdsourced tasks in exchange of financial gains. More specifically, a general CSD process starts with task requesting companies distributing tasks with prizes online, and then crowd software 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/or document quality [1, 12]. The number of submissions and their evaluated scores reflect the level of success in task satisfaction or completion.

Designed to enable wide task accessibility and self-selection, most CSD platforms allow crowd developers to freely choose tasks to engage based on their personal skills, experience, and interests. This consequently results in two types of challenging issues: 1) manual task selection is very time consuming considering the large number of simultaneously available tasks; and 2) task requesters typically have very limited visibility and control over unknown workers. From the task requester's perspectives, it is challenging to identify best workers for their tasks, and even more challenging to monitor risks related to workers reliability shortfalls. While most CSD platforms employ certain trust or reputation system, the effectiveness of solely relying on such system to identify or filter workers is rather limited [8]. Inappropriate task-worker matching may harm the quality of software deliverables [11]. Some CSD decision support methods were proposed for task requesters in facilitating decisions related to software task pricing [3] and developer recommendation [11].

However, a more important issue in competitive CSD is that worker decisions are highly volatile from task registration to task submission, and may cause cascading effects on crowdsourcing failure. As the current baseline, the following observations are drawn from data from January 2014 to January 2015, extracted from TopCoder, the most popular software crowdsourcing platform [9]:

- High quitting rate. In this period, 50089 out of 60433 records of worker-task registration led to no submissions, which indicates a high task-quitting rate of 82.9%.
- Weak average submission quality. Among the 10344 submissions, 5777 (55.8%) has failed quality review.
- Non-trivial task failure rate. 769 out of the 4907 tasks (15.7%) were cancelled due to zero or failed submissions.

These alarming observations motivate the research presented in this paper. We aim at tackling the problem to improve these outcomes

ESEM '16, September 08-09, 2016, Ciudad Real, Spain
© 2016 ACM. ISBN 978-1-4503-4427-2/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2961111.2962594

from a different perspective, which is to support dynamic decision making of crowd workers. We report the results of an empirical study that investigated the influencing factors of crowd worker's behaviors in a competitive CSD context, introduce dynamic features extracted to characterize dynamic competition factors, and propose an analytics-based dynamic worker decision support framework using random forest learners. The results are the first step towards the development of a dynamic recommendation systems for crowd workers to make decisions on their best-matching tasks with high winning probability.

The paper is structured as follows: Section 2 introduces a motivating example; Section 3 summarizes related work; Section 4 presents the modeling of the task recommendation problem; Section 5 details the experimental study design; Section 6 reports empirical results; Section 7 discusses the results and threats to validity; and finally Section 8 provides conclusions and an outlook to future work.

2. MOTIVATING EXAMPLE

To better understand the basic decision scenario of CSD, we provide an illustrative example presented in Figure 1. The example includes information about task selection and completion of five tasks and among six workers. The three numbers along with each worker summarize the number of registered tasks, the number of submitted tasks, and the number of winning tasks, respectively, during the period from Jan 2014 to Dec 2015. For example, worker *Sunbinbrother*, besides registering for all five tasks shown in Figure 1, has totally registered for 4096 tasks, in that period, but did not submit for any of them (i.e. submission rate = 0). Another worker, *OlinaRuan*, in total registered for 162 tasks, made 14 submissions, and 6 wins. The two numbers under each task ID represent the numbers of registrants and submissions per task. The difference between them are the number of workers who only registered for that task, but not submitted.

For a given task, after the specified deadline, each submission to that task is reviewed and evaluated with a quality score. A CSD task is called *failed* if it does not receive at least one acceptable submission exceeding the minimum quality score. In Figure 1, successful tasks are colored in green, and failed tasks are colored in orange. For example, task #30041810 is failed due to zero submissions, even though its registered crowd of 41 workers indicates a very broad participation.

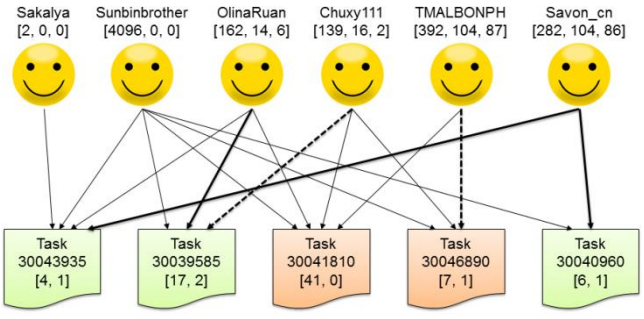

Figure 1. Motivating example to illustrate workers' behaviors in task selection and completion. Dotted, bold and dashed lines represent registration without any submission, winners, and submissions without winning, respectively.

As introduced earlier, the 82.9% task quitting rate and 55.8% failed submissions relate to a total CSD failure of 15.7%. To assure greater CSD success, it is essential to ask: Why do crowd workers quit tasks? From crowd workers' perspectives, even though they have the freedom in selecting tasks, many workers are trying to maximizing their utility in winning a competition. If a worker's perceived chance of winning a task is low, they may choose to quit a task [4]. There could be many imperfect information during this decision making of crowd workers that results in high task-quitting rate, for example:

Optimism bias in task selection. Software developers have a record of overestimating their productivity [15]. Most workers tend to be over-optimistic and select more tasks than what they can complete. The six workers shown in Figure 1 demonstrate different level of optimism in that all of them selected more tasks to register than those that they submitted. Such optimistic belief includes: what types of tasks they are interested, skill match to those required by a task, affordability in terms of time and effort required to complete a task, etc. The higher the optimism bias, the more likely a worker will change mind and quit the task later on.

Effort concentration in competition context. When facing time pressure and the need to cut down workload, workers tend to focus on those tasks that best match their personal interest and expertise. Table. 1 summarizes the competition history of worker *Savon_cn*. While most task types are self-explaining by names, a First2Finish task is a software development contest which stops when receiving the first satisfactory solution. These results reflect that *Savon_cn* have a broad scope of interests, and his personal submission focus (i.e. the last column) is only on 3 task types.

Perceived pressure from competition dynamics. Crowd workers may feel more stressed and anxious when competing for a task with more participants, especially in the presence of some strong competitors. For example, as shown in Figure 1, task #30043935 has four registrants, but only with one submission. The primary reason for the other three registrants quitting the task might be the perceived competition pressure received from worker *Savon_cn*. On the one hand, such perception is extremely subjective and highly depends on an individual worker' particular motivation or availability at that time. On the other hand, the task outcome would be jeopardized if crowd workers hold inaccurate or wrong perception about their chance of winning.

In this study, we aim at modeling the above factors in addressing the high quitting issues in CSD.

Table 1. Competition history summary of worker "savon_cn".

Task Type	#Reg. (%)	#Subm. (%)	Subm. ratio
First2Finish	185 (66%)	76 (73%)	**41.1%**
UI Prototyping	48 (17%)	0 (0%)	0
Code	27 (10%)	7 (7%)	**25.9%**
Assembly	21 (7%)	21 (20%)	**100%**
Specification	1 (0%)	0 (0%)	0
Total	282 (100%)	104 (100%)	36.9%

3. RELATED WORK

3.1 Crowd Worker Motivation and Behavior

Many studies have reported various motivational factors of crowd workers, which largely fall into two categories: intrinsic and extrinsic factors [7, 9, 16]. Kaufmann and Schulze [7] combined different models from classic motivation theory, work motivation theory, and Open Source Software Development model to crowdsourcing markets, and identified extrinsic factors, such as immediate payoffs, delayed payoffs, and social motivation, have

strong influence on a worker's time spending on crowdsourcing tasks.

Many software workers tend to optimize their utility of choosing the task based on different attributes and personal utility [2, 10]. For example, it is reported in [17] that a high ranking motivation factor is requesters with brand names, such as Google and NASA, which attracts software workers to apply for tasks, that potentially can be used in strengthening resumes and affects software worker ratings indirectly or directly.

For newcomers or beginners, it takes time to improve and turn into an active worker after their first arrival [2, 10]. Therefore, most of them focus on registering and gaining experience by competing with peers, but the chance of them winning the competition is rather low. Existing studies show that by passing time, registrants gaining more experience, hence better performance is expected and consequently higher score will be granted [13]. In addition, "Cheap Talk" [13] has been reported as a strategy referring for tough competitions, in which strong contestants strategically employ this strategy to deter additional competitor entry into the task.

3.2 Software Task Assignment

Assigning human resource to software development tasks has been studied intensively [18]. 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 and constraints related to time, effort or other dimensions have been studied. As one of the first attempts in that direction, evolutionary algorithms were introduced for that purpose by Chang et al. [19]. Alba and Chicano [20] moved ahead and employed genetic algorithms to assign resources to tasks while taking into account duration, resource skills, cost and global complexity. More recently, Karim et al. [21] studied the assignment of tasks (in that case, for bug fixing) to developers based on the match of expertise.

CSD is different from proprietary development in its process, level of control and objectives to be achieved. Scheduling aspects are of minor relevance as there are fixed time frames of delivery for all tasks. Allocation of tasks is replaced by the bidding process performed by crowd workers. The main flexibility is on the crowd workers to decide about registration and performance of tasks, in this paper, we provide decision support for developers to facilitate answering "Who Should Take This Task"?

3.3 Decision Making for Crowdsourcing

Online decision algorithms have a rich literature in operations research, economics, machine learning, and artificial intelligence, etc. Much of existing work on crowdsourcing decision making is addressing problems in the general crowdsourcing markets. For example, many studies have applied machine learning techniques in learning worker quality and optimizing task assignment decisions [5], aggregating individual answers to improve quality [6], and worker incentives [7]. Slivkins and Vaughan [22] identified a variety of modeling choices for repeated decision making in crowdsourcing, including five categories: task design, platform design, quality of work, incentives and human factors, and performance objectives. Karger et al. [23] introduced a task assignment model for classification tasks and proposed a non-adaptive assignment algorithm based on random graph generation. Singer and Mittal [24] proposed a dynamic procurement model for crowdsourcing in which workers are required to explicitly submit their preferences. Bernstein et al. [25] proposed to apply queuing theory in real-time crowdsourcing to predict the expected waiting time and cost of the decomposed uploaded tasks.

In software crowdsourcing, only a few studies have focused on decision support for software crowdsourcing market. Among them, most focus on supporting decision making from the perspectives of task requesters or crowdsourcing platforms. These studies include task pricing [3, 10], developer recommendations [11], and understanding worker behaviors [4, 12, 13, 14]. For example, Mao et al. [11] presented a content-based developer recommendation framework for CSD context, to recommend reliable workers based on static features extracted from participation history and winning history. However, there is no consideration on dynamic aspects of ongoing competitions at any given decision time. In addition, most of such models are from the perspectives of task requesters or crowdsourcing platforms, and there is a lack of research on decision support for crowd workers.

4. MODEL BUILDING AND PROBLEM FORMULATION

The objectives of this study is to support software worker's decisions in CSD, drawn from workers' past competition history. To that end, we propose to model incorporating both conceptual and dynamic aspects of worker decision process in CSD, and formulate a novel worker decision support problem towards the measurable goal of reducing overall task quitting rate.

4.1 Conceptual Worker Decision Model

In this study, we adapt the competitive CSD process model from [4] and extend the award-worker behavior model to elaborate on key decisions that crowd workers frequently need to make. More specifically, the CSD process starts with task requestor. We propose a static worker behavior model as illustrated in Figure 2.

In this model, we define four terms characterizing different types of software crowd workers: (i) a *Quitter* is a worker who did not make submissions to a task (s)he once signed up for, by the given deadline; (ii) a *Winner* is a worker who has submitted a piece of code for the stated task, and was evaluated as the winner or runner-up among all submissions; (iii) a *Submitter* is a worker who has submitted for a task but did not win the competition; and (iv) the Uninterested being workers who were active but did not register for a task.

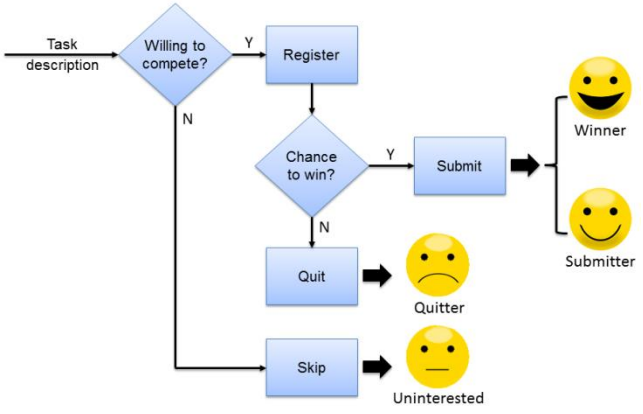

Figure 2. Conceptual worker decision model.

The two key decisions to be made of crowd workers, depicted as two diamonds in Figure 2, are:

1) **Willing to compete?** Decision on whether the worker is willing to compete on a task. Our previous work assumed that the task award and perceived required effort or skills to complete the task are two main factors to consider at this point. In this work, we propose to include more information from task description such as challenge type, technology, semantics in task requirements, in order to provide best matching tasks for active and interested workers, based on past competition history.

2) **Chance to win?** Decision on whether there is a chance to win a task. Our previous work drew light on the role of distracting factors that may lead workers change their minds to drop a once registered task. Such factors could be competition anxiety according to the Yerker-Dodson law [4, 28], unavailable time and resource, etc.

In this study, we employ the first three worker types as the labels on competitor outcomes that describe worker participation in CSD tasks. While the reasons for different workers to quit a competition may be complicated, there is a need to better understand software workers' dynamic decisions factors with respect to different specific stages of a task competition. More specifically, we need to better understand the context and characterize, for a specific worker, his/her current state of competition. This will be introduced in the next section.

4.2 Dynamic Worker Decision Model

To model the contextual factors for CSD competitions, assuming when a new task t_i is posted, we define the following competition attributes in order to characterize a worker w_j's previous competition history:

- $r_{i,j}$: the number of times w_j registered for tasks similar to t_i;
- $s_{i,j}$: the number of times w_j submitted for tasks similar to t_i;
- $v_{i,j}$: the number of times w_j won tasks similar to t_i.

Next, we define a set of dynamic features for worker w_j, based on these competition attributes.

4.2.1 Measuring Worker's Optimism Bias

Workers' participation history offers helpful information on the tendency of optimism bias in making task selection. Presumably, a worker with strong optimism bias tends to register more tasks that (s)he can possibly complete in a given timeframe, which forces him/her to quit a significant number of them due to time constraints. Therefore, we propose to measure such optimism bias in task selection through worker's submission rate.

Definition 1: For a given worker w_j, the worker's *submission rate* (SR_i) is defined as the ratio of number of submissions and number of registrations in a worker's competition history, i.e., all tasks the worker has competed on during a fixed time duration.

In this study, SR of worker w_j is defined by Eq.-1. This can also be used to indicate the relative degree of a worker's optimism bias in task registration. The higher the submission ratio is, the lower the optimism bias.

$$SR_j = (\sum_{j=1}^{m} s_{i,j}) / (\sum_{j=1}^{m} r_{i,j}) \text{ for all i} \qquad \text{Eq.-1}$$

4.2.2 Measuring Worker's Effort Concentration

Learning from competition history can lead to a better understanding and modeling of worker's personal submission effort concentration.

Definition 2: *Effort Concentration (EC)* is measuring the submission focus of a worker w_j using her average submission ratio on similar tasks. For all pairs of task t_i and worker w_j, effort concentration $EC_{i,j}$ is calculated according to Eq.-2 as the number of submissions divided by the number of registrations across the set of similar tasks. Details on task similarity analysis will be discussed in Section 5.4.

$$EC_{i,j} = s_{i,j}/r_{i,j} \text{ for all i,j} \qquad \text{Eq.-2}$$

Definition 3: *Submission Quality (SQ)* is defined as the worker's average score of her submissions (if $s_{i,j} > 0$) on a task's previous similar tasks. Suppose for worker w_j, we can calculate her average score of $s_{i,j}$ across all previous submissions on similar tasks. The detailed definition is shown in Eq.-3:

$$SQ_{i,j} = \text{Average score of } s_{i,j} \text{ submissions , if } s_{i,j} > 0 \text{ and } SQ_{i,j} = 0 \text{ otherwise} \qquad \text{Eq.-3}$$

4.2.3 Measuring Current Competition Status

Definition 4: *Competition Status* of a task consist of a group of time-related metrics to measure worker's competition preferences during his most recent competition history w.r.t. the past T days. Therein, where T represents the number of days to look back from any given current day. These dynamic metrics include:

- NumTask: the number of currently registered but not yet submitted tasks of worker wj at any given time;
- NumRegTasksTDays: the number of registered tasks in the last T days;
- NumSubTasksTDays: the number of submitted tasks in the last T days;
- NumWinTasksTDays: the number of won tasks in the last T days;
- AvgPrice: the average price of the registered tasks in the last T days;
- CompetitorFactors: For each worker, we also derived competitor related metrics based on the above dynamic metrics by taking average of the top Y competitors over historical data collected over the last T days. In this study, we set Y equals to 5 to compare with a worker's top 5 competitors.

4.3 Problem Formulation

We propose to model the problem of dynamic worker decision in CSD context as a single-label 3-value (i.e., winner, submitter, and quitter) classification problem as well as ranking relevant tasks to each worker. Each interaction of a worker with a task on the CSD platform is modelled as an instance in a chronically ordered stream of competition behaviors, for training and testing purpose. We define for each of the instance a set of dynamic features describing the status of the worker's behavior. The label on each instance is assigned by observing the competition outcome of each worker-task relationship.

In total, we define the dynamic crowd worker decision support problem **DCW- DS**:

With a given track record of workers and their task performance, predict whether (i) the worker is going to be a winner/submitter/quitter for a given task, and (ii) on a daily basis, recommend for each developer the top three tasks having highest winning chance?

5. STUDY DESIGN

To empirically investigate the machine learning based solution approach for the DCW-DS problem, we design three research questions and conduct experiments using real-world data collected

over a period of more than one year. Figure 3 summarizes the steps and research questions associated with the study conducted.

In the following subsections, details are presented related to the studied research questions, actual dataset, the data pre-processing steps, similarity analysis, classification of the workers using Random Forest RF machine learning technique, ranking of the tasks for workers and metrics selected for performance evaluation.

5.1 Research Questions

To investigate the dynamic crowd worker decision support problem **DCW-DS**, the following research questions (RQs) were formulated and studied in this paper:

RQ1: *What are the Top-10 impact factors for a worker to be a quitter, winner or submitter?* This RQ is designed to measure the relative importance of various attributes on the classification of a worker, specifically to see the most influential Top-10 impact factors.

RQ2: *How does the classification results vary in dependence of usage of dynamic versus using just static features?* This RQ is designed to investigate whether the use of dynamic features improves the quality of prediction of a *winner/submitter/quitter* worker compared to using just static features.

RQ3: *What is the potential effectiveness of the proposed method in reducing task quitting rate?* This RQ is designed to investigate whether the use of the most promising task recommendations per worker will reduce quitting rate and by how much.

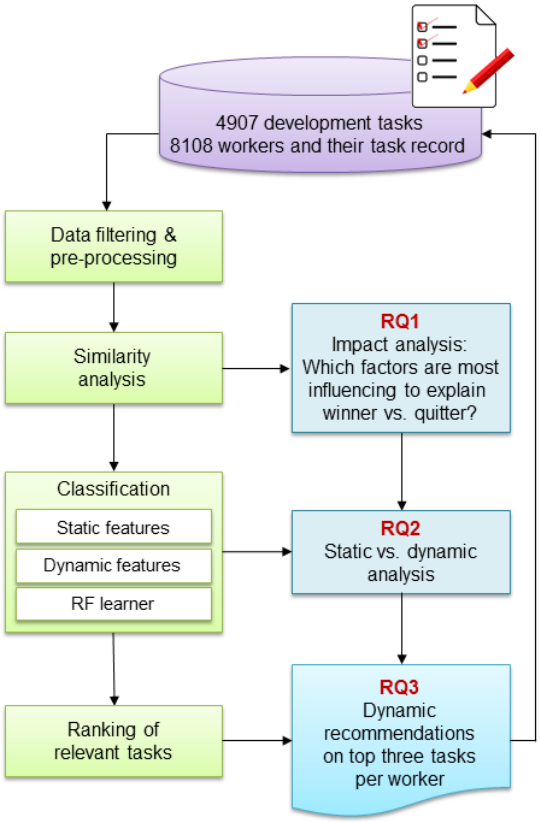

Figure 3. Main flow of the proposed framework and relationship to research questions.

5.2 Dataset

The dataset investigated in our study is extracted from the TopCoder website. It contains the following information:

Task metadata. The dataset contains 4907 competitive software development tasks posted during the time frame from January 22, 2014 to March 9, 2015. The tasks across a diverse range of types including design, UI prototyping, coding, assembly, bug hunting, etc. The task attributes include Name, Prize, StartDate, EndDate, Type, Platform, Technologies, and detailed requirements.

Worker metadata. The dataset includes metadata information about 8108 workers having been active during the given timeframe. The worker attributes are name, country, year joining the community, community rating, and overall earning.

Competition history. The dataset includes 60433 records of worker-task registration and submission relationship, i.e., RegisteredOnly and Submitted. Each relationship captures the dates a worker participated in a task by registering and by submitting to a task. If it is a RegisteredOnly relationship, then the submission date is NULL, as illustrated at Table 2.

Table 2. Example competition history of worker savon_cn during the week of 07/01/14 to 07/05/14.

Worker	Task ID	Registration Date	Submission Date
savon_cn	30043741	7/1/14 21:52	NULL
savon_cn	30043845	7/1/14 21:11	NULL
savon_cn	30043935	7/5/14 20:52	7/5/14 20:52

Competition results. Results of a competition include submission status, the scores for each submission, and their corresponding rank in the task competition. The scores were produced through multiple rounds of peer reviews organized by TopCoder. Submission status denotes whether the worker was the winner or just submitter for the task. There are cases that a task corresponds to multiple submitters but no winner, which means all submissions failed review and the task has failed. Table 3 below shows an example task competition results in the dataset.

Table 3. Example competition results of task #30043741.

Task ID	Worker	Score	Rank	Status
30043741	sin_hu	85.44	1	Active
30043741	gmagniez	74.94	2	Failed Review
30043741	NoRKin	72.83	3	Failed Review

In this study, the selected static data attributes are summarized in Table 4. Additionally, the extraction and preparation of time-based dynamic features used in this study will be discussed in Section 5.5.

Table 4. Summary of selected static data attributes.

Feature	Format	Description
Task ID	Numeric	The unique identity of the task.
Task Prize	Numeric	The amount money to the winner(s).
Registration Start Date	Date/Time	The date and time the task competition is open.
Submission End Date	Date/Time	The date and time the task submission is closed.
Task Type	Text	Type of task competition.

Feature	Format	Description
Technology	Text	Required technologies by the task.
Platform	Text	Required technologies by the task.
Detailed Requirements	Text	Detailed text information of task requirements.
Worker name	Text	The unique name of the worker.
Worker's registration date	Date/Time	The date and time the worker registering for the task.
Worker's submission date	Date/Time	The date and time the worker submitting for the task.

5.3 Data Filtering and Preprocessing

Data cleaning is the first step. We filter those historical tasks according to the following criteria:

1) Tasks with incomplete information: for example, some tasks were cancelled before their submission deadline. This step removes 190 such tasks.
2) Unusual tasks: to exclude irrelevant empirical knowledge, we remove the unusual tasks that are apparently different from the majority of CSD tasks. Such tasks are typically associated with extremely high prices, i.e., typically greater than $5000 in total prize. They are organized in either Hackathon or Mashathon challenges, or fixed-duration, multi-winner grand challenges. For example, there is a 30-day debugger challenge with a total prize of 100,000, including 3296 registered workers, and 1253 winners. We excluded 8 tasks (1 Mashathan, 4 Hackathon, and 3 grand challenges) priced above $5,000.
3) Inactive worker: as discussed earlier, if a worker never submitted to any task, he is treated as inactive and filtered out.

Next, we convert each text feature into word vector format, keeping only meaningful and descriptive tokens processed by tokenizing and stop word removal. Suppose there are m terms, then the corresponding vector v_t would be $v_t = (w_{t,1}, w_{t,2}, \ldots, w_{t,m})$, where $w_{t,j}$ stands for the weight for each $term_j$. The weights are calculated as *Term Frequency Inverse Document Frequency* (TF-IDF) [26].

5.4 Similarity Analysis

For each new task, a starting point of extracting dynamic features is to identify a set of similar tasks from the past, and then extract the current worker's dynamic features based on their past performance on the similar task set.

Definition 4: Similarity $Sim_{i,j}$, between two tasks t_i and t_j is defined (see Eq.-4) as the weighted sum of all local similarities across the features listed in Table 4:

$$Sim_{i,j} = w_1*Dist_1(t_i,t_j) + w_2*Dist_2(t_i,t_j) + \ldots + w_n*Dist_n(t_i,t_j) \quad \text{Eq.-4}$$

Where $Dist_j$ indicates the local distance function and w_j stands for the weight assigned to the corresponding attributes. In our study, we used 7 similarity attributes, as summarized in Table 5, and assume equal weights. We treat the 7 task distance measures in Table 5 equally important, but not the dynamic features derived from this component.

When extracting dynamic features, for simplification reason, we set a similarity threshold value of 0.8 in order to only include the most similar tasks. Then, at any given time, for each worker-task pair in our dataset, we calculate the worker's dynamics features based on the most similar tasks, according to the definitions in Section 4.2.

Table 5. Feature distance measures used.

Feature	Description of distance measure $Dist_i$
Task Prize	(Prizei - Prizej) = PrizeMax
Registration Start Date	(Datei - Datej) = DateMaxDiff
Submission End Date	(Datei - Datej) = DateMaxDiff
Task Type	Typei == Typej?1 : 0
Technology	Match (Techi; Techj) = NumberOfTechsMax
Platform	PLi == PLj?1 : 0
Detailed Requirements	(Reqi*Reqj)/| Reqi |*| Reqj|

5.5 Classification Using Random Forest Prediction

In our study, we built a separate prediction model using Random Forest [30] algorithm on each day for the period between Jan 22, 2014 and Mar 9, 2015. Random Forest (RF) is an ensemble of classifiers and was found to outperform other classifiers in many other applications in the software engineering and machine learning literature [31]. Even though we used RF, any other prediction algorithm that assigns probability scores for each class label (e.g., Naïve Bayes, Support Vector Machine) can be used as prediction model. A comparative analysis was outside the scope of this paper and will be an aspect of future work.

Before building a model for a specified day, for each sample (definition given below), the static and dynamic features, as defined in Section 4.2, were extracted. Dynamic features were derived for a historical data of past T days. In this study, we have selected three configurations of T, i.e., 60 days, 90 days, and 120 days respectively.

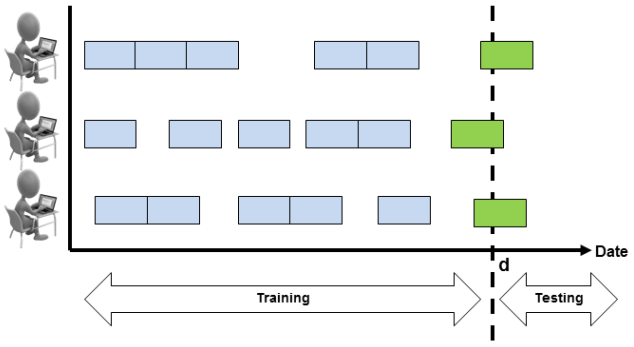

Figure 4. Illustration of selecting training set and testing set.

Before building and evaluating a model on a particular day *d*, we created two sets of samples of data called Training set (TR) and Testing set (TS). This is illustrated in Figure 4. TR contains all samples for training the model, while TS contains all test samples of ongoing tasks. Each sample in the set TR represents information (static and dynamic features related to the relevant task and the worker) for a developer-task pair where the developer actually registered for a task and the submission deadline was earlier than the current date *d*.

Similarly, each sample in the sets TS contains same kind of information but for all the tasks where the submission deadline is beyond the date d. The task's registration open dates for the test samples can be no later than date d. We also labeled each sample in the training set TR as winner, quitter, submitter based on the definition given in section 4.1. Once training and test samples were created, we predicted the label for each sample in the test set TS. For each sample, we also extracted the probability scores for each class label (winner, quitter, and submitter).

The WEKA machine and data mining library [30] was used for building and evaluating the models. The predictive modeling experiments conducted were performed with 124 features (excluding class variable), including 14 dynamic features and 110 static features. Among the static features, 107 binary features encoded the required technologies (e.g. css, html5, java) of the task. The rest of the static features were task duration, task total prize and overall submission rate of the worker. The whole dataset used in this study, including task attributes, worker attributes, and extracted features, is posted in a Github repository [27].

5.6 Ranking of Relevant Tasks for Workers

For each worker from the test set TS, we ranked the relevant tasks. To come up with the ranking, we first identified the test samples belonging to each worker. Then ranked the identified samples in descending order of the workers' winner label probability score and put them in a list.

We discarded the samples with low winner probability score (i.e. less than 0.33) as well as where the winner probability score is less than the submitter probability score. Next sorted the same identified samples in descending order of their submitter label probability score. In this case, we filtered samples based on a probability threshold (i.e. less than 0.33) as well as when submitter probability score is less than the winner probability score. Then appended the remaining samples in the tail of the previously constructed list if already not added. The constructed lists contained ranked tasks for each worker with tasks with high winning chance followed by high submission chance. For some workers, especially with workers with no winning history, the constructed list contains ranked tasks with high submission chance only. Our hypothesis is that this kind of task ranking for workers can reduce task quitting rate of the worker in the recommended tasks.

5.7 Metrics for Performance Evaluation

For the goal of evaluating the quality of our predictions, we have defined four metrics to evaluate the accuracy of predictions:

Definition 5: *Precision* describes the percentage of samples of correctly predicted quitter (or winner or submitter).

Definition 6: *Recall* describes the percentage of samples of the quitter (or winner or submitter) class in the predicted results, out of all the samples that are quitter (or winner or submitter, respectively).

Definition 7: *F-measure* is the harmonic mean of precision and recall and combines these two measures into one.

Definition 8: *Quitting rate (@Top 3)* is the average of the number of quitted tasks out of the Top 3 recommended tasks for all workers.

6. EMPIRICAL RESULTS

Following the DCW-DS study design presented in Section 5, we conducted our empirical analysis. In this section, we report the results for answering the three research questions.

6.1 What are the Top-10 impact factors for a worker to be a quitter, winner or submitter (RQ1)?

For this RQ, we applied Chi-Squared [29] and Information Gain [29] attribute evaluation algorithms to determine the impact of the static and dynamic features. The former evaluates the importance of an attribute by computing chi-squared statistics with respect to the class variable, while the later evaluates the impact by measuring the information gain with respect to the class variable.

Table 6 shows the Top10 ranking of the features extracted on the entire data set. The final ranking, shown in the last column, is determined by taking average of the rankings gained from Information Gain [29] and Chi-Squared algorithms [29]. We can see from the table that dynamic features like SQ, EC, CompetitorFactors-SubRate, number of won/submitted/ registered tasks in last T days, and AvgPrice are among the Top10 attributes.

We also found several static attributes are among the Top10 attributes like SR, TotalPrize, and Task duration. Among the competitor related attributes, average of average success rate for the top Y competitors was ranked fourth. These results indicate that dynamic features of the worker as well his/her competitors' dynamic features have high impact in determining the class label for a worker on a particular task.

6.2 How classification results vary in dependence of usage of dynamic features (RQ2)?

To answer RQ2, we compared the classification and ranking performance of the Random Forest classifier with and without using the dynamic features. When we considered only static features for building predictive model, for each training and test sample only one feature was related the worker (overall submission rate). The other features were related to the task which are task duration, task total prize and technology related features. When we

Table 6. Ranking of Top-10 features impacting predictions.

Rank	Attribute Name	Type	Info Gain Rank	Chi Squared Rank	Average of ranks
1	SQ (average submission quality on similar tasks)	Dynamic	1	1	1
2	SR (Overall submission rate)	Static	2	2	2
3	EC (average submission rate on similar tasks)	Dynamic	3	3	3
4	CompetitorFactors-SubRate (average of average submission rate For Top Y Competitors)	Dynamic	4	4	4
5	NumWinTasksTDays (number of won tasks in last *T* days)	Dynamic	5	5	5
6	NumSubTasksTDays (number of submitted tasks in last *T* days)	Dynamic	6	6	6
7	AvgPrice (average price of registered tasks in last *T* days)	Dynamic	9	7	8
8	TotalPrize (task total prize won)	Static	8	8	8
9	NumRegTasksTDays (number of registered tasks in last *T* days)	Dynamic	7	10	8.5
10	Task Duration	Static	10	9	9.5

added dynamic attributes with the static attributes, we had more attributes related to the worker as well as his/her competitors.

For the classification performance, we compared results in terms of average precision, average recall and average F-measure computed over consecutive 30 days period. Results are shown in Table 7 for 30 days period starting at September 01, 2014. For each day, we used the dedicated training set generated for that day to build two predictive models (with and without dynamic features) and tested on the dedicated test set for that day. The dynamic features were computed considering the historical data of the last 90 days (T=90).

For both the quitter and winner models, we achieved very high precisions, recalls and F-measure values. Random Forest algorithm could predict a worker as quitter with around 99% average precision and average recall with the addition of all dynamic attributes with the static attributes. Based on the large presence of the quitter class samples in the training data, it is quite expected that the models perform very well in detecting quitter class samples.

Table 7. Performance evaluation on 30 consecutive days with dynamic and without dynamic features in terms of average Precision (P), average Recall (R) and average F-Measure (F).

Feature type	Quitter			Winner			Submitter		
	P	R	F	P	R	F	P	R	F
Dynamic	.99	.99	.99	.78	.88	.82	.80	.54	.64
Static	.88	.93	.90	.43	.27	.33	.51	.29	.31

If we look at the results for the winner class, we notice that a worker can be labelled as a potential winner with around 78% average precision and 88% average recall when all the dynamic attributes are added. For the submitter class, both models have high average precision but low average recall.

For other values of T, we also achieved almost similar precision, recall and F-measure values for all classes (results not shown) when dynamic attributes were added with the static attributes. To make sure that sufficient past data (registration, submission and winning information for each worker) is available for computing the dynamic feature values and also to use relatively recent data, minimum value of T was chosen as 60 days and maximum value of T was chosen as 120 days, with 30 days interval.

If we compare the results of the models with and without dynamic attributes, we can see that addition of dynamic attributes significantly improves the performance in terms precision, recall and F-measure for all classes (see Table 7 and Figure 5). Figure 5 shows by how much the addition of dynamic attributes improves precision, recall and F-measure values over the same consecutive 30 days period for the winner class with T=90. For the winner class, the daily performance improvement lies between 10% and 80%.

To compare the performance of predictive algorithms with and without dynamic features, we also conducted non-parametric statistical test with the results obtained with three different values of T (T=60, T=90, T=120) for the same period of 30 days of analysis. Since variances were observed in daily performance for each metric, statistical tests were necessary to make fair comparison. Nonparametric statistical methods were chosen in this study as they do not make any rigid assumptions regarding how values in the population are distributed.

For each performance metric (precision/recall/F-measure), we conducted *Mann-Whitney U* test to compare the performance with and without dynamic features obtained for a particular value of T with $p < 0.001$. When the difference was statistically significantly different, we computed *Vargha-Delaney* effect size measure value to measure the probability that the addition of dynamic features improve the performance of predictive model when static features are only used. For this measure, probability above 0.5 means addition of dynamic features improves performance, probability below 0.5 means dynamic features leads to worse performance, equal otherwise. With our *Vargha-Delaney* effect size comparison, we noticed that dynamic features improves precision with 95% probability, recall with 94% probability and F-measure with 96% probability. For each performance metric (e.g. precision), we took the average of the probability for 9 comparisons (3 classes multiplied by 3 configuration of T).

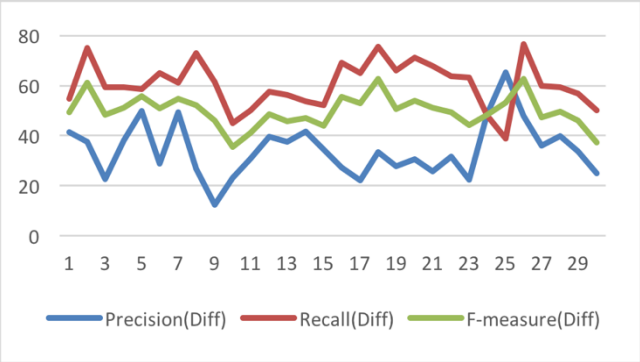

Figure 5. Improvements (%) achieved with dynamic features for the winner class (compared to just using static features) over 30-day period.

6.3 Can we recommend the most relevant tasks to each worker (RQ3)?

To answer this RQ, we rank the recommended tasks per worker on a daily basis and average the success of the workers in terms of their quitting rate in top ranked three tasks. Figure 6 shows the average (taken over 30 days) of the quitting rate of all workers on the recommended Top 3 tasks on 30 consecutive days starting at September 1, 2014. When the data over last 60 days (T=60) was used to compute the dynamic features, we observed mean quitting rate of 3.56% with a standard deviation of 4.78%. For the other values of T, the mean and standard deviation was almost same (around 5.4 and 7.0 respectively). From our analysis, it is evident that the ranking approach can significantly reduce the quitting rate and propose relevant tasks each day for any values of T.

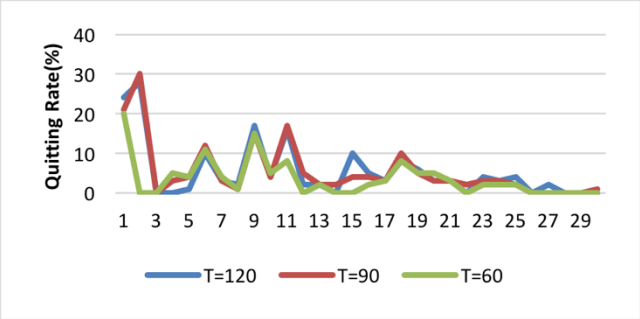

Figure 6. Average worker quitting rate of tasks recommended. Results are compared for varying training sets (durations).

For this RQ, we also conducted a *Kruskal-Wallis* test (with $p < 0.001$) to see whether any significance difference exist in the daily quitting rate when we change the value of T. From our statistical test, we did not observe any statistically significance difference.

7. DISCUSSION

7.1 Implications to CSD workers

In the dynamic worker decision model, the overall goal is to come up with a daily-based ranking of current open competitions, for each worker. Results in RQ1 shows that, learning from worker's competition history, the proposed dynamic features on worker's overall submission rate, the most concentrated type of tasks, and the most frequent competitors, help to improve the performance in recommending the most relevant tasks to him with high winning chances, in another words, low quitting rate. Leveraging Random Forest, the proposed DCW-DS method can accurately predict the tasks on which the developer is going to be a potential submitter or winner. These kind of rankings and predictions can reduce task quitting rate if the proposed model is utilized by the developers before registration in a task and as well as increase the probability that best skilled workers are submitting.

Among the already registered workers for a task, the model can provide decision support to two specific types of workers, as discussed in [11], on a daily basis: the workers who registered for the task with strong desire to win, and the workers who registered just to submit and get feedback on their work. The former type of workers can quit if they are not predicted as winner, while the workers in the second type can discontinue if our model predicts them as potential quitter.

Though it appears that the proposed DCW-DS framework might cause bias to veterans of CSD platform instead of encouraging new or less experienced developers, actually it is not prescribing anything and the final decision is still with the workers. Moreover, it is the responsibility of CSD platforms to employ such decision support as well as mechanisms in balancing wide participation and deliverable quality between veteran and novices. For example, TopCoder provides a special venue for new comer workers to experience and train their competition skills.

The potential impact of such prediction capability is substantial. Currently most decision support studies on CSD are from task requesters or platform perspectives, and the workers are provided with no facilitation, and hence mostly based on gut-feelings in selecting tasks and deciding when to stop. Incomplete information about the competition situation esp. the biased pressure from competitors may lead to unwise quitting decision, such as "Cheap Talk" phenomena [13], which may be avoided by a more analytics-based decision support as the proposed framework. We are currently collaborating with TopCoder on further evaluating the proposed method, and the feedback of the potential value is very encouraging.

7.2 Implications to task requestors

Though it is not the focus of this paper, the proposed dynamic worker decision model offers a number of practical insights for task requestors:

i. It may be used to recommend best workers for particular tasks based on worker dynamics at the point of time. The output of the framework can be configured to produces a ranked list of registered workers for each task in terms of their winning likelihood;
ii. The modeling and output of the competition pressure provides a dynamic monitoring of the evolving competition situation, associated with potential winner/submitter/quitter projection;
iii. More transparent competition helps to retain solid workers and consequently aid in preventing uninformed quitting and reducing task failure risk;
iv. The quitter prediction may be employed as dynamic qualification screening criteria to identify and filter out unreliable workers which are frequently classified as "Quitter".

7.3 Threats to Validity

There are a number of threats to validity. First, the proposed method does not consider worker's multi-tasking factors, i.e., how many tasks can a worker take, at maximum, during the same period of time. Though this is one of the important factors influencing software project scheduling decisions, we don't see there is a pattern in the upper bound limits for worker's multi-tasking.

Second, the data preparation, feature extraction and analysis is complicated by the temporal nature of the worker activities and competition outcomes. Different weights and threshold setting, different selection of temporal window may lead to different grouping of the instance data, as well as the calculated data on dynamic features. To avoid this potential threat, we configured DCW-DS to accommodate different temporal window settings (e.g. 60, 90, and 120 days) to select the training dataset. However, the validity of the results may be impacted by adjusting some simplification assumptions including the similarity weights and threshold value in Section 5.4.

Third, the data sets used in this study for both training and testing are mostly unbalanced data sets in terms of the number of samples belonging to different classes: quitter, winner and submitter. The issue is not addressed in DCW-DS and will be considered in our future work.

8. CONCLUSIONS

In this paper, we proposed a novel problem formulation, DCW-DS, and introduced an analytics-based decision support methodology to guide dynamic decision making of crowd workers in the CSD context. The proposed method was evaluated using real-world data from TopCoder. Compared with the baseline 82.9% task-quitting rate, the results imply that such kind of dynamic decision support for crowd workers is critical towards achieving an increased submission rate and reduced failure rate due to no or poor submissions in current CSD market.

The following extensions are planned as future work: (i) focus on winning rate by filtering and collecting the right amount of data; (ii) to provide decision support from requesters perspective to recommend best-matching workers based on dynamic competition status; and (iii) comparative analysis of RF with other predictors.

ACKNOWLEDGEMENTS

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada, NSERC Discovery Grant 250343-12. The authors thank the reviewers for insightful comments to early version of the paper, and appreciate the access to data and support provide by TopCoder's Dave Messinger. One of the authors was supported by a grant from Alberta Innovates Technology Future.

9. REFERENCES

[1] K. R. Lakhani, D. A. Garvin, and E. Lonstein, "TopCoder (A): Developing Software through Crowdsourcing," Harvard Business School Case 610-032, Jan. 2010.
[2] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton, "The Future of Crowd Work," in Proc. CSCW 2013, pp. 1301–1318.

[3] K. Mao, Y. Yang, M. Li, and M. Harman, “Pricing Crowdsourcing-based Software Development Tasks,” Piscataway, NJ, USA, 2013, pp. 1205–1208.
[4] Y. Yang and R. Saremi, "Award vs. Worker Behaviors in Competitive Crowdsourcing Tasks," ESEM 2015, pp. 1-10.
[5] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. Movellan. “Whose vote should count more: Optimal integration of labels from labelers of unknown expertise,” Advances in Neural Information Processing Systems. 22(2035-2043):7–13. 2009.
[6] A. Mao, A.D. Procaccia, and Y. Chen, “Better human computation through principled voting,” in Proc.of the AAAI Conference on Artificial Intelligence 2013, pp. 1142-1148.
[7] N. Kaufmann, T. Schulze, and D. Veit, “More than fun and money. Worker Motivation in Crowdsourcing - A Study on Mechanical Turk”, Proc. 17th AMCIS, 2011.
[8] A. Mao, E Kamar, and E. Horvitz, “Why Stop Now? Predicting Worker Engagement in Online Crowdsourcing,” Proc. HCOMP 2013.
[9] T. D. LaToza et al., “Microtask programming: Building software with a crowd.” In Proc. Symp. UI Software and Technology, 2014.
[10] S. Faradani, B. Hartmann, and P.G. Ipeirotis, "What's the Right Price? Pricing Tasks for Finishing on Time", In Proc. Human Computation, 2011.
[11] K. Mao, Y. Yang, Q. Wang, Y. Jia, M. Harman, “Developer Recommendation for Crowdsourced Software Development Tasks,” SOSE 2015: pp 347-356.
[12] TopCoder website: “10 Burning Questions on Crowdsourcing: Your starting guide to open innovation and crowdsourcing success,” https://www.topcoder.com/blog/10-burning-questions-on-crowdsourcing-and-open-innovation/, Access date: March 14, 2016.
[13] N. Archak. “Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on topcoder.com”, In Proc. Conference on World Wide Web, pages 21–30, 2010.
[14] H. Zhang, Y. Wu, W. Wu. Analyzing Developer Behavior and Community Structure in Software Crowdsourcing. Information Science and Application. Vol. 339. Pp981-988.
[15] M. Jorgensen and S. Grimstad. Over-Optimism in Software Development Projects: "The Winner’s Curse". In Proc. 15th International Conference on Electronics,
[16] M. & Marsella, S. C. (2014). Encode Theory of Mind in Character Design for Pedagogical Interactive Narrative. Advances in HCI, vol. 2014, Article ID 386928.
[17] J. Yang, L.A. Adamic, and M.S. Ackerman, “Crowdsourcing and knowledge sharing: strategic user behaviour on tasks,” In: Proc. ACM conference on Electronic Commerce, pp 246-255.
[18] C. Stylianou, and A. S. Andreou, “Human Resource Allocation and Scheduling for Software Project Management,” In: Software Project Management in a Changing World (G. Ruhe, C. Wohlin, eds.), Springer 2014.
[19] C.K. Chang, C. Chao, S. Hsieh, “SPMNet: a formal methodology for software management,” in Proc. COMPAC, November, 1994.
[20] E. Alba, J.F. Chicano, “Software project management with GAs,” Journal of Information Science, 177(11):2380- 2401, 2007.
[21] M. R. Karim et al., "An Empirical Investigation of Single-objective and Multi-Objective Evolutionary Algorithms for Developer’s Assignment to Bugs," to appear in Journal of Software: Evolution and Process, 2016.
[22] A. Slivkins and J. W. Vaughan. Online Decision Making in Crowdsourcing Markets: Theoretical Challenges. ACM SIGecom Exchanges, Vol. 12, 2013, pp 4–23
[23] D. Karger, S. Oh, and D. Shah, “Iterative learning for reliable crowdsourcing systems,” In 25th Advances in Neural Information Processing Systems. 2011.
[24] Y. Singer, and M. Mittal, “Pricing mechanisms for crowdsourcing markets,” Proc. Intl. WWWConf. 2013.
[25] M. S. Bernstein et al., “Analytic Methods for Optimizing Real-time Crowdsourcing”, CS.SI 2012, 1204.2995
[26] G. Salton, M.G. McGill. Introduction to modern information retrieval. McGraw-Hill. (1986).
[27] Github repository “TopCoder-Winner-Quitter”: https://github.com/yy2111/TopCoder_Winner_Quitter. Access date: March 15, 2016.
[28] P. A. Hancock, H. C. Ganey, “From the inverted-U to the extended-U: The evolution of a law of psychology,” J. Human Performance in Extreme Environments, 2013, pp 5–14.
[29] Y. Yang and J. O. Pedersen, “A Comparative Study on Feature Selection in Text Categorization,” in Proc. Conference on ML, Morgan Kaufmann Publishers. pp. 412-420, 1997.
[30] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, “The WEKA data mining software: an update,” SIGKDD Explorations Newsletter 11(1), 10–18. Nov 2009.
[31] S. Lessmann et al., “Benchmarking classification models for software defect prediction: a proposed framework and novel findings,” IEEE TSE, vol. 34, no. 4, pp. 485-496, 2008.
[32] A. Dwarakanath, N.C. Shrikanth, K. Abhinav, A. Kass, “Trustworthiness in enterprise crowdsourcing: a taxonomy & evidence from data,” Proc. ICSE 2016 pp 41-50.