Dynamic Crowdsourcing Consensus Tasks with Workers That Can Learn

by

Shengying Pan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2016

© Shengying Pan 2016

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Shengying Pan

Abstract

Crowdsourcing has become one of the most popular topics in both academia and industry in the past few years. By hiring workers online, task assigners can take advantage of the wisdom of the crowd and solve problems that used to only be solvable by experts or that are too hard for automated computer algorithms. Consensus tasks are some of the most common tasks in the crowdsourcing domain, e.g. labeling, classification, pattern recognition, and etc. To solve consensus tasks, a group of workers are hired to report possible outcomes, and their opinions are aggregated to produce a final prediction.

When working with consensus tasks, crowd workers' quality information is always valuable for system designers to find better workers and to devise opinion aggregation approaches that can produce predictions with higher accuracy. Traditional approaches used in many research literatures either assume every worker has a fixed quality, or the worker population as a whole has a fixed quality. They calculate worker quality from historical data, assume it stays as is and use it for estimating existing workers' future performance and future workers' starting performance. We argue this is not true, especially for more complicated tasks where workers may need time and experience to fathom the idea and become adequate. We think the learning curve model used in factories and plants to model traditional workers' productivity and quality can be adopted to work in a crowdsourcing context, where workers quality should improve too as they work on more tasks and become experienced. We modified the hyperbolic learning curve and setup a sleep spindle detection experiment on Amazon Mechanical Turk to validate our argument. The results proved our hypothesis that crowd workers do achieve higher quality as they work on more tasks, and different workers have different quality and learning speed. Moreover, our linear regression test shows that hyperbolic learning curve is a good fit to model crowd workers' learning process. This gives us a more accurate approach to estimate and project crowd workers' quality.

With the help of learning curve model, we design a new dynamic hiring system that can make smart tradeoffs between workers' current quality and future potential, as well as the accuracy of aggregated prediction and hiring cost. It can take advantage of crowd workers' quality improvement, find, train, stay with high potential workers, and reduce the number of hired workers when some of the top performers stand out. Our system models the hiring process as a Markov Decision Process, and uses Monte Carlo Planning to find the best hiring path which optimizes the utility of both prediction accuracy and worker training. We compared our proposed hiring mechanism against other common methods such as randomly selecting workers and a bandit based approach which lets all workers work on a small training set first and picks the best performing ones. From results in both sleep spindle detection tasks and simulated tasks, our proposed dynamic hiring system yields a competitive or better performance with significant saving in hiring cost.

Moreover, to handle situations where crowd workers may not always be available and they are not free to hire to work on tutorial tasks, we extend our dynamic hiring mechanism to incorporate randomization so the system can choose between utilizing high quality workers and exploring unknown workers. From our simulation tests, the extended hiring system works as expected and can save even more in hiring cost while providing high quality predictions. Our dynamic hiring system especially outperforms traditional hiring approaches when the worker population is mixed where some workers have higher availability or learning speed. Overall, our dynamic hiring system is always able to find and take advantage of high potential workers and make hiring decisions adaptively. Our work opens up a new direction for hiring mechanism design in crowdsourcing settings where the usage of learning curves enable us to distinguish crowd workers and to model them more accurately.

Acknowledgements

I would first thank my supervisor, Professor Kate Larson. She is the perfect supervisor one can ask for, kind, supportive, and give me the freedom to work on something I really enjoy. I would also like to thank the readers of my thesis, Professor Edith Law and Professor Robin Cohen, for their constructive and valuable feedback. Moreover, I would like to thank Josh Bradshaw for his work on implementing and running the sleep spindle detection tasks on Amazon Mechanical Turk for me.

Dedication

To my parents who always support me and believe in me, I could never achieve what I have done without you in my life. You are the most important ones to me, forever. I also want to thank all my good friends who filled the non-academic part of my master's years with cheer and joy.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past few years, crowdsourcing has not only become a very popular topic in academic research, but also been adopted by the market and industry. With the power of crowdsourcing, companies and individuals are no longer limited by their internal resources, and can take advantage of the knowledge, energy, and creativity of a global, online and offline community, to solve problems that are comprised of tedious tasks, or not suitable to machines and algorithms.

Crowdsourcing can be applied to aid a vast range of activities. There are many different types of crowdsourcing projects online that utilizes crowd workers to collect geographical data (e.g. OpenStreetMap), create graphical designs (e.g. 99designs), help scientific research (e.g. Zooniverse), and so on. Additionally, there are general purpose crowdsourcing platforms like Amazon Mechanical Turk that enables companies and individuals to outsource specific errands to crowd workers of different backgrounds to help them complete tasks that are too hard or inefficient for computers.

Among all the crowdsourcing applications and projects, many involve hiring workers to perform repetitive tasks. For instance, on Zooniverse [1], the crowd workers are asked to classify sequences of photos of galaxies. When multiple workers are needed to predict answers to a set of problems, from the task assigner's perspective, he/she wants to hire as few workers as possible to save on the cost while still have a good enough aggregated performance. As a result, he/she would like to employ high quality workers that are more efficient or accurate at doing the tasks.

Some research has been done to tackle this problem [25] [42], by designing mechanisms that get workers' quality information and only hire workers at the top of the quality ranking. However, such mechanisms assume that the quality of workers is fixed, while in reality, it can be improved by training and learning from doing more work. For example, workers in factories and production plants become more productive as their work experience increases, crowd workers' quality and efficiency can also improve as they get more experienced and adequate after completing more tasks. Moreover, crowd workers are individuals who may come from very different backgrounds. They may possess different initial quality and ability to learn. The mechanisms that only select the top workers from a small set of data may end up with mediocre workers and ignore new coming workers with high learning potential, resulting in a solution which is not optimal in the long run.

In this thesis, we propose a new way to model crowd workers' quality and learning process with a hyperbolic learning curve model, which has been used in traditional industry such as manufacturing plants to model factory workers' quality and productivity for many decades [5]. We argue that crowd workers follow a similar pattern when doing complex consensus tasks that require high level understanding of abstract ideas such as sleep spindle detection tasks, and by using learning curves, we can not only estimate their quality more accurately, but also be able to project their quality improvement in the future. With the new model, we then devise a dynamic hiring mechanism which takes advantage of workerss learning potential and quality improvement, and make smart tradeoffs to save hiring cost in the long run while providing good performance in predicting correct answers. The thesis makes the following contributions as listed in the section below.

1.1 Contributions

As the topic of crowdsourcing is becoming more and more popular, the problems associated with how to hire crowd workers and aggregate their opinions when crowdsourcing consensus tasks have been studied actively in the domain of artificial intelligence in the past few years [7]. For example, many researchers have studied ways to estimate workers' quality and use collected information to improve the performance of their hiring mechanisms. They adopted machine learning, planning methods in artificial intelligence to solve crowdsourcing tasks and designed decision theoretic approaches with tools such as Markov Decision Process and Multi-armed bandit [23] [26] [24] [25] [42]. In this thesis, we created a new method to model crowd workers' quality and learning process. Unlike traditional approaches which assume either the entire worker population has a fixed quality or every individual worker has a fixed quality, our proposed hyperbolic learning curve model is able to parameterize each crowd worker's quality and learning speed. Thus it gives us the ability to estimate each worker's quality separately and more accurately, predict their future quality improvement, and distinguish fast learning workers from their peers. The validity and applicability of this learning curve model is verified by both sleep spindle detection experiments on Amazon Mechanical Turk (MTurk) and by computer simulation.

Second, with the new learning process model, we devised a dynamic hiring mechanism which models the hiring process as a Markov Decision Process. It uses Monte Carlo Planning to search all possible hiring outcomes and find hiring decisions that optimize the task assigner's utility by making smart compromises between prediction accuracy and hiring cost. Unlike other hiring mechanism, our new approach is able to find not only high quality workers but also high potential workers, get them trained, take advantage of their quality improvement and reduce the hiring cost in the long run. From experiments on both on MTurk and by simulation, our proposed hiring mechanism can save up to 50% in hiring cost while providing competitive or even better performance on prediction accuracy.

Last but not least, to handle more realistic crowdsourcing settings where crowd workers' availability is uncertain and unknown, and training crowd workers with sample tasks are not free, we extended our hiring mechanism to add randomness to the decision making process. The extended dynamic hiring mechanism makes a tradeoff between staying with existing workers and exploring new workers based on its knowledge of the learnt worker population and workers' quality information. It is able to take advantage of high availability workers and high potential workers while partially exploring the worker population, and saves even more on hiring cost. It especially yields better performance than other hiring mechanisms when the worker population is mixed with workers of different characteristics.

1.2 Thesis Outline

Besides the introductory chapter, the rest of this thesis is organized as follows:

In Chapter 2, we give background knowledge on crowdsourcing, consensus tasks, and the learning curve model. We also talk about problems we are interested in solving, as well as related research in this area working on similar problems with different approaches.

In Chapter 3, we introduce our learning model that uses a hyperbolic learning curve to model each crowd worker's learning process. We use a linear regression approach to find the parameters for each individual worker. We also provide results from sleep spindle detection tasks on MTurk and simulation tests to validate our proposed model, as well as discussing associated problems and challenges we discovered from these tests and how to handle them.

In Chapter 4, we present our dynamic hiring system that models the consensus task hiring process as a Markov Decision Process (MDP). We show all the details of the MDP and the algorithms we used for Monte Carlo sampling and worker ranking. The performance of our proposed hiring system is tested against MTurk data and simulation tests and the results are discussed and analyzed.

In Chapter 5, we extended our dynamic hiring system to work in settings where crowd workers are not always available and we can only run tutorial tasks with workers that are hired. We added randomization to our hiring mechanism so the system can spend some of its hiring cost to explore new workers. The extended hiring system is able to take advantage of high availability workers and high potential workers. We use simulation tests to verify its performance against different types of worker population.

In Chapter 6, we conclude our work and suggest future research directions to expand our proposed model and mechanisms.

Chapter 2

Background

In this chapter we first give an overview of crowdsourcing problems in general, and then focus on elucidating the subset of crowdsourcing tasks we are interested in. We also give an introduction to learning curves, which is the foundation of the method we use to model crowd workers' learning potential and process. We talk about related work in this area tackling similar problems and their limitations in this chapter, too.

2.1 Crowdsourcing Consensus Tasks

Crowdsourcing is a relatively new concept, and there are many varying definitions of it from different sources. E. Estellés-Arolas *et al* have done a thorough analysis of existing definitions to extract common elements and basic characteristics of crowdsourcing initiative [16]. In general, it is a participative activity in which a task assigner proposes to a group of individuals to undertake some tasks. The participants receive rewards (monetarily or virtually) while the task assigner utilizes their effort and benefit from their participation.

The domain and applications of crowdsourcing are broad. It spans from business, to academia, science, and forth [45] [7] [19] [14]. Organizations use crowdsourcing to gather knowledge from the general public to create an online encyclopedia e.g. Wikipedia, and companies such as Instacart use crowd workers to deliver groceries door-to-door to their customers. Each of these applications faces its own problems and challenges. In this thesis we focus on a subset of crowsourcing problems called consensus tasks.

A task is classified as a consensus task if its goal is to identify a correct answer that is not known to the task owner and there exists a population of workers who can make predictions about the correct answer [23]. To put it formally, for any consensus task t, there exists a set of possible answers A for t, and a correct answer $\hat{a} \in A$. Each hired crowd worker w reports a predicted answer $a_w \in A$, and the task owner aggregates all the reported answers to produce a final prediction $\bar{a}$, which is the consensus of the workers. The prediction is correct if $\bar{a} = \hat{a}$, and incorrect otherwise. There is a cost c_w associated with hiring worker w, and the task owner may have a budget that limits number of workers he can hire. Generally speaking, the task owner wants to get a good prediction that costs as little as possible.

Consensus tasks have many applications in the real world where the task assigner wants to identify, tag, label, and classify data which is too abstract and complex to be processed and analyzed by a computer algorithm. Pattern recognition tasks, for example, are more fitted to human brains and hard for computers to perceive. Projects such as Galaxy Zoo hires non-expert workers to classify celestial objects [29]. Online crowdsourcing platforms like MTurk [21] gives companies and individuals the ability to hire low cost crowd workers and take advantage of the wisdom of the crowd [40], for example, classifying objects into different categories, labeling venues from pictures, and identifying sleep spindles by reading electroencephalographic recordings from polysomnography [46]. Many research has shown that the performance of crowd workers is oftentimes better than using computer algorithms. For example, in Warby's paper, crowd workrers are shown to outperform all existing sleep spindle detection algorithms [46].

Crowdsourcing consensus tasks generally consists of two major parts: how to hire workers and how to aggregate their opinions. For the latter, research has shown combining multiple opinions often produce an improved performance of forecasting and labeling tasks in many domains [12] [39] [10]. There are many different methods to aggregate opinions, such as majority voting, opinion polling, and quality based Bayesian Network approaches [11] [9] [47] [22]. In this thesis, as we assume that we have reliable estimation of workers' quality which is the probability they are making correct/incorrect prediction, and each worker's answer is independent from others, this gives us the possibility to use Bayesian Network to aggregate reported opinions and the details can be found in Chapter 4 when we introduce our dynamic hiring system.

The hiring process, is the more complicated facet of this task. From the task assigner's perspective, he wants a good compromise between the quality of the aggregated consensus opinion and the cost of hiring workers. As he may have a lot of or even an unlimited amount of tasks to post, the hiring expense in the long run can be an important constraint to him. For solving every single consensus task, the task assigner needs to make two decisions: how many workers to hire and which workers to hire. Generally speaking, he wants to spend his hiring budget as little as possible while still maintaining a high confidence in the accuracy of the aggregated opinion.

Empirical study has shown not only that the expected accuracy of the aggregate answer increases as the number of hired agents increase, but also selecting a small group of top quality agents yields more accurate results than randomly selected agents [8]. Thus, hiring a proper amount of top performing agents can make the crowdsourcing task more efficient by both providing high quality aggregate answer and helping the task assigner to save on the hiring cost. However, the quality information of the workers is often unknown to the task assigner at the beginning in crowdsourcing setting. As a result, the designed mechanism must first gather such information before it can make any intelligent decisions. Long Tran-Thanh *et al* [42] proposed an approach using bounded a multi-armed bandit model that first spends portion of task assigner's budget to sample the quality of available workers by hiring them to do a small subset of tasks. After quality information is collected, they then use the leftover budget to hire workers based on observed quality ranking. They consider the hiring process as a bounded knapsack problem and solved it with a bounded greedy approximation algorithm. Although their hiring process targeted a different type of crowdsourcing tasks where the goal was to spend all the budget to hire software developers to create as many features for software project as possible, their approach to sample agents' quality is an interesting attempt to solve the problem of observing quality information.

On the other hand, Ece Kamar *et al* [23] [26] [25] have been working on solving crowdsourcing consensus tasks with a dynamic hiring process. They use machine learning to study agents' behaviour doing consensus tasks on Galaxy Zoo, and use their learned model to predict a potential new agent's behaviour given a classification task and the information the system has collected from previously hired agents. They model the hiring process as a Markov Decision Process where the rewards are the system's belief in the correctness of the aggregate prediction from the hired workers minus the cost to hire those workers. They use Monte Carlo Planning to approximate the optimal decision path from sampling. However, their approach faces the challenge that you must have significant amount of historical data to train the prediction model, thus it is not applicable to most task assigners where they are starting afresh. Moreover, their prediction model may not even work for tasks not on Galaxy Zoo as they are assuming there are certain relationships between tasks, which may not fit settings where tasks are independent. Furthermore, they assume workers have a somewhat fixed quality, e.g. if historical data shows workers are 80% correct in one setting, then a new worker will be 80% correct in the same setting. We argue this is not always true as crowd workers should have different quality. Their usage of MDP and relating system's belief in the aggregate answer to its utility however can still be adopted to design other hiring processes.

There are also other approaches that can reduce task a assigner's expense when crowdsourcing consensus tasks. One is to use a hierarchical crowdsourcing approach to combine automatic detection/classification algorithms with crowd workers' manual effort. Rahman *et al* [36] [37] have designed a mechanism to identify photos of weeds by first running an identification algorithm and dynamically make the hiring decision based on the system's confidence in the result of the algorithm. By replacing some of the crowd workers with a no cost automatic algorithm, it can reduce the number of workers needed and improve the utility of task assigners by saving on hiring cost.

However, all these approaches are oversimplifying the crowd worker population by assuming a worker's quality can either be estimated by averaging his past performance, or be predicted from other agents' (in this thesis we use worker and agent interchangeably) past behavioural history. It is quite unrealistic in crowdsourcing settings where agents can come and go and many agents may not even have prior knowledge of the tasks they are working on. When agents can learn from completing more tasks, there is a significant discrepancy between their quality in the past, their current quality, and their future quality. This discrepancy is even more significant between an averaged quality from the entire worker population that have already completed lots of tasks, and a new worker starting from scratch. Thus some of the approaches discussed above may face the problem of getting stuck with local high quality workers and ignoring high potential fast learning agents without prior experience that can quickly surpass their peers and help to improve the task assigners' utility in the long run. Moreover, their estimation of agents' quality may not be accurate enough as they are taking the average value of agents' learning process. This will impact the accuracy of the confidence and belief level their systems have upon the aggregate answer from hired workers.

The goal of our work is to propose a new model to mathematically represent crowd workers and their learning process. Thus we can have a more accurate estimation of workers' quality, and be able to model workers of different learning behaviours. We want to design a mechanism that can dynamically hire crowd workers and help task assigners to make smart tradeoffs between hiring current high quality workers and workers have potential to learn quickly, so they are able to train a pool of high quality workers efficiently and improve their utility in the long run when they are crowdsourcing a long horizon of consensus tasks. To achieve this goal, we want to introduce the concept of learning curve, which is the foundation of our crowd worker learning process model and one of the key components we build our dynamic hiring process upon.

2.2 Learning Curve

One of the features the consensus crowdsourcing tasks share is they hire workers to perform repetitions of manual-based tasks. Past study in many industrial sectors has shown that the quality of workers improve as they complete repetitive tasks [3] [33] [35] [2] [43]. There are multiple factors that may impact the workers' learning process, and the learning curve has proven to be an efficient tool to monitor such performance improvement, by providing means of mathematical models to analyze it.

Crowdsourcing consensus tasks are essentially the same as these production tasks. Crowd workers are hired to use their cognitive and analytical skills to perform repetitive tasks. For instance, a science enthusiast classifies many photos of galaxies on Galaxy Zoo. His efficiency and accuracy increases as he completes more tasks and gain more experience. It's just that their work place has been moved from production plant and factory to an online platform. It is natural to migrate knowledge we have on learning curve in industrial settings to crowdsourcing settings.

In M. J. Anzanello *et al*'s literature review [5], they covered most relevant models of multiple learning curves and discussed their mathematical aspects and applications. By definition, a learning curve is a mathematical description of workers' performance in repetitive tasks [6] [17]. There are a few different ways to measure workers' performance, and many different variables are used in learning curve models. One group of learning curves where they measure a worker's quality by percentage of non-conforming units in his overall production [41] [18] is the most interesting one to us. It can be directly translated to our crowdsourcing setting where non-conforming units are essentially incorrect answers a crowd worker has reported, and vice versa.

In this group there are many learning curve models proposed by researchers along the years, including univariate and multivariate models of varying complexity [5]. Among the univariate models the log-linear, exponential and hyperbolic models are the best known. In this thesis we mainly focus on the hyperbolic learning curve as it fits the best to our interests. The hyperbolic learning curve model was proposed by Mazur and Hastie [31]. It relates the number of conforming units to the total number of units produced. In their paper they provided both 2-parameter and 3-parameter forms. We are more interested in the 3-parameter form since it supports prior knowledge which is important in a crowdsourcing setting where workers have different starting quality (some workers may be smarter than their peers or understand the tasks better). The 3-parameter hyperbolic learning curve looks like the following:

$$y = k\left(\frac{x+p}{x+p+r}\right)$$

where y is the number of items produced in x units of operation time, k is the maximum performance level (its range depends on the definition of performance), p is a worker's prior experience, and r is the learning rate (time to achieve half of the maximum performance level k) [31] [34], where p and r are units of operation time, too (x, p, and r are all greater than 0). We will further tweak this model to fit our setting of consensus tasks in next chapter when we define our learning process model.

It is shown that the 3-parameter hyperbolic model of learning curve presents better fit, outperform many other models in terms of efficiency, stability, parsimony, and is more robust in many scenarios [31] [34] [4]. Also, its ability to model learning speed, prior experience, and complexity of work is a perfect fit to crowdsourcing consensus tasks. Essentially, for different domains of problems we are crowdsourcing, we should run tests and compare different learning curve models to check for the fit based on the coefficient of determination. However, in this thesis we want to provide a generic approach to hire crowd workers dynamically with knowledge of the underlying learning process. Our later proposed hiring process, although is designed based on the 3-parameter hyperbolic model, can be easily modified to cater other learning curve models if they are shown to be better fits to a certain selection of workers and problems.

In next chapter, we will introduce the modified 3-parameter learning curve for our learning model, the linear regression approach we use to learn the parameters, and present results from sleep spindle detection tasks on MTurk to verify its validity. We will also use simulation tests to discuss the problems and challenges our proposed model faces.

Chapter 3

The Learning Model

As we discussed briefly in previous chapter, the traditional way to measure a worker's quality is to divide his correct answers by his total number of answers reported. However, in crowdsourcing scenarios, workers have very different background and prior knowledge, and are asked to perform repetitive tasks. A worker's quality is not a constant variable and improve as he becomes more experienced. By adopting the 3-parameter hyperbolic learning curve [31], we propose a model where a worker w's cumulative quality is formulated by function 3.1:

$$Q_w(x) = \frac{x + p_w}{x + p_w + r_w} \tag{3.1}$$

where x is the number of tasks w has worked on, p_w is w's prior knowledge about the tasks, and r_w is w's learning speed defined as the number of tasks w needs to perform to achieve a $\frac{1}{2}$ cumulative quality, where cumulative quality is defined as the percentage of tasks w has completed correctly up to x-th task. In 3.1, the units of all x, p_w, and r_w are number of tasks (≥ 0). In another word, p measures a worker's starting quality: assuming his initial skill, understanding, or past experience of the tasks equals to having completed p_w tasks. Clearly, w has a greater initial quality if he had a greater prior knowledge p_w, and is faster to achieve a higher cumulative quality with a smaller r_w, vice versa. With $Q_w(x)$ we can derive w's actual quality $q_w(x)$ at task x for $x = 1, 2, 3...$ as

$$q_w(x) = xQ_w(x) - (x-1)Q_w(x-1) \tag{3.2}$$

When dealing with crowd workers who can learn, if we are estimating an worker's quality by using the percentage of his correct answers out of all answers, what we are getting is actually his cumulative quality. And the difference between one's cumulative quality (average performance from the beginning to current task) and actual quality (spot performance at current task) is:

$$|Q_w(x) - q_w(x)| = (x-1)(Q_w(x) + Q_w(x-1))$$

To illustrate this difference, for a worker with a prior knowledge of 5 and is working on tasks that take him 10 rounds to achieve 50% cumulative quality, the discrepancy between his cumulative quality function and quality function is about 20% at the widest gap, and significant cross the span of tasks (around 10% even after completing 200 tasks), as shown in the following graph (the curve on top is $q(x)$ and the other is $Q(x)$.

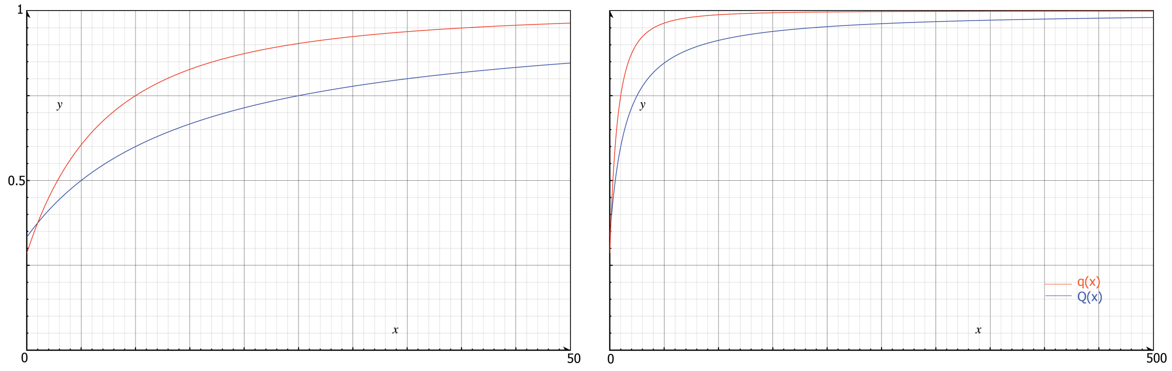

Figure 3.1: Compare $Q(x)$ and $q(x)$

Take this example, if we estimate this worker's quality by using his cumulative quality at 10th task, we may think his quality is 60%, while it is actually 75%. If our hiring system underestimates his quality like this, it may make the decision to hire more workers to maintain the quality of the final prediction. In fact, in this example, after completing the first task, $Q(x)$ is always underestimating $q(x)$. Thus in scenarios where agents are improving, assuming they have a fixed value of quality will add significant amount of error to ones designed hiring process. Our proposed 3-parameter learning curve on the other hand will make a better fit and provide more accurate estimation of workers' quality. We will validate this statement in a later section in this chapter. Now we need to first explain how we estimate the parameters in this model.

3.1 Learning A Worker's Quality Curve

As we defined crowd workers' cumulative quality functions at 3.1, we can learn it from collecting data of workers' completed tasks and the numbers of tasks they answered correctly, since it is available to the system as workers' past performance data. With collected data, we can use regression analysis to estimate parameters in ones cumulative quality function (CQF). As an agent's CQF is in hyperbolic form, we may first transform it into a linear function so that it becomes a linear regression problem and we can use more readily available tools. Letting $Z(x) = \frac{1}{1-Q(x)}$, then we can rewrite 3.1 as

$$Q(x) = \frac{x+p}{x+p+r} = 1 - \frac{r}{x+p+r}$$

$$1 - Q(x) = \frac{r}{x+p+r}$$

$$\frac{1}{1-Q(x)} = \frac{x+p+r}{r}$$

thus

$$Z(x) = \alpha x + \beta \tag{3.3}$$

where $Z(x) = \frac{1}{1-Q(x)}$, $\alpha = \frac{1}{r}$ and $\beta = \frac{p}{r} + 1$. We can then use linear regression analysis (we use scipy.stats.linregress in Python) to estimate α and β. In statistics, linear regression is an approach for modelling the relationship between a a scalar dependent variable and one or more explanatory variables. Note that the linear regression is giving the least square fit in $\frac{1}{1-Q(x)}$ not $Q(x)$. And the data our system collect is for $Q(x)$, so we need to transform it to $\frac{1}{1-Q(x)}$, too. Furthermore, the error term ϵ for $Z(x)$ at $Z(x) = \alpha x + \beta + \epsilon$ is in fact $Q(x) = \frac{x+p+r\epsilon}{x+p+r+r\epsilon}$ for $Q(x)$. We can then reproduce $Q(x)$ by deriving r and p from α and β. Once we have the estimated $Q(x)$ we can project an agent's future cumulative quality, as well as his immediate quality information $q(x)$ at task x by using 3.2.

However, one may ask here if our learning curve model and the linear regression approach works in the real world. Although there is an obvious connection between crowdsourcing consensus tasks and traditional production tasks in factories and plants, we are interested in exploring the connection using a concrete example in order to verify whether our assumption that crowd workers do actually learn and improve with experience. We found sleep spindle detection tasks to be a good domain for our experiments [46]. The tasks are not so difficult that crowd workers are unable to provide reasonably accurate answers, but at the same time the tasks are challenging enough that one would expect to see improvement over time if workers did actually learn with experience.

3.2 Sleep Spindle Detection Tasks

First we briefly go over the definition and features of sleep spindles. As our focus is more on the learning process of workers, one can refer to Warby's paper [46] for more details. Generally speaking, sleep spindles are discrete, intermittent patterns of brain activity that arises as a result of interactions of several circuits in the brain. Sleep spindles are measured by electroencephalography (EEG) as brief distinct bursts of activity in the sigma frequency range (11 - 16 Hz). They have a characteristic waxing and waning shape and are a key EEG feature used during sleep scoring to define non-rapid eye movement stage 2 sleep [46]. The formation and frequency of spindles have been used as markers of the developing brain in infants and change over the lifespan, and they play an important functional role in synaptic plasticity and memory consolidation during sleep. Moreover, sleep spindles are clinically important because alternations in spindle density are observed in several disorders and neurodegenerative diseases. Therefore, detection of sleep spindles in EEGs is very valuable and important in not only clinical diagnosis and but also many ongoing researches.

Research has shown that sleep spindles can be detected by crowd workers hired online, and crowd workers can produce results that are better than any current detection algorithm, and even competitive to experts [46]. According to Warby's paper, a sleep spindle can be defined based on its shape, speed, duration, and height:

- A sleep spindle is usually shaped like a diamond or football.
- A sleep spindle is a group of waves that oscillate at approximately 12 - 15 cycles per second.
- Most commonly sleep spindles are around 0.5 to 1.0 seconds in length, but can be as short as 0.4 seconds and as long as 5 seconds.
- The height of sleep spindles is usually a little larger than the waves around it, although it's less important than other criteria.

Some examples of sleep spindle in an EEG sequence can be seen in Figure 3.2, where sleep spindles are bounded by blue boxes.

We put our tasks onto Amazon Mechanical Turk (Thanks Josh Bradshaw for his implementation and deployment of the experiment). The EEGs we are using are from Stéphanie Devuyst's DREAMS sleep spindles database [13]. For each worker, a task is a 20 seconds window of EEG sequence which may contain one to many sleep spindles. For each worker hired on MTurk, we asked him to perform a total of 55 windows (1080 seconds of EEG in

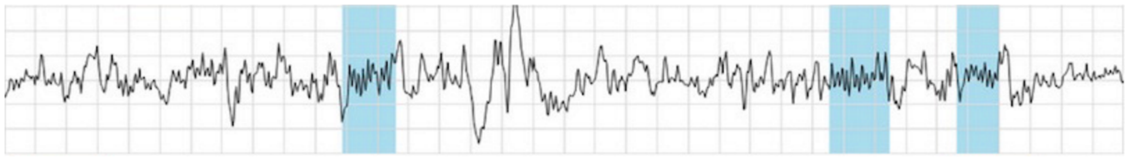

Figure 3.2: Example of Sleep Spindles

total). The windows were pre-shuffled to make sure the difficulty level of each window is mixed enforce some uniformity (as the original sequence of EEG from DREAMS database is getting harder towards the end). Every worker receives the exact same windows, in the exact same order. To evaluate each worker's performance, we compare their submitted sleep spindles to our gold standard data which is based on an agreement between two neurologist's annotations from DREAMS database.

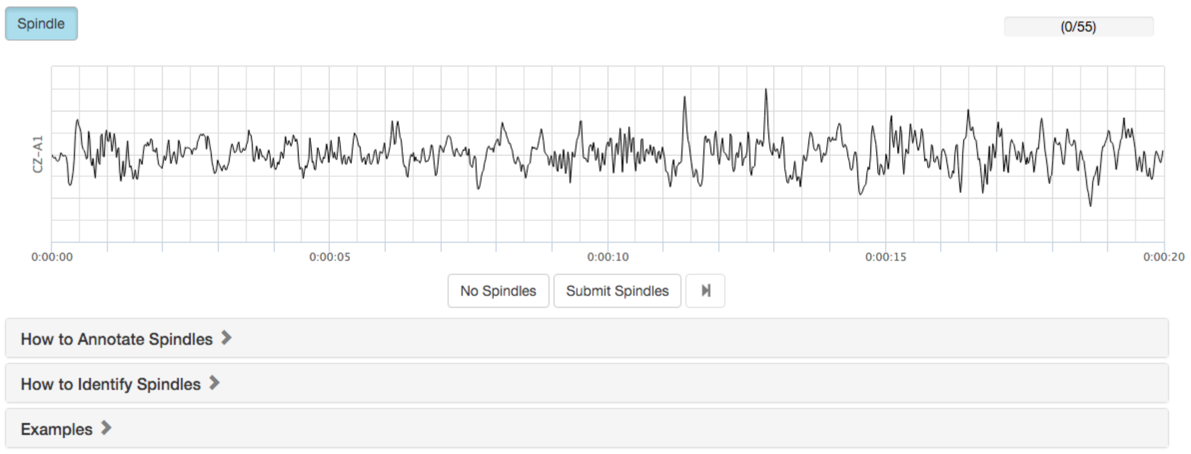

Figure 3.3: Sleep Spindles Detection User Interface

For each 20 seconds window, our user interface on MTurk can be seen in Figure 3.3. A worker can draw light blue boxes in the window to mark segments they think are spindles. Once they submit spindles for the current window, they can see feedback before moving onto next window. The feedback tells them where the actual spindles are in the window which are marked by dark grey boxes as show in Figure 3.4.

Note that a worker is always given the definition of sleep spindles and some examples from our gold standard data showing them how to detect them at the bottom of the

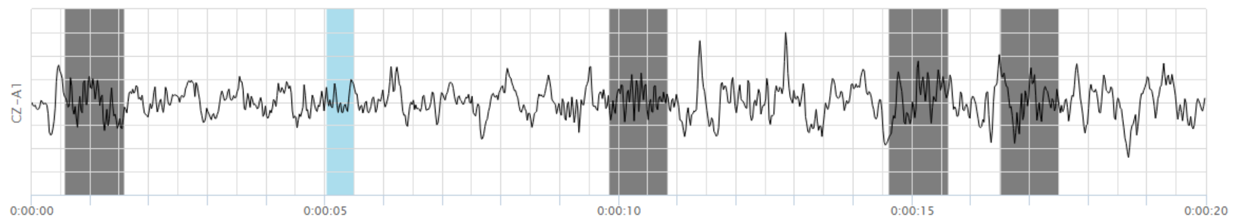

Figure 3.4: Sleep Spindles Detection Feedback to Worker

web page. Payment-wise, we use a structure of fixed payment plus bonus for additional work. As it's quite some work to complete all 55 windows and MTurk users are not often committed to worker on lengthy tasks, we gave them a fixed amount of money for joining and completing the first ten tasks, with extra money for the remaining 45. We filtered out workers who didn't complete all 55 windows for our tests, as we are only interested in analyzing their learning process in a longer horizon. Additionally, to stay away from workers who are possibly spamming, we removed workers who spent too much less time compared to their peers. More specifically, the average time spent on 55 windows was 18.6 minutes, so we removed workers who spent less than 10 minutes which is 45% less than average time consumption. In the end, we got a list of 10 crowd workers, which we will analyze and try to fit with our proposed hyperbolic learning curve in the following sections.

3.2.1 Overall Performance

First of all, we want to test if the worker population as a whole is improving, as our foremost assumption is that the worker population is going to get better after completing more tasks in crowdsourcing settings just like in production settings. Our hypothesis is, although there might be some outliers who are not learning nor improving, if we average every one as a group, their quality should be trending up and show features of learning curve. For sleep spindle detection tasks, we are interested in both precision and recall, so we will analyze them separately.

Since we are conducting our tasks in 20 seconds windows, and workers are only given feedback after completing each window, our analysis is window based and we only update workers' quality information on a per window basis. At the beginning of the tasks, as the sample size is small, a worker may get lucky or unlucky to get the first few tasks all correct or incorrect, thus the quality information is not reliable. Since we are only interested in the general trend, all our analysis is starting from the 10th window to stay away from the such drastic fluctuation. For instance, if a worker got lucky and answered first three tasks correct, it still doesn't make sense to assume he has a 100% quality at that time, and 10 tasks gives a buffer to mitigate the influence of luck. Moreover, as our proposed hyperbolic learning curve represents the cumulative quality of workers, we will be analyzing the cumulative precision (percent of segments a worker marked as sleep spindles are truly sleep spindles), cumulative recall (percent of actual sleep spindles recognized correctly by a worker), and cumulative f-score, which is a combination of these two. We define a worker's reported answer as correct if it overlaps with a sleep spindle in the gold standard dataset.

To analyze the overall performance of the whole worker population, we average cumulative precision, recall, and F-score across all 10 workers by calculating the total percentage of these values by each window. Since everything is on a per window basis, every dot in our plot on x-axis is at a number of spindles by the end of a 20 seconds window. For example, if the first window contains 3 spindles and the second window 2, the cumulative recall is plotted at 3, 5 spindles, respectively, and so forth.

The results can be seen in Figure 3.5. From top to bottom, left to right are cumulative precision, recall, F_1 and F_2 scores, respectively. And we use the common definition for F scores where

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

and

$$F_2 = (1 + 2^2) \times \frac{precision \times recall}{2^2 \times precision + recall}$$

F_1 score treats precision and recall equally, while F_2 score favours recall which could be important for sleep spindle detection tasks as we are more interested in how many sleep spindles a worker can find.

The smoother curves are the estimated learning curves derived by running a linear regression test on the spikier curves (which are the actual averaged cumulative values). For these four quality measurements, the standard error and p-value from linear regression tests are $0.0018/6.36 \times 10^{-14}$, $0.0027/8.36 \times 10^{-9}$, $0.0020/1.07 \times 10^{-12}$, and $0.0023/1.41 \times 10^{-10}$, respectively. Statistically, the p-values are significant and much less than the commonly used threshold of 0.05, which proves the hyperbolic curve is a good fit to represent the underlying worker performance. The graph and results give us an empirical proof that crowd workers are actually learning and improving their quality as they complete more tasks.

The estimated learning speed r^* and prior knowledge p^* are 50/86, 52/116, 50/98, and 51/98, respectively. We expected precision to be the best indicator of crowd workers' learning process for sleep spindle detection tasks, as some EEG may contain spindle segments

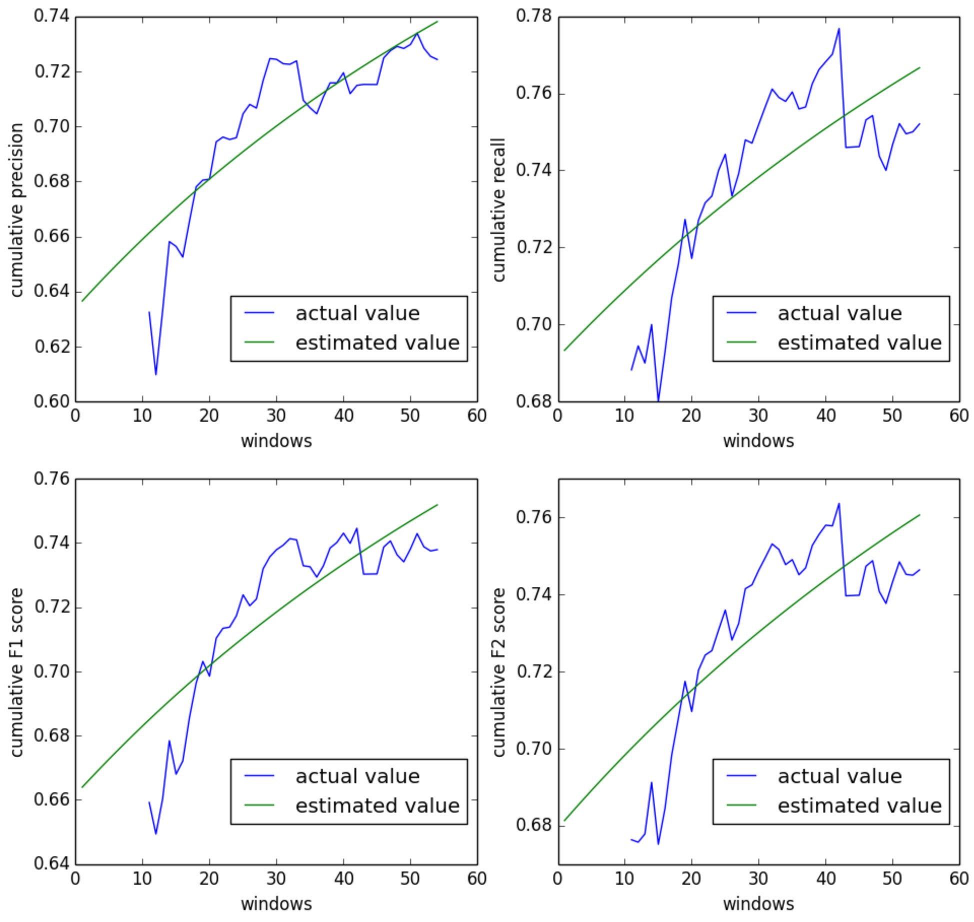

Figure 3.5: Average Cumulative Precision, Recall, F Scores For Worker Crowd

which are difficult to pick up, but once a worker gets better, he is less likely to make any mistake when he annotates something as a sleep spindle. Surprisingly, the learning speed for recall is almost identical to its counterpart for precision, while recall even has a better prior knowledge (starting quality). However, there is a slight drop in performance around the last 10 windows in the recall graph, and consequently in the two F scores. We don't have exact explanation for this phenomenon, but we suspect it's due to the length and repetitiveness of our tasks, as turkers don't usually do such long repetitive tasks. Towards the end some of workers may become too tired to pick up all sleep spindles from the EEG. This could be an interesting future research direction to setup comparison experiments to analyze this behaviour, and we will discuss this in future research section at the end of this thesis.

These experiments show us the crowd worker population as a whole are in fact improving, and their learning process can be modelled by our proposed hyperbolic learning curve. Moreover, with the learning curve model we are able to measure the difficulty of the tasks and evaluate a worker's learning speed quantitatively. For example, our sleep spindle detection tasks have a learning speed r around 50 for all the quality measurements, that in another word is saying it takes workers in general 50 tasks to reach 50% quality by the definition of our learning curve model. This gives us the ability to have a general idea of how fast the worker population can reach a certain quality level. However, to build a system that can hire workers dynamically to take advantage of this learning process, we want to see if this learning curve approach can be applied to crowd workers at an individual level, as each individual worker may have very different learning behaviour.

3.2.2 Fitting Individual Worker

We will only analyze the F_1 score for an individual worker to reduce some redundancy as it combines both precision and recall scores and our previous experiment has already shown that workers' precision and recall curves are very similar. We order all ten workers from 1 to 10, and their individual F_1 curves are shown in Figure 3.6, while their learnt parameters of estimated learning speed and prior knowledge are shown in Table 3.2.2, along with standard errors and p-values from their associated linear regression tests.

We can see from Figure 3.6, other than worker 2 and worker 5, the other 8 workers exhibit clear uptrend quality movement. The 8 workers all have p-values way below 0.05 from their linear regression test which are statistically significant to show strong fitting of their learning curves. This gives us needed validity to use learning curves to model individual worker. For the outliers, worker 2 has a drop of quality after 35 task windows, while worker 5's quality fluctuates up and down, although remains very high across all 55 windows. Although the hyperbolic learning curve may not be a good fit here for these 2, the estimated learning speed is so slow that the projected learning curve is almost a flat line which in fact is a reasonable estimation of worker quality. Thus we think it's safe to use such estimated quality when we are aggregating worker opinions in both situations.

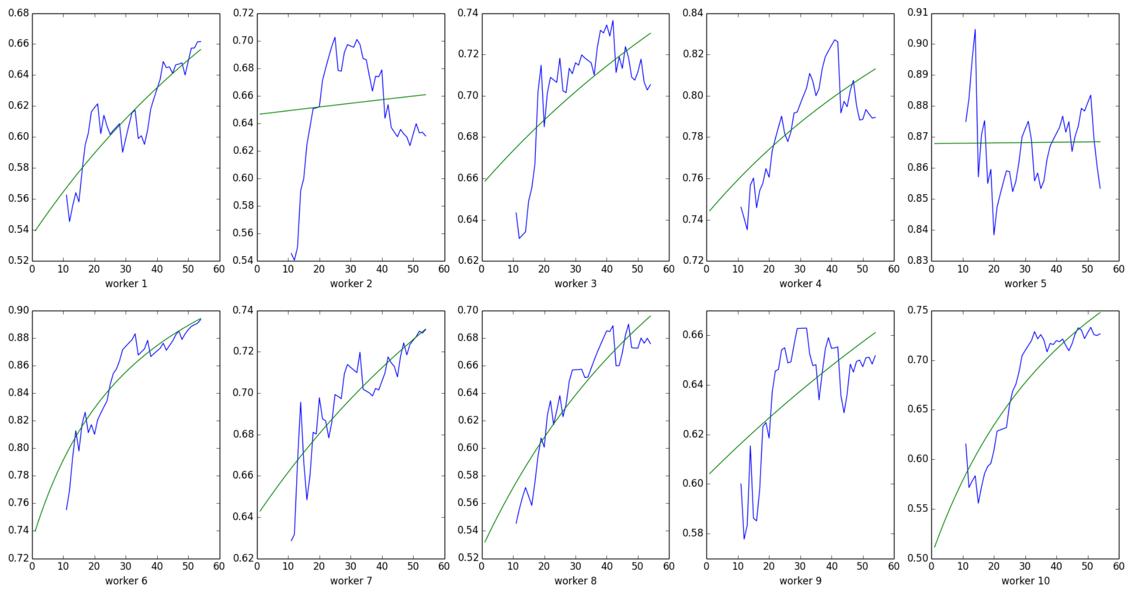

Figure 3.6: Cumulative F_1 Scores For Individual Worker

Worker No.	learning speed (r)	prior knowledge (p)	standard error	p-value
1	72	83	0.001	1.7×10^{-16}
2	450	822	0.004	0.554
3	68	130	0.003	7.2×10^{-7}
4	37	106	0.005	5.0×10^{-7}
5	1613	10597	0.009	0.948
6	9	26	0.006	1.1×10^{-20}
7	58	103	0.001	3.2×10^{-15}
8	46	51	0.001	1.8×10^{-18}
9	125	190	0.002	7.4×10^{-6}
10	28	28	0.003	2.1×10^{-17}

Table 3.1: Learning Speed, Prior Knowledge, Standard Error and P-Value for Individual Worker

3.2.3 Predicting Future Quality with Learning Curve

Overall, the previous experiment gives us confidence to use hyperbolic learning curves to model individual crowd worker's quality. However, to take advantage of their quality improvement, we want to see if we can use our learning curve model to project their future quality level so we can make hiring decision based on not only their current quality but also their future quality, as one goal of our going to build hiring system is to train high potential workers to benefit in the long run. So we want to test the robustness of using workers' performance in first few windows to predict their performance in later tasks. To show how accurate the prediction is, as well as its relationship to number of samples used for running linear regression test, we did the following experiments that draw estimation from both the first 20 and 30 windows. We tried for 10 windows too but found the results are not too reliable for such small amount of samples.

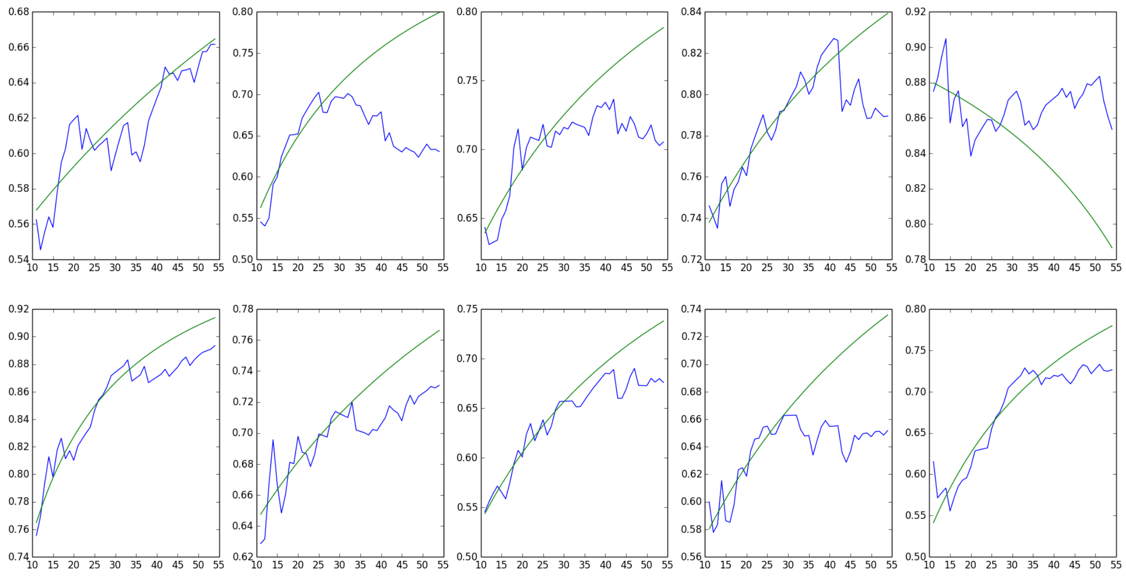

Figure 3.7: Cumulative F_1 Scores For Individual Worker Projected From 20 Windows

The results are shown in Figure 3.7 and Figure 3.8. We can see that for a projected learning curve from the first 20 windows, about half of them are fitting the actual cumulative F_1 score curve pretty well, while the other half overestimate at the end when there appears to be a drop in quality. And for worker No. 5 the curve goes downward due to downward movement at the first 20 windows. We compare the difference between the projected cumulative F_1 score and the actual cumulative F_1 score at both window 45 and 55, as well as the number if we assume crowd workers have a fixed quality (so they will have the same F_1 score at all window 30, 45, and 55). The results are displayed in Table 3.7.

No.	F_1 at 45	fixed	estimated	F_1 at 55	fixed	estimated
1	0.64	0.60 (-6%)	0.63 (-1.5%)	0.66	0.60 (-9%)	0.66 (0%)
2	0.63	0.69 (+9.5%)	0.73 (+15.8%)	0.63	0.69 (+9.5%)	0.79 (+25.4%)
3	0.71	0.72 (+1.4%)	0.74 (+4.2%)	0.71	0.72 (+1.4%)	0.79 (+11.3%)
4	0.79	0.80 (+1.3%)	0.80 (+1.3%)	0.79	0.80 (+1.3%)	0.84 (+6.3%)
5	0.87	0.87 (0%)	0.84 (-3.4%)	0.85	0.87 (+2.4%)	0.79 (-7%)
6	0.88	0.87 (-1.1%)	0.88 (0%)	0.89	0.87 (-2.2%)	0.91 (+2.2%)
7	0.71	0.71 (0%)	0.72 (+1.4%)	0.73	0.71 (-2.7%)	0.76 (+4.1%)
8	0.70	0.66 (-5.7%)	0.68 (-2.9%)	0.68	0.66 (-2.9%)	0.74 (+8.8%)
9	0.64	0.66 (+3.1%)	0.68 (+6.3%)	0.65	0.66 (+4.6%)	0.73 (+12.3%)
10	0.72	0.71 (-1.4%)	0.71 (-1.4%)	0.73	0.71 (-2.7%)	0.78 (+6.8%)

Table 3.2: Comparing Actual, Fixed, and Estimated F_1 Scores, 20 Sample Windows

In Table 3.2.3, the fixed cumulative F_1 scores are the scores at window 30 (after taking window 10 - 30 as the learning samples) when we assume workers have a fixed quality (this is used in many hiring mechanisms which pick best workers from past performance). And estimated F_1 scores are the ones projected by learning curve provided by linear regression test with the same 10 windows of samples. The percentage numbers in brackets are the difference from the actual cumulative F_1 scores at window 45 and 55, respectively. We can see from the table that 9 out of 10 workers' estimated score are within 10% range of the actual value at window 45 (7 out of 10 within 5%), while 7 out of 10 are with in 10% at window 55. This shows that it's more reliable towards the near future than windows far away, while in general it's still a decent measurement. Moreover, at window 45, 5 out of 10 workers have estimated score equal or better than their fixed counterparts. However at window 55 this number decreases to 2 due to a quality drop for some of the workers at the end. We are not sure if the quality drop is actually due to workers feeling tired and becoming lazy, or if this is just some fluctuation which will recover if we can get more tasks. At the moment we cannot deal with recurring workers on MTurk and we are unable to make workers work on longer tasks (as our tasks are already lengthy). If we can setup a crowdsourcing platform where workers are allowed return, we can design comparison experiments to analyze this dropping effect, and discuss the results for a longer horizon of tasks. This could be an interesting research direction for the future.

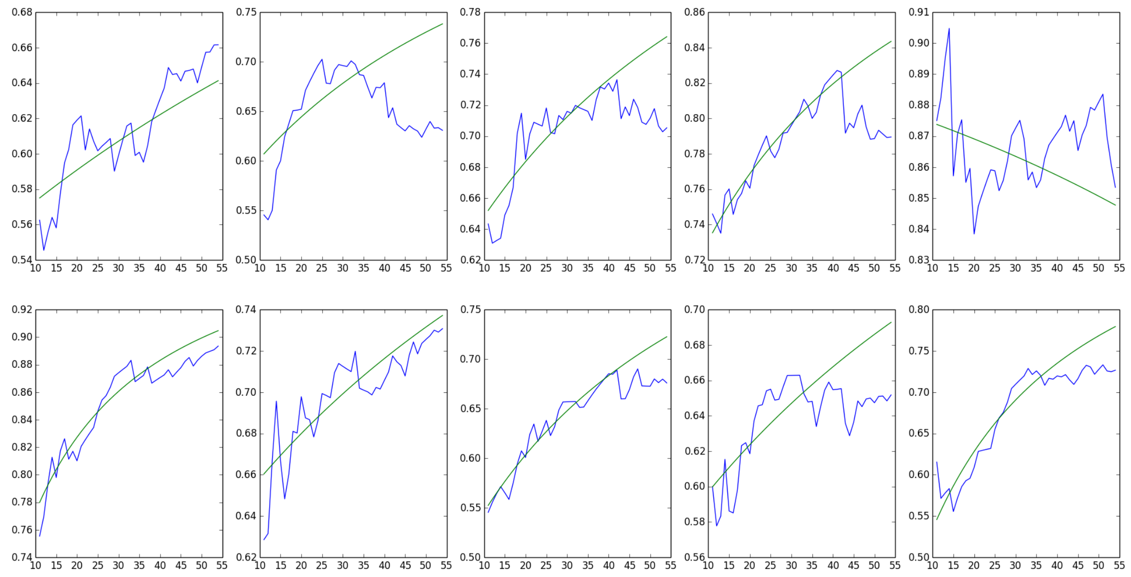

Figure 3.8: Cumulative F_1 Scores For Individual Worker Projected From 30 Windows

For projected cumulative F_1 scores learnt from first 30 windows, the ones that were doing well in Figure 3.7 fit even better, while another two of the workers become well fitted now. From the graph, seven workers have a well fitted learning curve. The actual numbers are listed in Table 3.2.3. Now we have 9 out of 10 workers' cumulative F_1 scores estimated within 10% range of the true value for both window 45 and window 55, except for worker No. 2 who has a significant performance drop after window 30. This observation stays inline with our simulation result in next chapter where we found 30 samples is a good threshold for a linear regression test to provide a mostly reliable learning curve while 20 samples still provide considerable noise. Note that the fixed F_1 scores are taken from window 40 so it's very close to the true F_1 scores at window 45 as they are just 5 windows away.

However, from both Figure 3.7 and Figure 3.8 we can observe that although workers'

No.	F_1 at 45	fixed	estimated	F_1 at 55	fixed	estimated
1	0.64	0.63 (-1.6%)	0.63 (-1.6%)	0.66	0.63 (-4.5%)	0.64 (-3.0%)
2	0.63	0.68 (+7.9%)	0.72 (+14.2%)	0.63	0.68 (+7.9%)	0.74 (+17.4%)
3	0.71	0.73 (+2.8%)	0.74 (+4.2%)	0.71	0.73 (+2.8%)	0.76 (+7.0%)
4	0.79	0.82 (+3.8%)	0.82 (+3.8%)	0.79	0.82 (+3.8%)	0.84 (+6.3%)
5	0.87	0.87 (0%)	0.85 (-2.3%)	0.85	0.87 (+2.4%)	0.85 (0%)
6	0.88	0.87 (-1.1%)	0.89 (+1.1%)	0.89	0.87 (-2.2%)	0.90 (+1.1%)
7	0.71	0.71 (0%)	0.72 (+1.4%)	0.73	0.71 (+2.7%)	0.73 (0%)
8	0.70	0.69 (-1.4%)	0.70 (0%)	0.68	0.69 (+1.5%)	0.72 (+5.9%)
9	0.64	0.65 (+1.6%)	0.67 (+4.7%)	0.65	0.65 (0%)	0.69 (+6.1%)
10	0.72	0.72 (0%)	0.75 (+4.2%)	0.73	0.72 (+1.4%)	0.78 (+6.8%)

Table 3.3: Comparing Actual, Fixed, and Estimated F_1 Scores, 30 Sample Windows

quality are trending up generally, their immediate quality may fluctuate from time to time. If we use a fixed number of tasks to take their averaged quality for future reference, it may be far off. For example, if instead of using cumulative F_a score at window 30 and 40 for fixed quality, it can be as much as a 15 - 20% difference for half of the workers if we take fixed quality from window 15 or 20. And in real world applications it's almost impossible to find the perfect stop point, while the estimated quality from learning curve projection provide a consistently reliable estimation.

The experiments on MTurk have shown us that crowd workers definitely are getting better and follow the learning curve model when doing more complex tasks such as sleep spindle detection. It builds the ground for our proposed dynamic hiring process that can take advantage of their quality improvement and a more reliable and consistent estimation of workers' quality from learning curve projection. We will introduce our hiring mechanism in next chapter, but for now let's first discuss some of the problems we are facing with this learning model.

3.3 Problems and Challenges

The first problem we are facing is that we may not have enough data to get a very accurate estimation of the parameters, especially for brand new tasks where no worker history is available, even if the workers' cumulative qualities are perfectly following learning curve patterns. To illustrate this, we setup a simulation of 1000 workers where their cumulative qualities are generated exactly from 3.1, with each learning speed r drawn from truncated normal distribution

$$R(x) = f(x; 50, 10, 0, \infty)$$

and prior knowledge p drawn from truncated normal distribution

$$P(x) = f(x; 65, 10, 0, \infty)$$

Note that in this thesis a truncated normal distribution is defined as:

$$f(x; \mu, \sigma, a, b) = \begin{cases} \frac{\frac{1}{\sigma}\phi(\frac{x-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma})-\Phi(\frac{a-\mu}{\sigma})} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

where ϕ and Φ are probability density function and cumulative distribution function of the standard normal distribution, and μ and σ are mean and variance of the corresponding normal distribution. In this simulation we created 500 binary (two outcomes) tasks and tried to measure the difference between our estimated worker learning speed and quality, and the true values.

In Figure 3.9, the two graphs in the leftmost columns present the averaged percentage difference of workers' estimated values to their true values, while the other three columns measure percentage of workers whose estimated values are within a certain difference range from their true values. As we can see, after completing 500 tasks, majority (60%) of workers' learning speed can be estimated within 15% range to their true learning speed. On the other hand, the estimation of workers' quality is actually much better, in fact after completing 100 tasks, more than 90% of workers' quality can be estimated within the 15% difference range. Moreover, half of them can be estimated within 15% range after completing 50 tasks. This result tells us we cannot trust the estimated quality nor learning speed for a worker when his number of completed tasks is low.

There is another challenge we are facing, too. Note here we are making an assumption that we have true knowledge about the actual answers to all the tasks which are generally unavailable in settings of crowdsourcing as the goal is to predict them. We may run a few tutorial tasks first with true answers to kickoff the estimation to mitigate this problem. But we still want to analyze scenarios where the aggregated opinion we are comparing each agent's reported opinion to is not matching the underlying truth. In fact, if we assume the probability of the aggregate opinion being correct is ρ, and let c denote the number of tasks the worker predicts correctly, then the data points we collect for a worker's cumulative quality becomes:

$$Q^*(x) = \frac{c - (1-\rho)c + (x-c)(1-\rho)}{x}$$

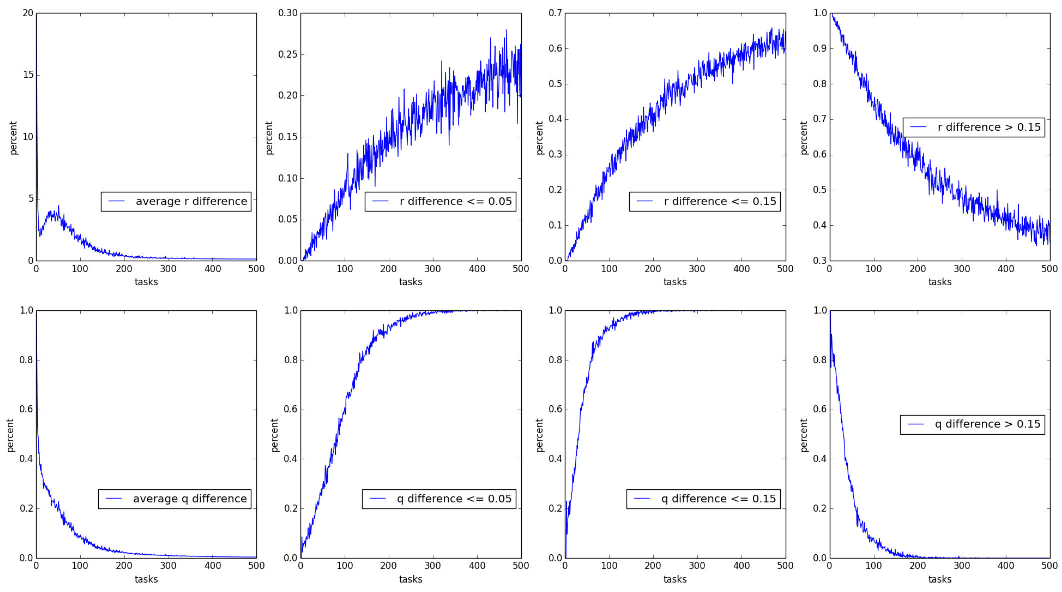

Figure 3.9: Learning Speed and Estimated Quality Difference

Now the estimated number of correct answers is his true correctness in addition to the answers he gets wrong but are recognized mistakenly by the system minus the answers he gets correct but are classified incorrectly due to noise introduced into opinion aggregation. For example, if all workers are working on some really difficult tasks they have never seen before, they may all report the wrong answer thus the final prediction itself is wrong. When we are measuring their quality, we will then use the wrong prediction and update their quality information wrongly in consequence. Once we simplify it, we obtain:

$$Q^*(x) = (2\rho - 1)\frac{c}{x} + (1 - \rho) = (2\rho - 1)Q(x) + (1 - \rho) \tag{3.4}$$

Then with 3.2 we can derive the estimated quality function as:

$$\begin{aligned} q^*(x) &= xQ^*(x) - (x-1)Q^*(x-1) \\ &= x[(2\rho - 1)Q(x) + (1-\rho)] - (x-1)[(2\rho-1)Q(x-1) + (1-\rho)] \end{aligned}$$

After simplification, it becomes:

$$q^*(x) = (2\rho - 1)q(x) + (1 - \rho) \tag{3.5}$$

If we compare it to the actual quality, the difference is

$$q(x) - q^*(x) = (1 - \rho)(2q(x) - 1) \tag{3.6}$$

Note that when x goes to $+\infty$, $\lim_{x\to\infty} q(x) = 1$ for any crowd worker with a positive learning speed r. Thus with 3.5 we get $\lim_{x\to\infty} q^*(x) = \rho$. That is, after a long enough horizon of tasks, the learnt quality from regression analysis will merge to the accuracy of opinion aggregation. This problem can be mitigated as the accuracy of opinion aggregation increases as more tasks have been completed by workers and their qualities are improving. But combined with our observation of the unreliability of linear regression approach for small sample size, it is wise to not update crowd workers' learning curves until we collect enough data.

Now with the learning process model we have all the tools to start designing our dynamic hiring process. The problems we mentioned above will be handled in next chapter when we talk about quality estimation in opinion aggregation. We will also discuss the pros and cons of spending money on doing tutorial tasks and the tradeoff between tutorial tasks and accuracy of linear regression tests later.

Chapter 4

A Dynamic Hiring Process

The core of this thesis is to design a hiring mechanism that makes smart decisions dynamically about when and whom to hire. In this chapter, we propose using Markov Decision Process (MDP) and Monte Carlo Planning to sample all possible outcomes, and follow the hiring path that yields maximal utility to make all hiring decisions. We first introduce the model of MDP and explain how we model the hiring process as a MDP.

4.1 The Markov Decision Process

When solving consensus tasks, the hiring system has two options to choose from at any time. The first is to hire another worker and let him report his opinion. The other option is to stop hiring for the current task, aggregate all collected opinions, produce a prediction to the task, and move onto the next task. To make these decisions, the system is actually looking for the sweet spot between making more accurate prediction, the costs of paying additional workers, and the future benefit of improving agents' quality by training. The goal of our work is to build a decision theoretic hiring system that can model these decisions and find the one which optimizes the expected utility, which by definition is the statistical expectation of the system's valuation of the outcomes.

For each consensus task, this optimization problem can be formalized as a Markov Decision Process (MDP). With MDP, each state can store the information the system collected about its belief in the aggregated answer, as well as crowd workers' estimated quality and learning speed. Hiring decisions are made to maximize the system's net utility which is a combination of the system's utility for delivering the predicted answer, the cost of hiring workers, and the long term reward of having well trained workers.

4.1.1 Defining the MDP

The hiring process to solve one consensus task can be formalized as a tuple $< S, \mathrm{A}, T, R >$:

- $s \in S$ is a state of the MDP, which is a tuple of $< \{w_i\}_1^n, \{a_i\}_1^n >$. Every w_i is a crowd worker with estimated learning speed $r^*_{w_i}$, prior knowledge $p^*_{w_i}$, and quality function $q^*_{w_i}(x)$ all learnt with the learning curve model and regression test discussed in last chapter. $\{w_i\}_1^n$ are all hired workers till current state. And $\{a_i\}_1^n$ are reported answers collected from all hired workers, where a_i is the answer reported by worker w_i. $S^T \subset S$ is the set of termination states. Once the MDP reaches a termination state, the system stops hiring and then it aggregates opinions to produce the final prediction for the task.
- The set of actions are A. An action $\alpha \in \mathrm{A}$ can either be $\neg H$ not hiring, or H_w hiring a new worker w from the worker pool. After the system ends up in a termination state, no hiring action can be taken any more.
- $T(s, \alpha, s')$ is the transition probability from state s to state s' by taking action α. The transition function is determined by the system's current belief on the aggregate opinion, and the knowledge it has on crowd workers. We will go back to the details of estimating the transition probability function later when we introduce the Monte Carlo planning approach to solve this MDP.
- $R(s, \alpha)$ is the reward associated with taking action α at state s. The reward for taking hiring action H_w at a state is the benefit the system gets for training worker w minus the cost of hiring w. On the other hand, taking action $\neg H$ always lead to a termination state, as we do not allow the system to make pauses between hiring workers for a single task. When reaching a termination state, the associated reward is uncertain, and depends on the quality of the aggregate prediction, which is estimated by the system's belief on the accuracy of the prediction. We will look into the details of our opinion aggregation approach and the reward functions in the following subsections.

Note that we are not using the discount factor in our MDP, or in another word, it's always set to 1, as the rewards for hiring all workers and getting the aggregate opinion is equally important to the system. A policy π specifics the action that the system chooses at any state s, and an optimal policy π^* maximizes the system's utility from any state s:

$$V^{\pi^*}(s) = \begin{cases} R(s, \neg H) & \text{if } s \in S^T \\ max_{\alpha \in \mathrm{A}}(R(s, \alpha) + \sum_{s'} T(s, \alpha, s') V^{\pi^*}(s')) & \text{otherwise} \end{cases} \tag{4.1}$$

4.1.2 Opinion Aggregation

Before going any further, we need to first explain how we aggregate hired workers' opinions and calculate the system's belief in the generated prediction, as the belief is directly tied to the reward the system is getting at any termination state.

Once the system has all hired agents' reported answers and reaches the termination state, it needs to create a final prediction. Some traditional methods to aggregate these collected answers are voting, opinion pooling, and etc., where were discussed in artificial intelligence literatures [11]. However, since we have the estimated quality information of all our hired workers from our learning curve model, we can instead model their predicting process as a Bayesian network. In fact, it's not unrealistic to assume each crowd worker's prediction is independent given the true answer to a task. For a task with ground truth answer $\hat{a}$ and a set of n hired workers $w_1, w_2, w_3, ..., w_n$ and their reported opinions $a_1, a_2, a_3, ..., a_n$, we have a simple Bayesian network that looks like the following:

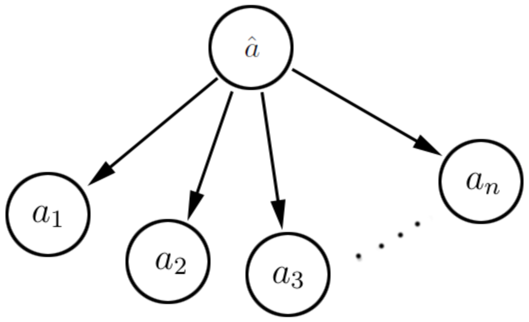

Figure 4.1: Bayesian Network for Opinion Aggregation

With Bayes' rule, we can derive the probability of an aggregate answer $\bar{a}$ being the true answer $\hat{a}$ as:

$$Pr(\hat{a} = \bar{a} | \{a_i\}_1^n) = \lambda_{\bar{a}} Pr(\{a_i\}_1^n | \hat{a} = \bar{a}) Pr(\hat{a} = \bar{a}) \tag{4.2}$$

where λs are normalizing factors to make sure

$$\sum_{\bar{a} \in A} Pr(\hat{a} = \bar{a} | \{a_i\}_1^n) = 1$$

To solve $Pr(\hat{a} = \bar{a}|\{a_i\}_1^n)$, we need to calculate both $Pr(\{a_i\}_1^n|\hat{a} = \bar{a})$ and $Pr(\hat{a} = \bar{a})$. Since we assume each worker reports his own opinion independently, the first one becomes:

$$Pr(\{a_i\}_1^n|\hat{a} = \bar{a}) = \prod_{i=1}^{n} Pr(a_i|\hat{a} = \bar{a})$$

As in this thesis we are only interested in binary tasks that contain only two possible outcomes, then we have:

$$Pr(a_i|\hat{a} = \bar{a}) = \begin{cases} q_{w_i}(x) & \text{if } a_i = \bar{a} \\ 1 - q_{w_i}(x) & \text{otherwise} \end{cases}$$

In another word, at task x, given the true answer to the task, the i-th worker with quality $q_{w_i}(x)$ has a probability of $q_{w_i}(x)$ to predict the correct answer, and a probability of $1 - q_{w_i}(x)$ to report the wrong one. For tasks with more than two outcomes, we may need to study and define a probability distribution for all incorrect answers based on the natural and underlying relations between them.

Now we need to estimate $Pr(\hat{a} = \bar{a})$, which cannot be calculated from workers' qualities. These probabilities are estimated from historical data or observatiosn against the nature of the underlying tasks if no past history is available. For example, if there are only two outcomes and in the past outcome a_1 happened 30% of times while a_2 70%, we then assume $Pr(\hat{a} = a_1) = 0.3$ and $Pr(\hat{a} = a_2) = 0.7$. Of course these probabilities are not set in stone and need to be updated after every single task to reflect the latest trend. In our analysis, experiment and simulation after, we assume all tasks are starting fresh and we have no knowledge of historical performance to make sure everything is generic enough. So we assume all outcomes are equally likely to happen at the beginning and use Laplace smoothing [30] to take care of data fluctuation. More specifically, we set the smoothing factor to 10 and let

$$Pr(\hat{a} = \bar{a}) = \frac{o_{\bar{a}} + 10}{N + 10 \times |A|} \tag{4.3}$$

where N is the total number of tasks the system has observed and $o_{\bar{a}}$ is the number of tasks where $\bar{a}$ was the final answer.

Now we have all the information to calculate the probabilities of all potential outcomes, and our system will pick the one with highest probability as the aggregate answer across all reported answers, where the associated probability is the system's belief of the likelihood the aggregate prediction being correct. Note that one problem we face when aggregating answers from crowd workers is that we want them to report their true opinions. Lying agents not only cause trouble to opinion aggregation, but also to our learning of their true quality information. There are some proposed methods to reinforce truthful reporting like using scoring rules [38] [8] [15] [32] [44]. As this is not the focus of this thesis we will not dive into any details. Additionally, our system proposed in this chapter is independent to different payment allocation methods and scoring rules. Thus, if you want to ensure truthful reporting from your hired workers, you may adapt any of the scoring rules as you like and reallocate payment to hired workers for each task based on the scoring rule you are applying.

4.1.3 Defining the reward function

In our model, there are two types of rewards. After taking action $\neg H$ and arriving at a termination state s^T, it's natural to set the reward based on the accuracy of the aggregate opinion. Ideally, the system should get reward when its prediction is correct and no reward if it was wrong. However, the system may have no knowledge about the underlying truth (that's why we are crowdsourcing these tasks in first place), so we have to make an estimation based on information we have collected. Recall from our previous discussion, we can calculate the estimated probability of an outcome being the true answer to the task from 4.2, given collected reported answers $\{a_i\}_1^n$. In fact, we define such probability as the system's belief b in the aggregate opinion $\bar{a}$:

$$b(\bar{a}|\{a_i\}_1^n) = Pr(\hat{a} = \bar{a}|\{a_i\}_1^n) \tag{4.4}$$

where $Pr(\hat{a} = \bar{a}|\{a_i\}_1^n)$ can be calculated from 4.2. After collecting all hired crowd workers' opinions, we set the aggregate answer $\bar{a}$ to the outcome with the highest $Pr(\hat{a} = \bar{a}|\{a_i\}_1^n)$, and then we define the reward function as the following:

$$R(s, \neg H) = \begin{cases} b_w \times (2^{b(\bar{a}|\{a_i\}_1^n)} - 1) & \text{if } b(\bar{a}) \geq b_t \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

where b_w is the associated weight assigned to the belief, and b_t is a threshold parameter so that only answer with probability higher than it gets reward.

The purpose of using weight parameter is to adjust the relative value of an accurate prediction to the monetary cost we are paying to each hired worker. The exact value of the weight parameter is determined later to one that optimizes the system's performance from the simulation test. The reason to have an exponential (convex) function for the reward is that improvement in the higher accuracy range is more important than in lower accuracy range. For example, a predicted answer's quality improvement from 50% accuracy to 60%

accuracy is not as valuable as one improved from 80% to 90%, because it becomes harder to improve the accuracy of prediction once it gets to a higher level. Of course it depends on how the task owner views the answer quality. It can be swapped to other functions according to ones specific needs. In this thesis we stay with the exponential function as we value higher accuracy predictions.

Moreover, we cut off the reward function at a threshold higher than 50% because when workers' qualities are low, it's very likely the Monte Carlo sampling approach we are later using will produce conflicting opinions when the number of hired workers is low, thus may stop the system from hiring more workers. For instance, if we only have a group of workers with qualities around 50%, two hired workers may produce opposite opinions. In this case, there is a huge reward drop from one hired worker if reward is assigned to all belief levels, and it will cause the system to stop hiring after one worker, which is not exactly what we wanted. When there is a cutoff threshold, hiring one worker and two workers or even more will produce the same amount of reward for the aggregate opinion unless it hits the threshold. So it gives us the ability to require a minimal level of confidence in our aggregate opinion and let the system keep hiring until we hit such a minimum. In case the minimum cannot be reached, the system will keep hiring up to the upper limit which in turn gets more workers trained.

The second type of reward is the benefit the system receives in the long run by training and improving a worker's quality. At state s, if the action to take is H_w that hires crowd worker w, then we have w's estimated quality function $q_w^*(x)$ and the number of his completed tasks x_w. Thus, the estimated quality improvement to w for working on the task is

$$q_\Delta^*(w) = (2^{q_w^*(x_w+1)} - 1) - (2^{q_w^*(x_w)} - 1)$$

and the associated reward for hiring action is

$$R(s, H_w) = q_w \times q_\Delta^*(w) - c_w \tag{4.6}$$

where c_w is the cost to hire worker w. If all agents cost the same amount of money, then c_w can be replaced by a fixed c. Similarly, q_w is the monetary weight factor for the quality improvement. Note that here we are using an exponential function to map workers' quality, too, as we think an worker's quality improvement in the higher range as more valuable to the system. For example, an improvement from 80% to 90% for a worker may enable the system to hire fewer workers, which cannot be said to ones improvement from 50% to 60%. Now we have everything defined for the MDP, we can move on to find a feasible solution for it.

4.2 Solving the MDP

The goal of solving MDP problems is to find the optimal policy at each state. However, when the state-space is large, as in our case of crowdsourcing consensus tasks, it's computationally infeasible to exhaust all possible action paths to find optimal solution. Moreover, it's impossible for task assigner to wait too long before making any hiring decisions. As a result, all we need to do is to design an algorithm which can run in timely fashion to find a near optimal solution.

One of the few viable approaches is to carry out sampling based lookahead search, and among them Monte Carlo planning is the most popular one to address such challenges. Monte Carlo planning is an online planning approach that can explore the search space of a large MDP efficiently. It uses sampling and builds partial search trees in the exploring phase to optimize decisions [27]. Along the years, many Monte Carlo algorithms have been proposed. However, in practice they may not be able to find actions under runtime limitation, as the amount of work needed to calculate near optimal action at any given state can be huge. In Kamar's work [24], she showed the general purpose Monte Carlo algorithm UCT[28] hits a combinatorial challenge in solving problems with long horizons.

Kamar and *et al* proposed a MC-VOI algorithm that customizes the sampling method in Monte Carlo planning to explore long horizons [24]. The MC-VOI algorithm can eliminate action selection in sampling when no hiring actions lead to termination states. Moreover, for each sample, it can evaluate the utility of hiring and no hiring simultaneously at each encountered state, which reduces the number of samples required to explore consensus tasks with long horizons [25]. The core concept of MC-VOI is to calculate the expected value of information (VOI) from sequences of observations. In their work, they proposed a solution to compute VOI for long evidential sequence (LES) tasks. By their definition, a LES task is a task that focus on identifying the best domain action to take under uncertainty about the world. It terminates when a domain action is taken, and is assigned a reward based on the action and the state of the world [24]. Clearly, consensus tasks are LES tasks where opinions reported by agents are evidential sequences.

We can then adapt and modify their notion of VOI, as in our setting hiring different workers has different implications. For any given state s and any worker w available at s:

$$VOI(s,w) = R(s,H_w) + \sum_{s'} T(s,H_w,s')V^{\pi^*}(s') - R(s,\neg H) \tag{4.7}$$

where $V^{\pi^*}(s)$ is defined in 4.1. $VOI(s,w)$ defines the quantitative value of hiring worker w at state s. At any state $s \notin S^T$, the system explores the VOI of the next worker to hire.

If the VOI is negative, then not hiring has a higher expected utility than hiring him. In that case the system goes to the termination state and report the aggregated answer.

We also need to estimate the transition model for the MDP as its structure is not known to the system. With the quality information we have, we need to come up with a way to calculate $T(s, H_w, s')$ for any $s, s' \in S$ and $w \in W$. In fact, at state s, after taking action H_w, all possible next states are results of w's answer a_w. In another word, if s is $< \{w_i\}_1^n, \{a_i\}_1^n >$, then all possible next states s's are $< \{w_i\}_1^n \cup w, \{a_i\}_1^n \cup a_w >$, where a_w can possibly be any answer in the answer space A. We use a function g to simplify the notation where

$$g(s, a_w) =< \{w_i\}_1^n \cup w, \{a_i\}_1^n \cup a_w >$$

for $s =< \{w_i\}_1^n, \{a_i\}_1^n >$. Then 4.7 becomes

$$VOI(s, w) = R(s, H_w) + \sum_{a_w \in A} T(s, H_w, g(s, a_w))V^{\pi^*}(g(s, a_w)) - R(s, \neg H) \tag{4.8}$$

and

$$T(s, H_w, g(s, a_w)) = Pr(a_w|\{a_i\}_1^n)$$

where $Pr(a_w|\{a_i\}_1^n)$ is the likelihood of worker w reporting a_w after the system has already collected $\{a_i\}_1^n$. Apparently, there is no way to calculate the exact value of $Pr(a_w|\{a_i\}_1^n)$ as the system has no knowledge about the ground truth nor each worker's true quality or behaviour. But recall from 4.2, since we are only dealing with binary tasks in this thesis, we can estimate these probabilities as:

$$Pr(a_w = a_j|\hat{a} = a_j) = q_w \tag{4.9}$$

and

$$Pr(a_w \neq a_j|\hat{a} = a_j) = 1 - q_w \tag{4.10}$$

We also assume an agent's answer reporting is independent from others. Now it becomes a Bayesian network and we can calculate the transition probability as:

$$\begin{aligned}
T(s, H_w, g(s, a_w)) &= Pr(a_w|\{a_i\}_1^n) \\
&= \sum_{\hat{a} \in A} Pr(a_w, \hat{a}|\{a_i\}_1^n) \\
&= \sum_{\hat{a} \in A} Pr(a_w|\hat{a}, \{a_i\}_1^n)Pr(\hat{a}|\{a_i\}_1^n) \\
&= \sum_{\hat{a} \in A} Pr(a_w|\hat{a})Pr(\hat{a}|\{a_i\}_1^n) \\
&\propto \sum_{\hat{a} \in A} Pr(a_w|\hat{a})Pr(\{a_i\}_1^n|\hat{a})Pr(\hat{a})
\end{aligned}$$

or

$$T(s, H_w, g(s, a_w)) = \gamma \sum_{\hat{a} \in A} Pr(a_w|\hat{a}) Pr(\{a_i\}_1^n|\hat{a}) Pr(\hat{a}) \tag{4.11}$$

where γ is a normalizing factor to make

$$\sum_{a_w \in A} T(s, H_w, g(s, a_w)) = 1$$

In equation 4.11, $Pr(a_w|\hat{a})$ can be calculated from 4.9 and 4.10. Although we cannot get the exact number as we do not have the true quality of a worker, we can use our learnt estimated quality function $q_w^*(x)$ to get a good estimation. Similarly,

$$Pr(\{a_i\}_1^n|\hat{a}) = \prod_{i=1}^{n} Pr(a_i|\hat{a})$$

and $Pr(a_i|\hat{a})$ is calculated from 4.9 and 4.10, too. Then we are left with $Pr(\hat{a})$, which can be estimated by keeping track of the distribution of aggregated answers to completed tasks, i.e. as what we did with Laplace smoothing in 4.3 when we discussed opinion aggregation.

The core idea of our system is the same to Kamar's MC-VOI algorithm for static LES tasks proposed in [24]. We sample all the states from top to bottom and evaluate rewards bottom up. The algorithm builds a partial search tree iteratively sampling the execution path. All sampling starts from the initial state s_0 where no worker has been hired. At each sampling call, the algorithm keeps hiring new workers and visiting new states until it hits the horizon (maximal number of workers to hire), then it ends in a termination state. The fixed horizon length makes sure the algorithm will always terminate. After hiring a new worker, the execution path reach the next state based on the probability calculated from 4.11. The algorithm grows the search tree by adding a new node whenever sampling encounters a new state.

However there is one major differences between our settings and the settings in Kamar's MC-VOI algorithm. In Kamar's setting, when the action is to hire another worker, there is no difference between all potential workers, as they assume all workers are the same. As in our system, different workers have different quality and learning speed, theoretically, we should sample all combinations of potential worker and his opinion at each state, and it can end up in $m|A| + 1$ different states. This is infeasible as the size of the crowd worker pool can be large (even a very small pool can grow the sampling tree dramatically, as the size of the tree is exponential to the number of possible outcomes at each state). We will discuss this problem in the runtime limitation subsection and propose a workaround approach to handle it.

The evaluation phase updates the expected utility and number of visitation at each state from bottom to top after all the sampling is done. For any state s that is at the maximal length of the horizon, it can only go to a termination state s^T. Thus we set its optimal expected utility $s.utility$ (this notation means *utility* is a property of s) to the immediate reward of aggregating collected opinions $R(s^T, \neg H)$, and we store the number of samples that visited s to $s.visitation$. For any other non-termination state s, any edge leading from s to its non-termination child s' can be defined by the hired worker w with his answer a_w, i.e. $s \xrightarrow{<w,a_w>} s'$. The estimated value of information of hiring a worker w can then be calculated from

$$VOI^*(s, w) = R(s, H_w) + \sum_{a_w \in A} T'(s, H_w, g(s, a_w))g(s, a_w).utility - R(s, \neg H) \qquad (4.12)$$

as the optimal expected utility $g(s, a_w).utility$ is already known, and $T'(s, H_w, g(s, a_w))$ is the estimated probability from sampling defined as

$$T'(s, H_w, g(s, a_w)) = \frac{\text{number of samples hired } w \text{ with reported answer } a_w \text{ at } s}{\text{number of samples reached } s}$$

If $VOI^*(s, w) < 0$, then no hiring is the optimal action here, and we set $s.utility$ to $R(s, \neg H)$. Otherwise, we set $s.utility$ to

$$\max_w (R(s, H_w) + \sum_{a_w \in A} T'(s, H_w, g(s, a_w))g(s, a_w).utility)$$

and set $s.tohire$ to w, where w is the worker we get the optimal utility $s.utility$ from.

After all evaluation is done, we can then perform the actual hiring based on all the information we have stored into each state. Starting from initial state s_0, for each state s being checked, if $s.tohire$ is not set, the optimal action is to stop hiring, aggregate collected answers, produce the prediction, and go to a termination state. Otherwise, hire worker $s.W$, collect his answer a_w and move along the edge $< w, a_w >$ to next state s', keep repeating this process until we reach a termination state. The core idea of this two phases approach is illustrated in Figure 4.2 with a simplified example.

In Figure 4.2, we set horizon to 2 and number of samples to 20. Each circle is a state and the information on top of each edge between two states shows the worker and his reported answer which lead the execution path in a sample from one state to the other. In every state, the two lists are hired workers and collected answers to reach it. The number is visitation, which is the number of samples that ended up passing through each state. Note that in our approach we are always hiring the same worker from a given state, and the reasoning is

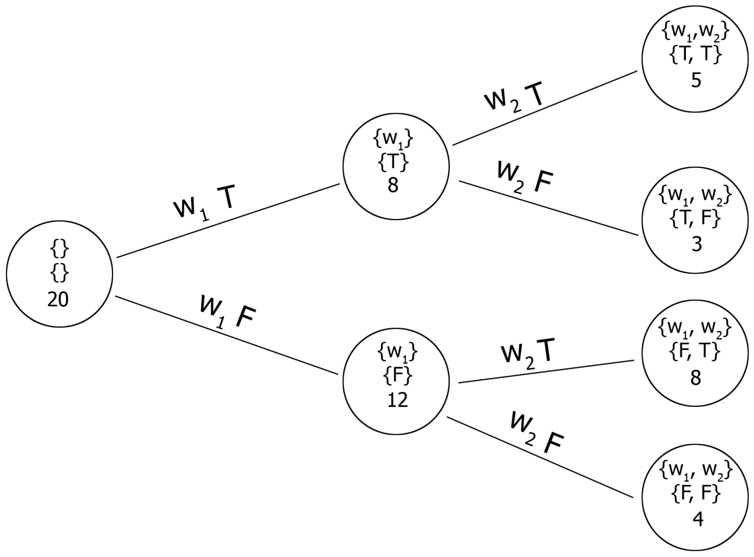

Figure 4.2: An Example of MDP and Monte Carlo Planning

explained in next section. In this example, 8 out of the 20 samples reported answer "False" from initial state, while the other 12 reported "True". The visitation number at every state always equals to the sum of visitation numbers of its children. In evaluation phase, we always start from the terminal states. Suppose state $< \{w_1, w_2\}, \{T, T\} >$ has utility 5 and state $< \{w_1, w_2\}, \{T, F\} >$ has utility 3 from their aggregated opinions calculated using our reward function for system's belief, then we can calculate utility for state $< \{w_1\}, \{T\} >$ which is their parent. For state $< \{w_1\}, \{T\} >$, the value of information for hiring the next worker w_2 is a proportional sum of its children's utility $\frac{5}{8} \times 5 + \frac{3}{8} \times 3$, plus the utility of hiring worker w_2 which is the benefit of w_2's quality improvement minus the hiring cost, and subtract the utility of aggregating collected opinion $\{T\}$, as we defined in 4.12. If the value of information is negative, $< \{w_1\}, \{T\} >$'s utility is set to the utility of the aggregated opinion and its *tohire* pointer is set to $NULL$. Otherwise it's set to the value of information plus utility of the aggregated opinion (this addition is to remove the third factor in 4.7, as we are not stopping hiring). This time *tohire* is set to w_2 as hiring worker w_2 yields a better utility. Similarly we can calculate the utility for state $< \{w_1\}, \{F\} >$, and once the utility for $< \{w_1\}, \{T\} >$ and $< \{w_1\}, \{F\} >$ are available we can calculate the utility for the initial state and complete the evaluation phase.

4.3 Challenges and How to Tackle Them

There are a few challenges the system may face and we need to take care of them before we can move onto present the actual algorithm. First of all, as we want our hiring system to run in real time to guide task assigners on how to hire crowd workers, it must also be responsive to meet time constraints.

4.3.1 Runtime Limitation

The Monte Carlo planning algorithm proposed above faces some runtime challenges. Unlike Kamar's MC-VOI, where each state has only two possible actions and $|A| + 1$ children states, hiring m different workers leads to up to $m|A| + 1$ states in our setting. Even for a relatively small but realistic number of m, the nodes to visit in sampling grows quickly to become impossible to solve within reasonable amount of time. In fact, the size of the sampling tree is $O((m(|A| + 1))^l)$, where l is the horizon (maximum number of workers to hire). Therefore, we need a filtering phase to reduce the number of workers to consider when sampling execution path. In fact, we can simplify the procedure of deciding which worker to hire at each state by ranking all available workers based on their quality and potential.

Overall, we want a ranking mechanism that orders available workers based on a combination of their immediate quality for short term benefit and future quality for long term projection. The latter can be estimated by using a crowd worker's estimated cumulative quality curve. For worker w at task x, his future quality after n tasks is

$$q^*_w(x + n)$$

Therefore, a worker's priority value at task x with a projection into n future tasks can be defined with the following weighted function:

$$prio_w(x, n) = \omega_q q^*_w(x) + \omega_{proj} q^*_w(x + n) \tag{4.13}$$

where ω_q and ω_{proj} are weights determining if we are more interested in accuracy of the current task, or training a high potential worker for future. Moreover, we have a constraint of $\omega_q + \omega_{proj} = 1$ to make sure $prio_w(x, n)$ is within the range between 0 and 1 and can be treated as a mixed quality value, too. The task assigner can set n to the estimated number of future tasks, or a very large number if he is unsure about the exact number but he is not ending the tasks any time soon. In our experiments and simulation tests, we set these two weights equally to 0.5 by default. On the other hand, n is set to the number of tasks left. For example, if the task owner has 500 tasks in total and have already finished 100, n is then set to 400, and is updated after each task being complete.

As a result, at each task, we can rank all available workers according to their priority values, and always pick the ones at the top for Monte Carlo sampling, which reduce the number of children at each state to $m+1$ and the size of the sampling tree to $O((m+1)^l)$. For consensus tasks of only two possible outcomes, this is just a 3-ary tree and can be sampled within a very short amount of time given a reasonable length of horizon. Note that this projection approach is taking advantage of crowd workers always being available, as a hired workers can be hired for every future task. We will try to add some randomization when we relax the availability constraint in next chapter to cope with situations where a fast learner may not be present at all time.

4.3.2 Quality Capping

Recall from Section 3.3 in Chapter 3, when the worker pool's quality is low, it introduces a lot of noise to our aggregated opinions when we are using them to update workers' performance and learning curve. Moreover, when the number of each worker's completed tasks is low, the estimated learning speed and quality from his learning curve may not be accurate enough. There are two approaches to handle this problem. The first one is to increase the number of training tasks each worker has to perform before he can start working on real tasks. As we have ground truth answers to all training tasks from gold standard, this will remove the noise introduced by incorrect predictions. Furthermore, more training tasks means workers have higher starting qualities and we have more data samples to kick start our linear regression test, and the estimated learning curve will as a result become more accurate, too.

However, we also want to deal with the inaccuracy of our learning curve model systematically. So we setup a simulation to analyze the difference between workers' estimated qualities and their true qualities. For this simulation, we created 1000 workers with learning speed r drawn from truncated normal distribution $R(x) = f(x; 50, 10, 0, \infty)$ and prior knowledge p drawn from truncated normal distribution $P(x) = f(x; 65, 10, 0, \infty)$ which is the same as what we did in Section 3.3. Every worker are requested to work on 200 binary tasks. The results are shown in Figure 4.3.

In Figure 4.3, the average worker quality curve represents the average true quality across all 1000 workers at any task, while the highest worker quality curve is the highest true quality among those 100 workers. Average estimated quality and highest estimated

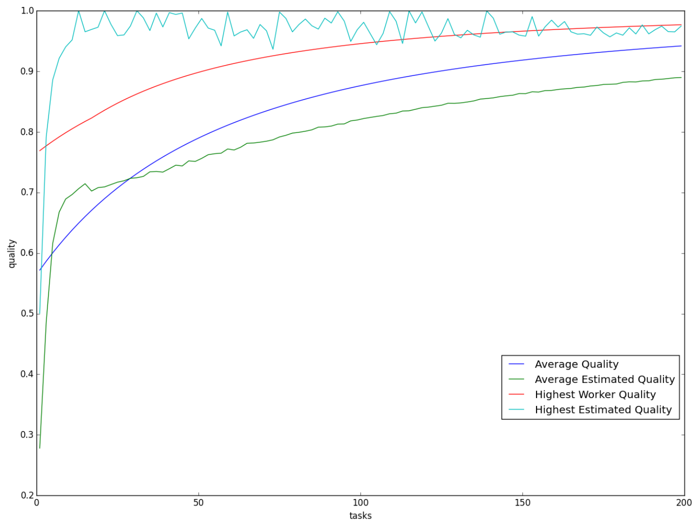

Figure 4.3: Difference between Actual and Estimated Quality

quality curves are on the other hand generated from workers' estimated learning curves. As we can see, the highest possible estimated qualities stay around the high 90%s, quite consistently, while the highest actual worker quality is improving from 80% to high 90%s, in our given population. After 200 tasks the estimated quality will become very accurate. On the other hand, the average estimated quality is very close the the average true quality. This is because at the beginning some workers may get lucky (outperform his true quality) or unlucky, but for the entire population, they offset each other and yield a reliable estimated quality if we take the averaged value across everyone. In the end, all the curves will merge after thousands of tasks, but we think 200 tasks is a good threshold as the difference is small enough, and the quality estimation is within a manageable range.

To take advantage of this observation, we introduce a penalty to workers' estimated qualities when the number of samples are small. We limit the maximum and minimum estimated quality to a certain range from the average value, based on number of tasks that have been completed. From our observation in Figure 3.9, we can see that after 200 tasks the estimated quality will become very accurate. As a result, 200 tasks is a good threshold; after it the estimated quality can be directly used. Before 200 tasks, to put everything in numbers, we use the following capping function to cap a worker's estimated quality based on Figure 3.9.

$$cap(x) = 0.004x + 0.2 \tag{4.14}$$

where $cap(x)$ defines the capped quality range and x is the index of current task to do, thus the hybrid quality we use for opinion aggregation and worker ranking becomes

$$q_h(x) = \mathbf{min}(q^*(x), \bar{q}(x) \times (1 + cap(x)) \tag{4.15}$$

where $q^*(x)$ is our estimated quality described previously, and $\bar{q}(x)$ is the average observed quality from all workers up to time. Note that we do not need to cap workers' estimated learning speed, as what we are actually using is their relative ranking among their peers for learning speed, it will not matter if the entire population's learning speeds are overestimated.

4.4 The Algorithm

Now we are finally ready to present our hiring algorithm. The algorithm has two passes: sampling and evaluation. In the first pass, the algorithm will keep sampling execution paths, where an execution path is essentially a path filled by hiring decisions and simulated reported answers from all workers hired along it. At each state, the sampling path is generated with the worker picked by the ranking mechanism and an answer randomized from the transition probability 4.11. We will keep hiring until reach the maximal length of the horizon, thus every single sampled execution path has the same length as the horizon. In the second path, the algorithm evaluates task assigner's utility at each state by aggregating utilities from its children states with 4.12, and determine if the optimal policy is to hire another worker and move onto next hiring state, or stop hiring and move to a termination state. We present the pseudocode of our dynamic hiring algorithm here, and the working Python version can be found at https://github.com/h2oloopan/dc.

Algorithm 1 Dynamic Hiring Algorithm

```
1: procedure INITIALIZE($task$, $horizon$, $samples$, $workers$)
2:     $root \leftarrow$ new State
3:     $root.visitation \leftarrow 0$
4:     $root.utility \leftarrow 0$
5:     $root.workers \leftarrow \emptyset$
6:     $root.answers \leftarrow \emptyset$
7:     for $i = 1$ to $samples$ do
8:         sample($task$, $root$, $horizon$, $workers$)
9:     evaluate($root$)
```

And the two key functions for sampling and evaluation are:

```
1: procedure SAMPLE($task$, $root$, $horizon$, $workers$)
2:     $cursor \leftarrow root$
3:     $cursor.visitation = cursor.visitation + 1$
4:     $hired \leftarrow 0$
5:     while $hired < horizon$ do
6:         $worker \leftarrow$ nextWorker($workers$)                ▷ From priority ranking
7:         $probabilities \leftarrow$ calculate($worker$, $cursor$)               ▷ Use 4.11
8:         $next$, $answer \leftarrow$ nextState($probabilities$)      ▷ Randomize with probabilities
9:         $next.hired \leftarrow worker$
10:        $next.workers$.add($worker$)
11:        $next.answers$.add($answer$)
12:        $next.visitation = next.visitation + 1$   ▷ visitation is default to 0 for new state
13:        $cursor.children$.add($next$)                  ▷ If not already added
14:        $cursor = next$
15:        $hired = hired + 1$
```

```
1: procedure EVALUATE($state$)
2:     if state has no children then                                      ▷ This is a termination state
3:         $prediction$, $belief$ ←aggregate($state.workers$, $state.answers$)
4:         $state.utility$ = getAnswerUtility($belief$)                                           ▷ Use 4.5
5:         $state.tohire$ = $NULL$                          ▷ You cannot hire anyone at termination state
6:     else                                                                ▷ Non-termination state
7:         $voi$ ←calculateVOI($state$, $state.children$)                                        ▷ Use 4.12
8:         $prediction$, $belief$ ←aggregate($state.workers$, $state.answers$)
9:         if $voi < 0$ then                                          ▷ The best policy is to not hire
10:            $state.utility$ =getAnswerUtility($belief$)
11:            $state.tohire$ = $NULL$
12:        else
13:            $state.utility$ = $voi$+getAnswerUtility($belief$)
14:            $state.tohire$ = $state.children[0].hired$                ▷ Hired worker to reach a child
```

The actual hiring part is easy. At any state after hiring a group of workers and collecting their reported answers, we can find a path from the root of the sampled tree and find the corresponding node. And the node's associated *tohire* pointer will either give you the crowd worker to hire, or $NULL$ which means the best strategy is to terminate hiring and return aggregate prediction. Note that after each aggregated opinion is reported, the system will also update each hired worker's learning curve based on ones performance against the final prediction.

4.5 Simulation Results

There are two traditional hiring approaches we are comparing our new hiring mechanism to. The first one is **RandomK**, which is the most commonly used hiring mechanism that randomly picks k available workers and let them work on given tasks. In our simulation, we combine it with majority voting for opinion aggregation, which is also the most common way to generate predictions. The second one we are comparing to is the bandit inspired method **TopK**, which rank all available crowd workers by their observed quality (calculated from dividing number of accurate prediction by total number of tasks completed) and pick the top k workers to work on the next task. **TopK** is also widely used in quality based crowdsourcing applications. We use majority voting to predict final answers when aggregating opinions for our **TopK** method, too. Moreover, for all of our simulation tests, we run each hiring algorithm 10 times and record their average performance. And for spot quality, we smooth it by taking the average of accuracy on previous 4, current, and next 4 tasks. For both **TopK** and **DynamicHiring**, all available workers are first asked to work on 10 training tasks to help the system collect their quality information. In this thesis, a training task is a task where we have ground truth and can update a worker's quality information correctly. In this chapter, a worker may perform a reasonable amount of training tasks for free, while he is get paid for them in next chapter.

We want to see these three hiring mechanisms' performance on worker populations of different quality settings. First, we want to test the case where crowd workers all have a very slow start, or in another word, they are very inexperienced and are asked to work on something relatively new to them. To achieve this, we draw workers' learning speeds from truncated normal distribution $R(x) = f(x; 50, 10, 0, \infty)$, and prior knowledges from truncated normal distribution $P(x) = f(x; 40, 5, 0, \infty)$, thus in average their qualities start from 44% and become 50% after taking 10 training tasks. The learning speeds are set around 50 which is what we got in sleep spindle detection tasks in Chapter 3, so it is representative of a real world problem. We randomly create a total of 1000 binary tasks whose outcomes are only True/False, and all hired agents can only report True or False which is randomly generated based on their inherent quality calculated from his assigned r, p values. Furthermore, we create a pool of 1000 workers from the truncated normal distribution described above, and every single worker has a fixed hiring cost of 1. We use such a large worker pool as we are more interested in situations there are tons of uncertainty.

Nothing else is needed to set for **RandomK** and **TopK** other than the k value. We set $k = 3$ for **TopK** and **RandomK**, as three workers in general can provide very good results in settings where learning speed is fast overall, and as an odd number this removes need for any extra tie breaking. For **DynamicHiring**, we set its hiring horizon to 5 to take advantage of its adaptive hiring. As a result it can hire up to 5 workers for any task. Moreover, we set the weight for aggregate opinion belief to 7, and workers' quality improvement to 100. These numbers are chosen from simulations as they yield consistent results across different task and worker settings. Furthermore, we set the cutoff threshold for opinion aggregation reward function to 0.85 so prediction with belief less than 85% will not yield any reward. The actual results are shown in Figure 4.4 and Table 4.5.

As we can see from Figure 4.4, the performance of **RandomK** is basically random. For 1000 tasks, randomly selected workers in such a large worker pool are not able to get enough tasks to get any significant quality improvement. **TopK** on the other hand yields a very solid performance. After about 200 tasks the predictions are almost 100% correct. In fact, for a worker population around a certain quality (learning speed), picking any 3 fixed workers and sticking with them can produce comparable results even if they were not the

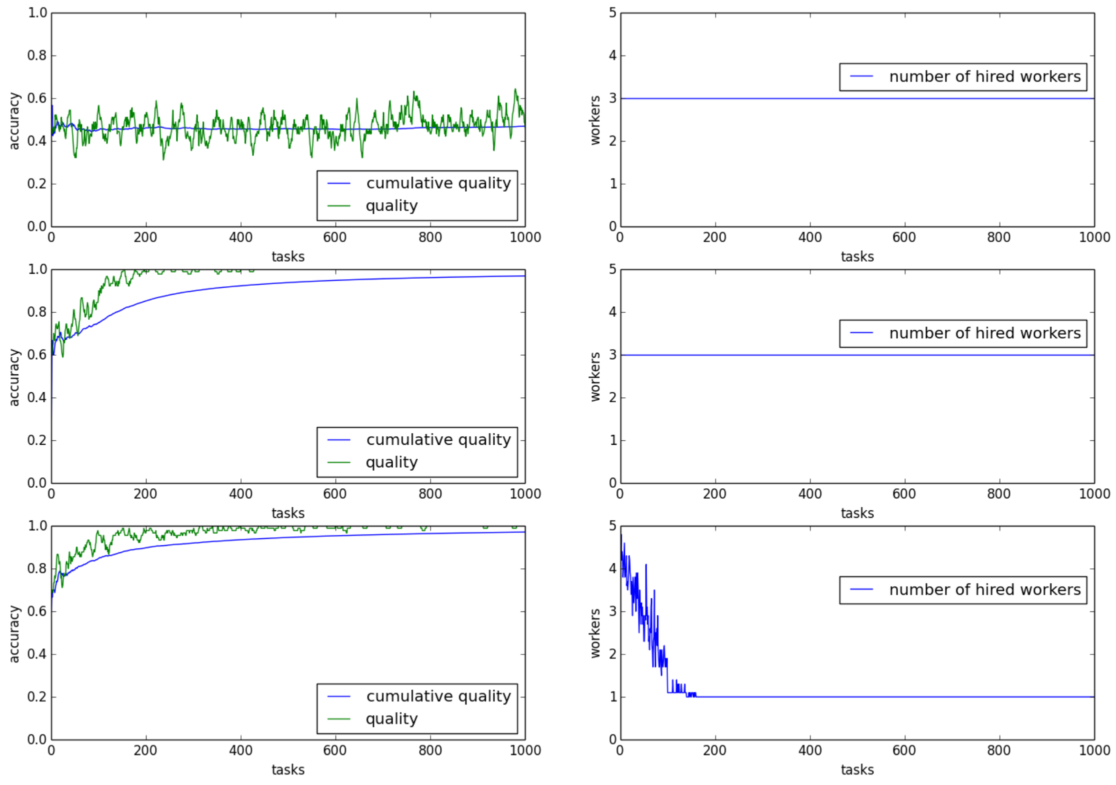

Figure 4.4: Results for simulation with $R(x) = f(x; 50, 10, 0, \infty)$ and $P(x) = f(x; 40, 5, 0, \infty)$

Hiring Mechanism	Correct Tasks	Cumulative Quality	Hiring Cost	Training Tasks
RandomK	468	46.8%	3000	0
TopK	969	96.9%	3000	10000
DynamicHiring	971	97.1%	1200	10000

Table 4.1: Results for simulation with $R(x) = f(x; 50, 10, 0, \infty)$ and $P(x) = f(x; 40, 5, 0, \infty)$

ones with best potential as all workers have a similar learning process. It will face some challenges when we later test on mixed worker population. Our proposed **DynamicHiring** system is slightly better accuracy wise, however the saving on hiring cost is significant as the hiring cost was decreased from 3000 to 1200 (a 60% cost reduction) since it was able to locate and train a high quality worker and take advantage of his presence. In general, after 200 tasks one of the top workers was trained to achieve a quality above the 0.85 threshold, and the benefit of hiring any new workers became marginal. As an adaptive hiring mechanism, **DynamicHiring** was able to hire more workers at the beginning when workers' overall quality is low, reduce number of workers hired when the quality of the top workers are improving, and hire only one worker when a top worker is well trained. This is exactly the behaviour we were looking for.

Moreover, **DynamicHiring** had a better start due to more workers hired at the beginning and more accurate quality estimation from learning curve model. It had slightly more quality dips when there was only one worker hired since aggregate opinion from three workers had a slightly better accuracy. However, such quality difference e.g. 99.5% against 99.3% cannot justify the cost of hiring two more workers for most of crowdsourcing applications. Overall, considering both accuracy and cost efficiency, **DynamicHiring** is much better than the two benchmarks we are comparing it to. On a side note, both **TopK** and **DynamicHiring** require training training at the beginning to collect quality information, which is proportional to the size of the worker pool, and it may introduce additional cost if we are unable to let workers work on training tasks for free. Furthermore, for all simulation tests in this chapter all workers are asked to do the training tasks first even if they were not selected to work on any of the real tasks. This can be enforced in contained settings where we have full control on the worker pool, e.g. we only ask nurses in a hospital to work on sleep spindle detection tasks as a mandatory job. However, it may not fit many other crowdsourcing applications. We will relax this constraint in next chapter when we extend our dynamic hiring system.

We then want to test the three approaches on tasks which are very hard to learn thus most workers have a very slow learning speed while they are somewhat decent to start with. To achieve this, we draw the worker population from $R(x) = f(x; 200, 20, 0, \infty)$ and $P(x) = f(x; 300, 20, 0, \infty)$ such that in average the agents start with qualities around 60% and it in average take one 500 tasks to reach a quality of 80%. Note here we are using larger sigma values for the two truncated normal distributions so that the worker population is more diversified. We keep all the other settings for all three mechanisms unchanged.

The result can be seen in Figure 4.5 and Table 4.5. Not too suprisingly, **RandomK** has a performance about workers' average quality like before. **DynamicHiring** still has the best quality-wise performance, while **TopK** is a very close second. We can improve **TopK**'s

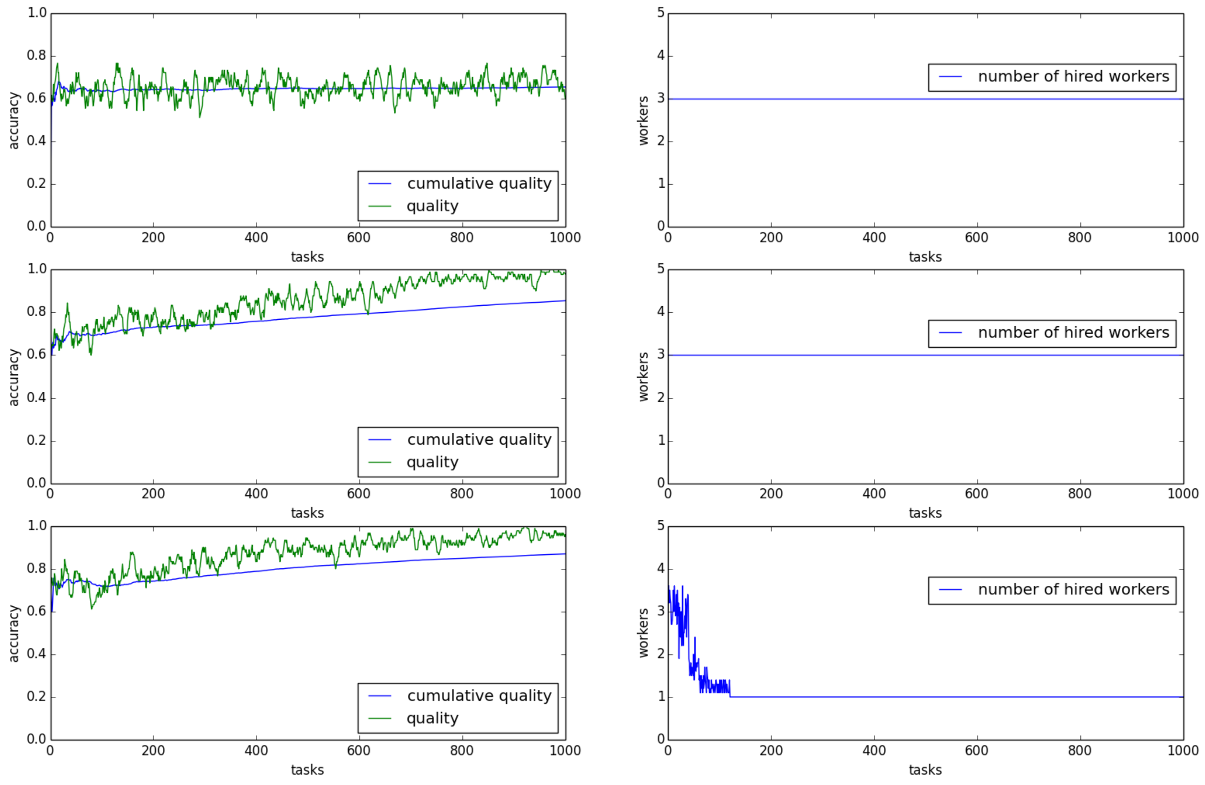

Figure 4.5: Simulation Result for Slow Learning Workers

performance by increasing k. In fact set $k = 5$ will give it a 91.4% cumulative quality after 1000 tasks. But we don't think such performance improvement are worth hiring 2000 more workers. Moreover, it only cost **DynamicHiring** 1107 (a 63.1% cost reduction) to achieve the best performance. Note that **DynamicHiring** hired fewer workers now for the first few tasks (not reaching 5 workers) due to workers' better starting quality. However, in this simulation test, we are not hiring more than one worker after about 150 tasks where the reduction in hiring actually happened quicker than our last simulation which was not expected. It was because of the system's suffering from overestimating worker quality as when workers are learning slowly, the estimated quality of workers are converging slower to the true value than settings where workers can learn faster. We may increase the number of

Hiring Mechanism	Correct Tasks	Cumulative Quality	Hiring Cost	Training Tasks
RandomK	654	65.4%	3000	0
TopK	854	85.4%	3000	10000
DynamicHiring	869	86.9%	1107	10000

Table 4.2: Simulation Result for Slow Learning Workers

training tasks or belief threshold to mitigate this problem, but the quality improvement is not significant as the worker with top potential becomes very good after 150 tasks anyways. In fact if we increased the belief threshold to 0.95, then hiring one worker would be delayed to after about 250 tasks, but the cumulative quality improvement was only about 1% in the end.

We then want to study the setting where workers start with a low quality but are improving rapidly. To simulate this scenario, we set up with $R(x) = f(x; 30, 5, 0, \infty)$ and $P(x) = f(x; 10, 1, 0, \infty)$, or in another word, it takes a worker in average 30 tasks to achieve 50% quality while he has a starting quality of about 25% which is even worse than coin flipping. The results are shown in Figure 4.6 and Table 4.5.

Hiring Mechanism	Correct Tasks	Cumulative Quality	Hiring Cost	Training Tasks
RandomK	228	22.8%	3000	0
TopK	988	98.8%	3000	10000
DynamicHiring	986	98.6%	1333	10000

Table 4.3: Simulation Result for Fast Learning Workers

Clearly, this setting really hurts **RandomK**, as when workers are randomly selected from a large pool of poor starting quality, no single one can get enough tasks to improve even though their learning speed is fast. As a result, what we get are basically just their starting quality, with slight improvement towards the end. For 1000 workers, 1000 tasks and 3 hired workers per task let each one get only about 3 tasks to improve his quality.

On the other hand, **DynamicHiring** has a similarly high performance to **TopK**, thanks to fast learning workers and less quality overestimating (as workers become good quickly enough). Note that in this scenario **DynamicHiring** hired more workers for the first few tasks (4-5 workers for about the first 8 tasks), as the workers' starting quality is lower and more workers are needed to reach the belief threshold. The cost saving is still

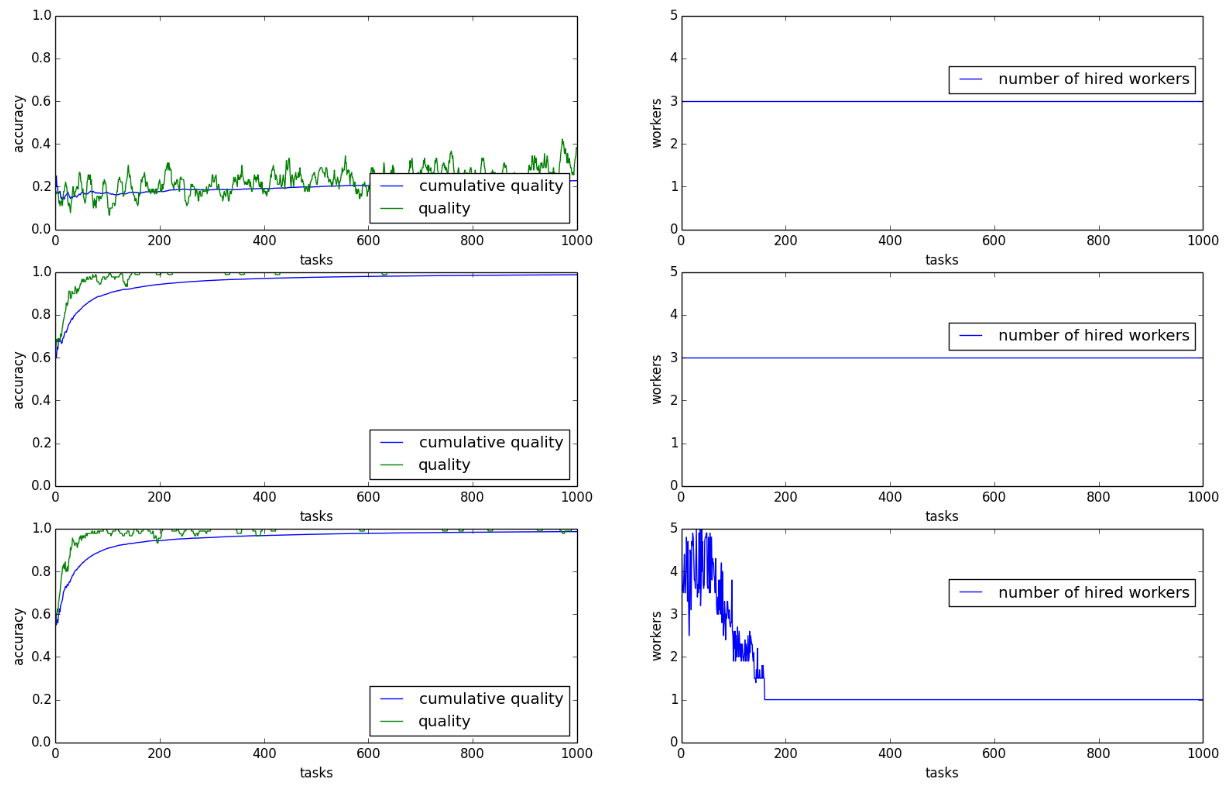

Figure 4.6: Simulation Result for Fast Learning Workers

significant as usual. It achieved the same performance level with only 1333 hired worker (55.6% cost reduction).

Last but not least, we want to test a situation which is very challenging to **TopK** and test if **DynamicHiring** is adaptive and robust enough to handle it. This time we create a worker population where majority of workers are very slow learners with a better starting quality, while a small group of workers starts poorly but are faster learners and can outperform others quickly. This will mislead **TopK** to hire slow learners with better starting quality thus unable to take advantage of faster learners and their future high performance. We created a worker pool where 900 workers are drawn from $R(x) = f(x; 2000, 10, 0, \infty)$

and $P(x) = f(x; 2000, 10, 0, \infty)$ and 100 workers are drawn from $R(x) = f(x; 80, 5, 0, \infty)$ and $P(x) = f(x; 20, 5, 0, \infty)$. 90% of the worker population are slow learners with an average starting quality of 50%, while the other 10% are fast learners but they start with an average quality of 20%.

This setting will cause a lot of troubles to **TopK** as even after taking 10 training tasks, the average quality of fast learners are still less than 50%. And since the population of fast learners is small, it's almost impossible for one of them to get lucky and perform better than the slow learners in the 10 training tasks. However, recall our earlier discussion and Figure 3.9 that the estimation of workers' learning speed is actually worse than the estimation of their quality. If we stayed with 10 training tasks, for such a large worker population, some of the slow learners' learning speed may get overestimated and surpass faster learners. To deal with this, we need to increase the number of training tasks to reduce such overestimation. After some testing, we found that 30 training tasks are good enough to distinguish fast learners from slow learners, although after completing 30 tasks, slow learners still have a better average quality than their fast learning counterparts (50% vs. 38%). To make it fair, we increased the number of training tasks for **TopK** to 30, too. And the simulation result can be seen in Figure 4.7 and Table 4.5.

Hiring Mechanism	Correct Tasks	Cumulative Quality	Hiring Cost	Training Tasks
RandomK	459	45.9%	3000	0
TopK	587	58.7%	3000	30000
DynamicHiring	884	88.4%	1134	30000

Table 4.4: Simulation Result for Workers of Mixed Learning Speed

We can see from the results, **RandomK** basically just yield a performance of around 50% as it's mostly likely picking up slow learners which are just coin flipping, and their quality will not improve as most workers' learning speed is too slow. Like what we had expected, **TopK** are also stuck with slow learners. Although their quality is still improving given more tasks, this time the improvement is slow and at the end it's only getting about 60% tasks correctly. On the other hand, **DynamicHiring** is able to find and hire the fast learners as it's looking for a weighted combination of worker's current and future quality, and finding one of the fast learners makes it possible to reduce the hiring cost as well as perform better once the worker surpasses all of his peers. However, we need 30 training tasks to wane down noise in linear regression tests which may be a bit too much in real world applications (at least if we are not paying for training tasks). In next chapter we want to explore the possibility to add randomness to the worker selecting algorithm in our

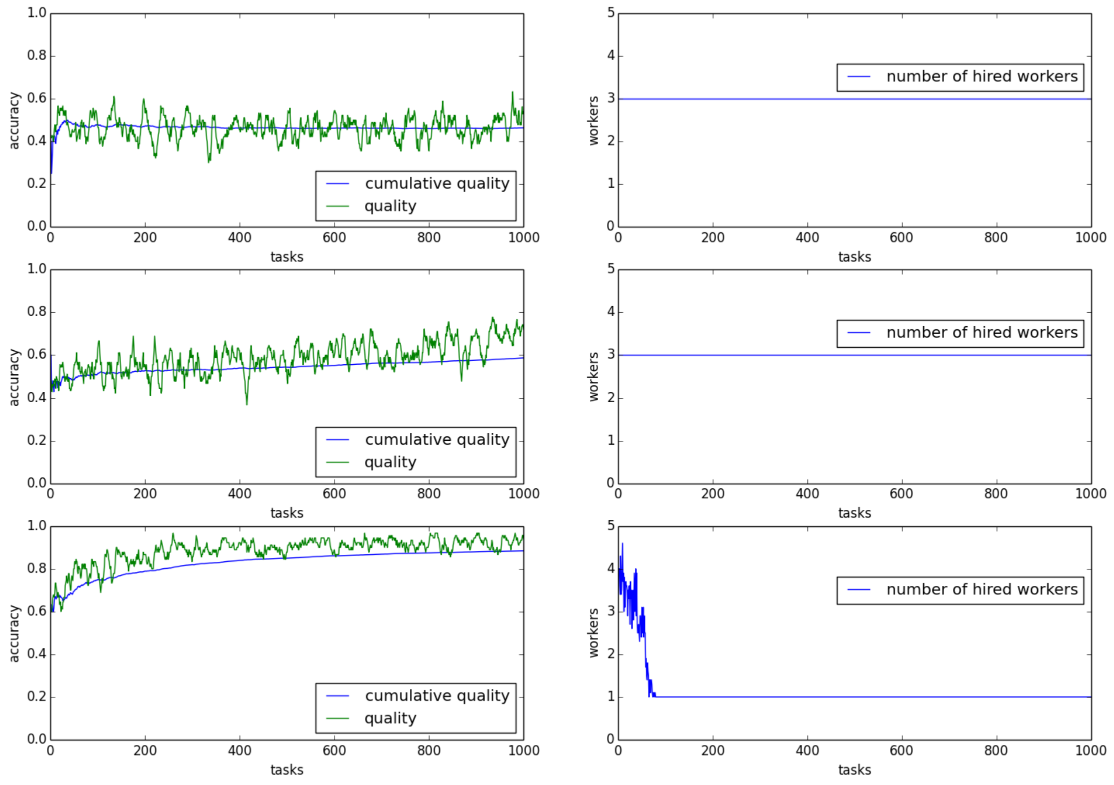

Figure 4.7: Simulation Result for Workers of Mixed Learning Speed

system so there are always some workers hired regardless of estimated quality and learning speed to tackle this problem. We will compare the modified system and see if it could reduce the number of training tasks needed in this type of extreme cases.

4.6 Test on Sleep Spindle Detection Data

After doing simulation tests on different worker population, we want to see how our proposed dynamic hiring system perform against real world data. In this section, we will compare the performance of **RandomK**, **TopK**, and **DynamicHiring** against the data we collected for sleep spindle detection tasks in Chapter 3. To fit our setting of binary tasks with only two outcomes, we will only test the recall scores of all expert annotated spindles. In another word, we let each sleep spindle found by the expert (ground truth) be a task, where each worker either reported that spindle (answered task correctly) or ignored that spindle (answered task incorrectly). In this experiment, we test our hiring systems against all worker data, including the ones we found are suspicious, to make sure the results are more realistic.

To setup our experiment formally, we take the first t tasks/spindles as training tasks, as workers were doing them first. We have 15 workers in total. As the tasks were not overwhelmingly complicated and workers did pretty well according to results in Chapter 3, we set $k = 3$ for both **RandomK** and **TopK** as before so we are not hiring excessive workers. This time we decrease the horizon of **DynamicHiring** to 3 too so we will at most hire 3 workers with it, to test its performance with even less exploration at the start. For **RandomK**, we select 3 workers from the 15 randomly for any given task. For **TopK**, we compare the averaged cumulative quality of all 15 workers for the first t tasks, and pick the top 3 for all the remaining tasks. For **DynamicHiring**, we also use the first t tasks for training. To compare all these hiring mechanisms fairly, we only test the result for tasks after first t for **RandomK**, thus the results are all shown against the exact same tasks. Moreover, for the quality curve shown in the following figures, each plotted point at task x is averaged from its 10 neighbour (including and centred at itself) to make the curve smoother to present the general trend.

In our first experiment, we set $t = 15$, to show the performance against a minimum number of training tasks for both **TopK** and **DynamicHiring**, The first 15 sleep spindles are found in the first 7 windows, which is a realistic length for training tasks. After removing the first 15 sleep spindles, there are a total of 86 tasks left. The results are shown in Figure 4.8. At the end, **RandomK** got 61 tasks correctly (70.93% accuracy), while **TopK** correctly answered 71 (82.56%). They both hired 258 workers in total. On the other hand, **DynamicHiring** answered 64 tasks correctly (74.42%), while only hired 111 workers.

Overall, **DynamicHiring** achieved a slightly worse performance with less than half of workers hired by **TopK**. If we look at Figure 4.8 carefully, **DynamicHiring** found its top agent after about 20 more tasks and stopped hiring more workers as it thinks that agent's performance will stay high and the benefit of more hiring is just marginal. However, that specific agent's performance actually dropped towards the end and the performance of **DynamicHiring** therefore suffered, too. The performance of such worker is shown in Figure 4.10, and we will discuss it later.

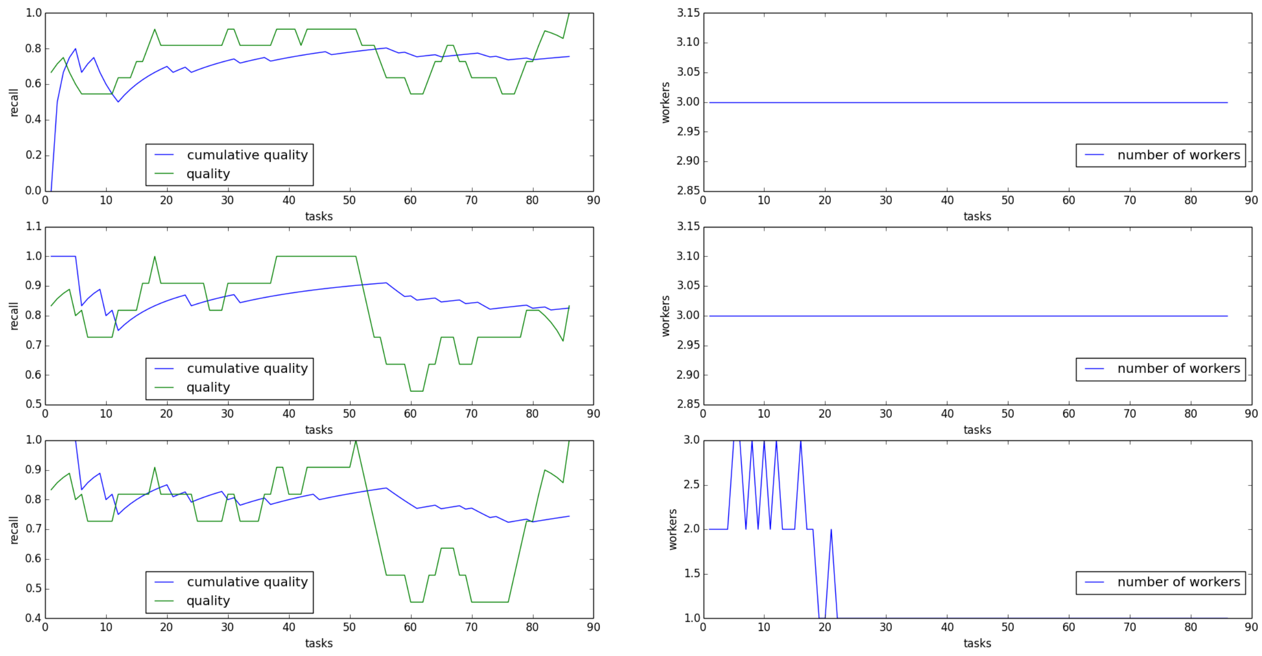

Figure 4.8: Hiring Mechanisms Tested Against Real Data, 15 training tasks

In our second experiment, we increase t to 20, to give **DynamicHiring** more tasks to train. As we have shown previously from simulation even such a small increment in training tasks should improve the performance of our hiring system vastly due to more accurate estimation and projection. To make the comparison fair, the number of training tasks for **TopK** is also adjusted to 20, and **RandomK** is starting from the 21st task, too. Note that now we have 81 tasks in total for our comparison. The results are shown in Figure 4.9.

Now **RandomK** got 62 or 76.54% tasks correctly while **TopK** answered 66 or 81.48% correctly. This time, **DynamicHiring** outperformed both by a big margin by finishing with 72 or 88.88% correctly answered tasks. Moreover, **DynamicHiring** only hired 81 workers as it found a top performer right in the training tasks while the other two hired 243. In this experiment, our proposed hiring mechanism achieved great performance while only hiring one third of the workers needed for the other two hiring mechanisms. The

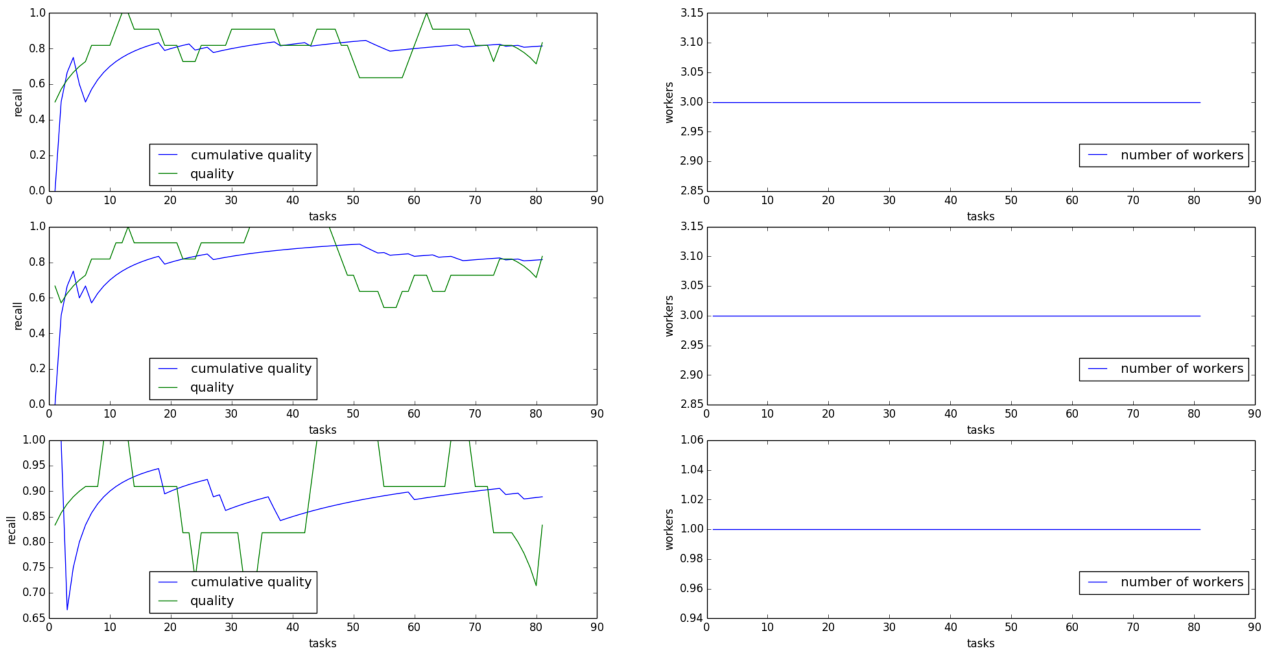

Figure 4.9: Hiring Mechanisms Tested Against Real Data, 20 training tasks

result also verified our observation in simulation tests that the more training tasks we have or more information collected at the beginning, the better performance **DynamicHiring** can yield.

Performance of the two solely selected workers at the end by these two experiments is shown in Figure 4.10. The graph on the left is the recall performance of the worker hired at the end for our first experiment with 15 training tasks, while the right one is from the second experiment with 20 training tasks. We denote the one on the left as worker one and the right one as worker two. As we can see, the performance improvement for worker one was significant for the first 15 tasks (almost from 0.5 to 0.8), and that was the reason our hiring system overestimated his learning speed (and quality in consequence). For worker one, even after a few more tasks, the projected learning curve from linear regression is still far above the worker's true performance. When more training tasks are conducted in our second experiment, the system had more information to analyze

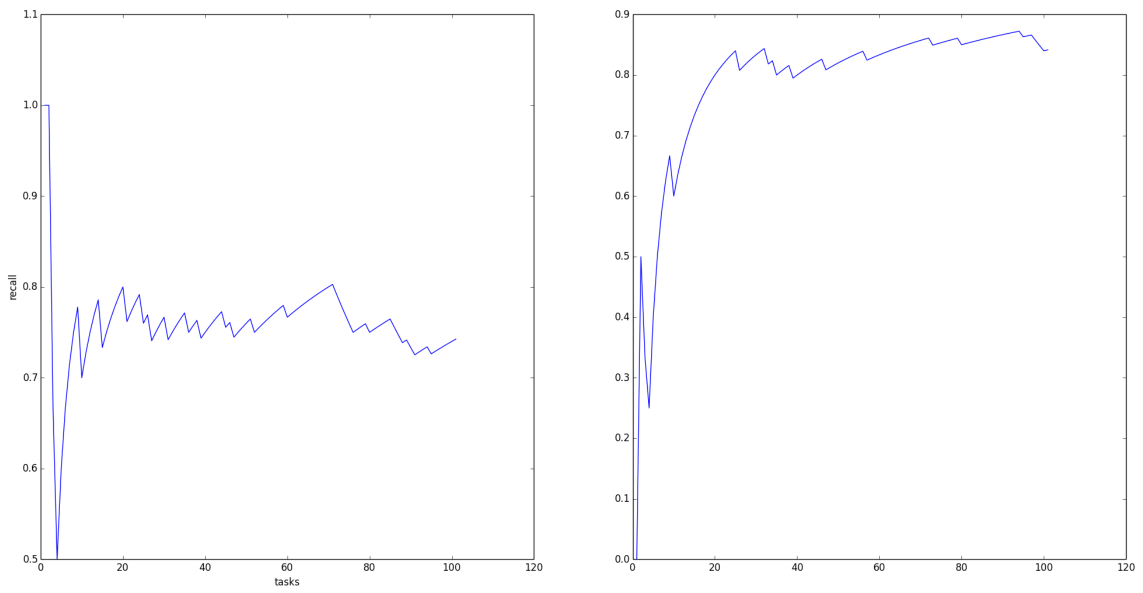

Figure 4.10: Hired High Performance Worker for 15 and 20 training Tasks

and was able to find the true top performer worker two. This is the biggest problem for **DynamicHiring**. When the number of training or initial information collection is not enough, the estimated learning curves for workers may not be accurate enough to truly predict their future performance. As a result, we may need to use our knowledge about the crowdsourcing tasks and worker population to set a proper number of training tasks to fully utilize the power of our **DynamicHiring** mechanism.

Overall, the results from both our simulation test and real world data prove the validity and advantage of our proposed dynamic hiring system, where it can estimate crowd workers' quality better, project their learning process, as well as give us some insight into the properties of tasks we are working on and the worker population. By using dynamic hiring process, we are able to cut down hiring cost significantly while still providing solid accuracy performance for consensus tasks, especially for a diversified worker population where workers' inner learning process is mixed.

Chapter 5

Extended Hiring Process

In this chapter we want to relax some of the constraints and assumptions we have made previously, and extend our dynamic hiring system to work with more realistic settings where workers are not always available, can only work on a small set of training tasks for free, and the task assigner is not allowed to ask workers he is not willing to hire to work on training tasks.

5.1 Dealing with Unavailable Workers

In most real world crowdsourcing applications, no worker is always available to be hired for any given task. In fact, only a small amount of the worker pool is available at times. Therefore, the hiring system may not be able to rely on a few high quality workers as their availability may indeed be low. As a result, we want a system that can train a larger group of workers or train the best workers among the highly available group. The system must make smart tradeoff between current quality, future quality, and availability among workers. To simulate this setting, we add a availability property (a probability) to each simulated worker, and at each task, a worker's presence is randomly decided by his availability probability. For example, if a worker's availability property is set to 0.3, for any given task, he is 30% likely to be present while the other 70% of time he will be absent. To make it more realistic, the availability of worker population is drawn from a truncated normal distribution $A(x) = f(x; 0.05, 0.04, 0, \infty)$, so in average only 5% of workers are available at a given task, while different workers have a slightly different availability (with some up to 10%). Note that in our setting workers' availability is independent from their learning speed and prior knowledge. We think this setup is more realistic as we cannot assume high quality workers are more available, nor vice versa. In our simulation tests we will still run each hiring mechanism 10 times to make sure we are getting the general trend and mixing everything up.

Since now workers are no longer always available, we have to make some changes to our benchmark hiring mechanisms. For **RandomK**, we will first gather all available workers before each task and randomly select k to hire from the available ones. For **TopK**, as we cannot just let all workers work on training tasks at the same time, now we keep track of each worker's average quality and number of tasks he has completed. For any new task, we let all available workers do the training tasks if they have not completed them. We then rerank workers based on their latest observed quality before making hiring decisions for the new task. As a result, the top k workers picked for one task may no longer be the ones for next task. This is a more practical setting to simulate applications in the real world. For example, on MTurk, you can not let a worker perform any tasks if you are not hiring him, and you are never able to evaluate the entire potential worker population because only a small portion of it is online at a given time.

For **DynamicHiring**, we also have to consider each agent's availability when we are looking for one to hire and train. One problem we are facing now is that availability stats are actually hard to measure, as it may change from time to time, and is subject to many unknown factors. To tackle this problem, we assume there is a certain general availability for the entire worker population, and set the availability of each new coming worker to this general availability by default. If we have some historical data, this general availability can be estimated from it. For example, if in the past workers are in average 30% available for a task, we can then set this general availability to 0.3. Then we need to update each worker's own availability when his presence information is known to the system after each task. We use the following function $f(w, x)$ to estimate a worker w's availability at x-th task:

$$f(w, x) = \frac{M + p(w, x)}{N + t(w, x)} \tag{5.1}$$

where function $p(w, x)$ gives the number of tasks where w showed up between his first appearance and the x-th task, while function $t(w, x)$ gives the total number of tasks from w's first appearance till x. The system will keep track of a worker once he become available for the first time for some task. The two numbers M and N are determined from our understanding and assumption of the worker population. For instance, if we think 0.3 is a good default availability for the population, we can then set $M = 3$ and $N = 10$. Changing M and N will change how fast the system is update workers' availability information.

However, since we won't be hiring a lot of workers for each task, we may not be able to get all workers to work on enough tasks to collect their availability information. For instance, for 1000 tasks and 1000 workers, if in average we only hire 3 workers for each task, even if we distribute tasks evenly, each worker will only get 3 tasks. Moreover, if we only select top agents, a lot of workers may never get picked by the system to learn their information. Thus, our system must make a smart tradeoff between the cost of hiring and information exploration.

5.2 Dealing with Fewer Training Tasks

Another assumption we made in last chapter was that all workers are asked to work on some training tasks first for the system to learn their quality information, even if they are not hired for any real tasks. In real world applications, this is not practical. On crowdsourcing platforms like MTurk, no worker will work on any task without being compensated, and it is unrealistic to let entire worker population work on a set of training tasks first without hiring majority of them. In fact, we can not draw clear boundary between training tasks and real tasks. In this chapter, although we still call them training tasks, we consider them as real tasks and workers performing them are getting paid as they are working on real tasks (they will not be told what kind of tasks they are working on, so the only real difference is that we have ground truth for these training tasks). In our system, a crowd worker only need to complete training tasks once. If he gets hired again and the system knows he has already completed training tasks, he will be assigned to work on the real task right away. This way we can cut down the number of total training tasks, which could save even more when compared to **TopK**, as **TopK** must use training tasks to sample all available workers to find the top k performers.

Overall, the system now has two goals when hiring workers, one is to find currently top performing workers to make an accurate and reliable prediction, while the other is to explore the worker population where quality and learning curve information is still unknown. Our existing system was able to handle the first one, while we need to add an exploration phase to it to determine the latter. To achieve this, we are going to replace the static priority ranking heuristic in Monte Carlo sampling with a randomized approach which decide between hiring existing workers and exploring new workers based on weights assigned from a combination of workers' current quality, future quality, and availability.

5.3 Extended Dynamic Hiring System

The core of our dynamic hiring system stays the same, as we still use MDP to model the entire hiring process, keep the Monte Carlo sampling and backward evaluation phases, and comparing VOI to make decisions about hiring and termination. There are only two changes we are making for the new system: how we let workers do training tasks, and the ranking mechanism to order all available workers before we feed them to Monte Carlo Planning. To find a sweet spot between accuracy of estimated quality information from linear regression test and hiring cost on training tasks, we set the number of training tasks to 20 for the new system (**TopK** is set to sample 20 training tasks, too). Now we introduce a confidence score to replace our old priority score in last chapter. This time we still rank workers by this score, but workers with higher scores may not always be picked before their peers with lower scores, as randomization is going to be added. The confidence score is a weighted measurement calculated from a worker's current quality, future quality, and availability.

At the very beginning, we assume every worker is of the same quality, learning speed, and availability, to make sure any one is equally likely to get picked. In fact, for crowdsourcing applications with historical data (or where we can collect some information beforehand), we should at least have some basic knowledge about the problem domain and worker population, e.g. a rough idea of workers' quality range and general learning speed. We can then use the information we have to setup a baseline which we can reasonably assume to fit many workers. For example, we may assume an average worker has a learning speed of $r = 80$ and prior knowledge $p = 80$ (thus a starting quality of 50%). These numbers are then used to calculate our baseline score. For any new worker that the system had not collected any information for, it will assume the worker follows these values. As a result, any worker the system had hired before with a better performance than this baseline is likely to outperform a random new worker picked from the population, and should more likely be picked by the system when it's looking for the next worker to hire.

Another reasonable assumption is, after hiring many workers, the likelihood of finding some new worker who is better than the best existing ones will become smaller. Therefore, after collecting information of enough workers such that the presence of recorded good quality ones can cover most of the tasks, the incentive to explore more unknown workers diminishes. However, since there is noise associated to every step of the hiring mechanism and none of the estimated numbers are absolutely accurate, we still want it to enter exploration phase and evaluate new workers from time to time, but with a decreased probability.

To put everything into perspective, for each task, the system first checks all available workers. If none of them were seen before, it hires someone new randomly. Then if all the ones hired before have confidence scores worse than the baseline score, it hires new workers randomly, too. But this time the system will lower its expectation of worker population, e.g. by lowering the baseline score. For all the returning workers with scores higher than the baseline, there are three problems. First, we do not know if he will come back again. Second, some of the new workers may outperform the baseline and have a greater potential Third, our estimation might not be perfect. The last one is taken care of by our quality capping policy described in Section 3.3.2 so that workers with fewer data samples have a tighter upper bound of allowed estimated quality. To deal with the first two, the system calculate a probability based on a worker's confidence score so that it may actually ignore the worker with higher score and enter exploration phase to randomly pick an unknown worker.

The confidence score s_w for worker w at task x is defined as:

$$s_w(x) = (1 - f(w, x)) \times q_w^*(x) + f(w, x) \times q_w^*(x + n \times f(w, x)) \tag{5.2}$$

where $f(w, x)$ is worker w's availability at x as defined in 5.1, and $q_w^*(x)$ is w's estimated quality function as before. For future quality projection, we multiply number of estimated future tasks n by worker's availability as the worker will only be available for a portion of these tasks. And the weights of current and future quality are now tied to worker's availability as the importance of worker's future quality is related to how available he is in the future. Note that the confidence score is a quality score and lies in the range of 0 to 1. The system updates all recorded workers' confidence scores after every task.

After ranking all existing workers according to their assigned scores, we still use the same hiring mechanism described in previous chapter. However, this time we have another option of exploring new workers. We derive a probability of staying with the selected worker from his confidence score as

$$Pr(s_w) = \frac{4s_w - 2}{\sqrt{1 + (10s_w - 5)^2}} + 0.6 \tag{5.3}$$

where s_w is the score in 5.2 and $Pr(s_w)$ is an algebraic sigmoid function (plotted in Figure 5.1) with a range from 0.2 to 1 for $0 \le s_w \le 1$.

The reason we are mapping confidence score to probability with this sigmoid function is that we want the system to stay with workers whose confidence score is high (in this case ≥ 0.6), and leave workers whose confidence score is low (in this case ≤ 0.4). Moreover, there should be minimal change above and under these two thresholds as the quality (remember confidence score is weighted quality score) change in these two ranges should not affect

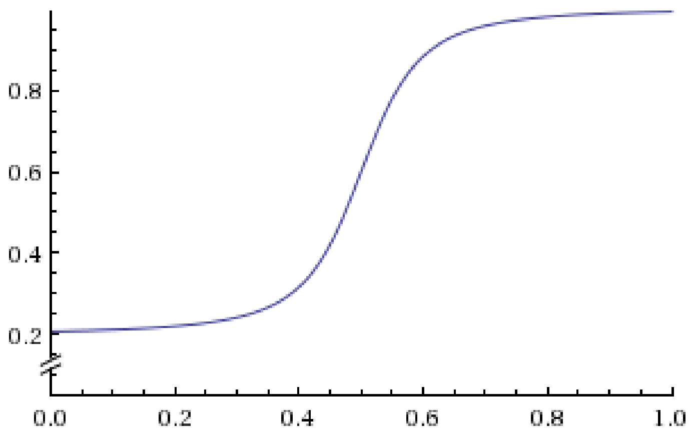

Figure 5.1: Illustration of the Sigmoid Probability Function

the system's behaviour much (worker's quality is either too low or too high). There is still the problem of overestimated quality, but the population as a whole should even out and the ones with higher estimated quality are still more likely to outperform their peers. Note that the probability has a lower bound of 0.2, because we still want a worker with low confidence score to have a chance to be selected as his confidence is still greater than our assumed baseline for the worker pool. The new algorithm (pseudo-code) to pick next worker to hire in Monte Carlo sampling is shown below in Algorithm 2.

Algorithm 2 Worker Picking Algorithm

```
1: $rankedExistingWorkers = rank(existingWorkers)$    ▷ From scores defined in 5.2
2: $nextWorker = getBestExistingWorker(rankedExistingWorkers)$
3: if $nextWorker = NULL$ then
4:     $nextWorker = getNewWorkerRandomly()$    ▷ pick an unknown worker randomly
5: else
6:     if $nextWorker.score < baseline$ then    ▷ baseline was estimated for population
7:         $nextWorker = getNewWorkerRandomly()$
8:     else
9:         if $random(0,1) > getProbability(nextWorker.score)$ then    ▷ Use 5.3
10:            $nextWorker = getNewWorkerRandomly()$
```

There are no other major modifications made to the system described in previous chapter. Since we do not have real world (sleep spindle detection) data for this setting as our experiment on MTurk doesn't support returning workers, we will use simulation tests to check the performance of this modified **DynamicHiring** mechanism against **RandomK** and **TopK**.

5.4 Simulation Results

This time we decrease the size of worker pool to 300 with availability distribution $A(x) = f(x; 0.05, 0.04, 0, \infty)$, that is about 15 available workers at any time for a task. For learning speed and prior knowledge distribution, we set $R(x) = f(x; 80, 10, 0, \infty)$ and $P(x) = f(x; 70, 5, 0, \infty)$ so tasks are more difficult than sleep spindle detection and the workers have a lower starting quality, because we want to test if **DynamicHiring** can find workers to stick to and take advantage of their quality improvement in a thougher environment. We set number of training tasks for each newly hired worker to 20 for both **TopK** and **DynamicHiring**. We assume we have some basic knowledge of the worker population and calculate the base line quality and score of unknown workers from $r' = 100$ and $p' = 100$ which is somewhat close but different from the actual values. We set the system's default availability for worker population to 10% which doubles the actual value so most workers will be less available than system's expectation. We want to see if the system is robust enough to handle this kind of discrepancy and uncertainty. We still use 1000 binary tasks like what we did before. The results for this simulation is shown in Figure 5.2 and Table 5.4. Note that for all the simulations in this section we still run each experiment 10 times and record its average performance. The ks are still set to 3 for **RandomK** and **TopK** while the horizon for **DynamicHiring** is still 5.

Hiring Mechanism	Correct Tasks	Cumulative Quality	Hiring Cost	Training Cost
RandomK	518	51.8%	3000	0
TopK	753	75.3%	3000	5960
DynamicHiring	775	77.5%	2558	4960

Table 5.1: Simulation Result for 300 Workers, 0.05 Average Availability

Unlike what we saw in the previous chapter, now the hiring cost doesn't drop as fast as in the settings where workers are always available. This is because most of workers will only be available for a small percentage of the 1000 tasks. When trained agents are not present, the system has to find some new workers to do the job. Moreover, as now we do not let un-hired workers work on training tasks, workers will not get many opportunities

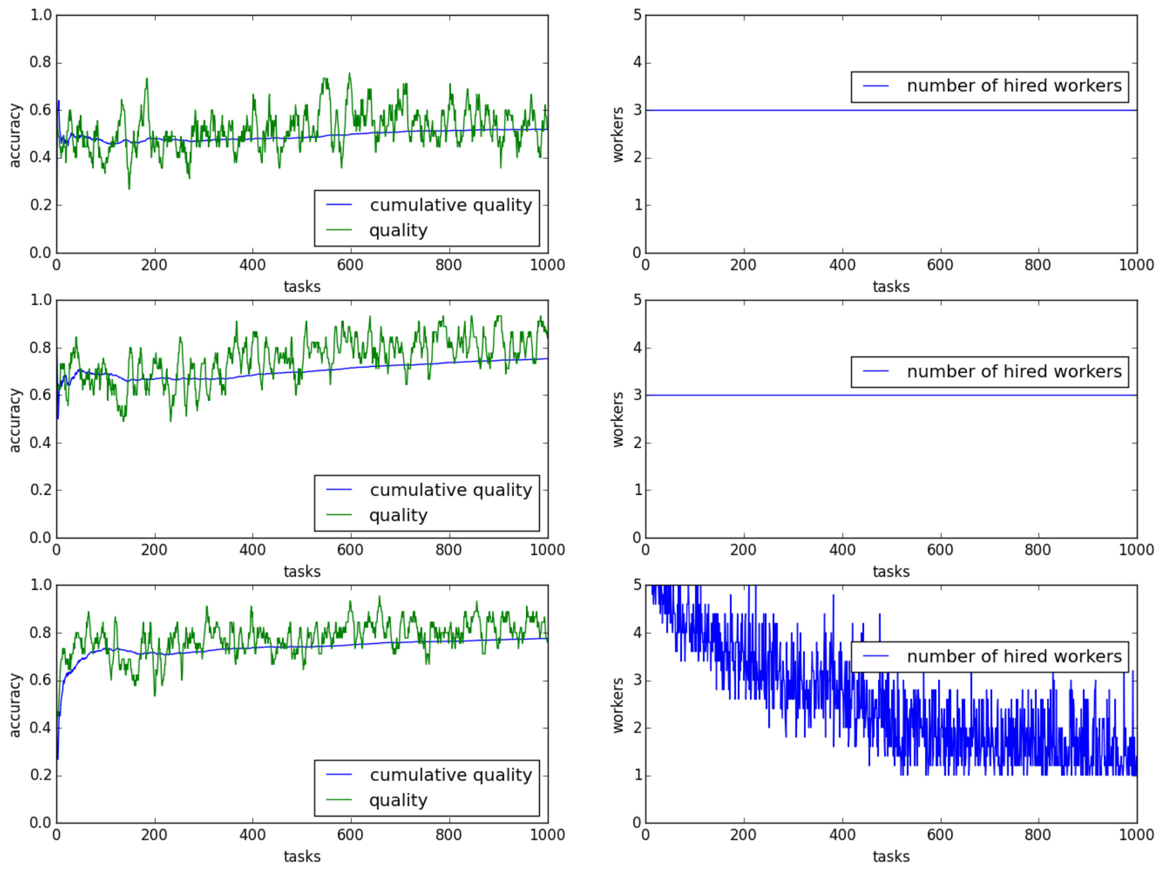

Figure 5.2: Simulation Result for 300 Workers, 0.05 Average Availability

to improve their quality. It becomes harder for the system to locate high quality workers as more tasks are needed to improve the accuracy of quality and learning speed estimation. Overall, the system works as expected, after about 500 tasks, when majority of workers get some tasks to work with, it's able to find a group of higher quality workers and the hiring cost decreases in consequence. Unlike **TopK** which almost run training tasks on all the workers (5960 training tasks means 298 out of 300 workers were asked to do training tasks), **DynamicHiring** has only a training cost of 4960. This behaviour is exactly what we wanted that we only explored part of the worker population, as from our previous discussion the benefit of exploring more workers is marginal after we have already collected information for majority of the worker pool.

Overall, if we consider all tasks of the same cost, from Table 5.4, **DynamicHiring** yield a slightly performance accuracy-wise compared to **TopK** for this worker population, with a total cost of 7518, which is 16.1% less than the total cost of **TopK** (8960). For crowdsourcing applications where cost is a big concern, this cost saving is a huge advantage. And the saving will become even more significant if the amount of tasks is larger, as **DynamicHiring** will hire fewer and fewer workers when number of tasks grows, while **TopK** with a fixed number of hired worker for each task will not benefit from the improved worker quality.

We then want to test the hiring mechanisms against mixed worker population. First we keep the total number of tasks at 1000, but change the availability distribution for 60 (20% of the population) workers to $A(x) = f(x; 0.5, 0.05, 0, \infty)$. Therefore, these 60 workers will be available for almost half of the tasks. We expect **DynamicHiring** to find these highly available workers and save more on hiring cosst. The results in Figure 5.3 and Table 5.4 meets our expectation.

Hiring Mechanism	Correct Tasks	Cumulative Quality	Hiring Cost	training Cost
RandomK	576	57.6%	3000	0
TopK	843	84.3%	3000	5980
DynamicHiring	874	87.4%	1430	1180

Table 5.2: Simulation Result for 240/60 Workers, 0.05/0.5 Average Availability

As we can see, the results for **RandomK** got modestly better. By randomly picking workers for each task, some of the high availability workers will get trained more and contribute more from performance improvement. Both **TopK** and **DynamicHiring** got better, too, while **DynamicHiring** is still outperforming **TopK**. **DynamicHiring** now quickly finds high availability workers and we can see the hiring cost drops significantly after 200 tasks and towards the end our dynamic hiring mechanism is able to stay with 1 to 2 of these highly available workers. As these high availability workers got more tasks to work with, they became well trained which also gave a performance boost. Expenses wise, it now cost $1430 + 1180 = 2610$, which is only 29% of the cost for**TopK**. Note that now we only need to run 1180 training tasks, which means only 59 workers were explored before the system find someone to stay with.

We then want to make the settings even more complicated. We want a worker population in which there exists some low availability but high performance workers. We want our system to find and take advantage of these workers among their peers. To simulate this scenario, we keep 240 low availability medium performance workers, but

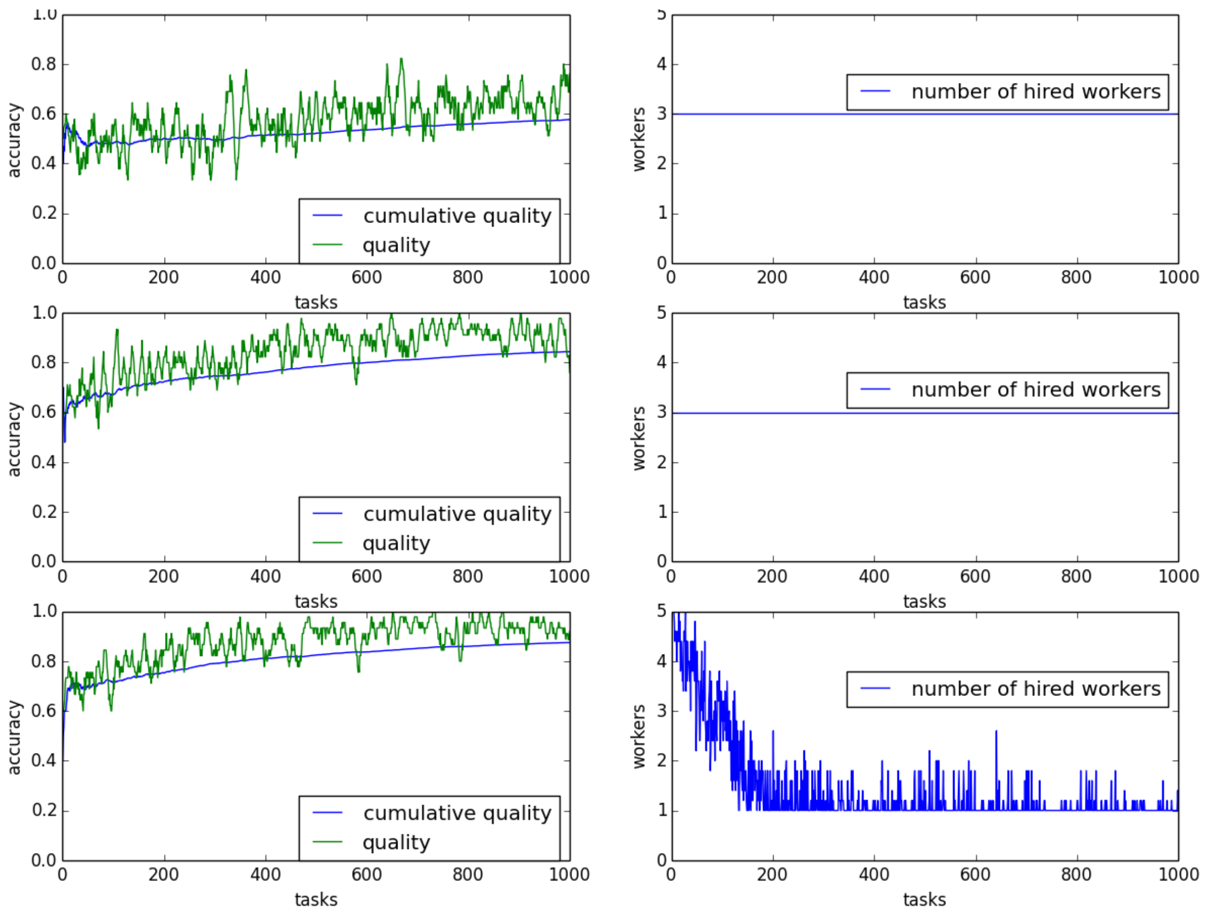

Figure 5.3: Simulation Result for 240/60 Workers, 0.05/0.5 Average Availability

replace the other 60 with workers drawn from distribution $R(x) = f(x; 50, 10, 0, \infty)$ and $P(x) = f(x; 50, 10, 0, \infty)$. These workers have a slightly better starting quality and a much faster learning speed. The availability distribution is still $A(x) = f(x; 0.5, 0.05, 0, \infty)$ so about 3 of these workers are available for any given task. The results are shown in Figure 5.4 and Table 5.4.

The performance for both **RandomK** and **TopK** are better than the first simulation, although the improvement is slight, as the fast learning workers are only a small portion of the worker population and will not help the performance much if they are picked randomly or be discarded as they are not much better at the beginning. In fact the results are worse than what we got in second simulation as slower learners with high availability

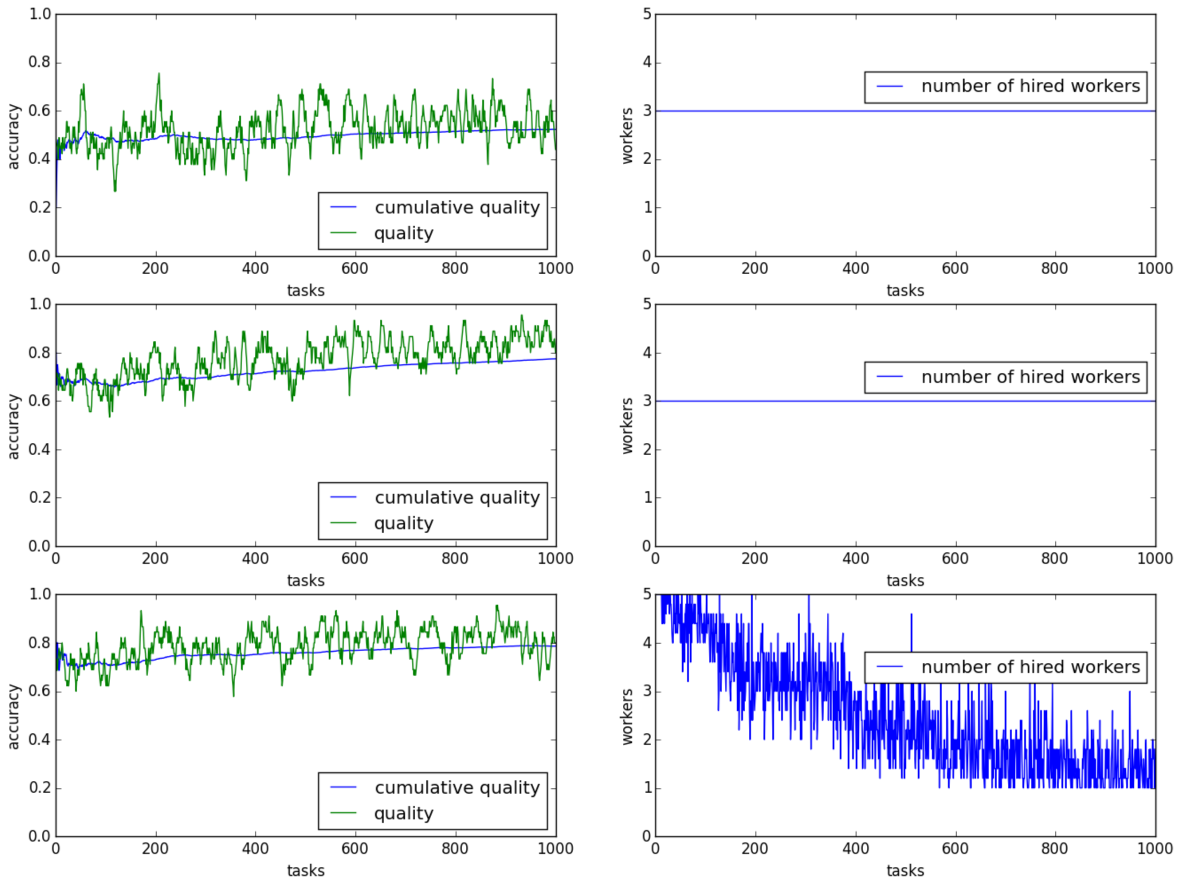

Figure 5.4: Simulation Result for 240/60 Medium/High Quality Workers

Hiring Mechanism	Correct Tasks	Cumulative Quality	Hiring Cost	training Cost
RandomK	524	52.4%	5000	0
TopK	774	77.4%	3000	5980
DynamicHiring	786	78.6%	2517	4560

Table 5.3: Simulation Result for 240/60 Medium/High Quality Workers

had more tasks to train themselves to outperform faster learners with less training. For **DynamicHiring**, we can see the performance got slightly better than the first simulation, too, although the small number of high performance workers will not help much. The cost dropped from 7518 to 7077 (-5.8%), and is 21.2% less than **TopK**. Combined with Figure 5.4 we can see the system was able to find these workers, but it wasn't able to take much advantage of them as their presence was very inconsistent.

Overall, our modified dynamic hiring system is able to handle situations where crowd workers are not always available, and we are not allowed to run free training tasks with all available workers to gather their quality information beforehand. Furthermore, our simulation results showed us it is able to pick up high quality workers or high availability workers from the population, take advantage of their quality improvement, and save significantly on the hiring cost. It provides competitive or better accuracy at solving consensus tasks with drastic lower hiring cost when compared to mechanisms where fixed amount of workers are hired and workers' qualities are assumed as constants.

Chapter 6

Conclusion

In this thesis we studied crowd workers' learning process in doing consensus crowdsourcing tasks. We showed with real data from MTurk that crowd workers are actually improving their quality as they work on more tasks, and each worker may exhibit a different learning behaviour (speed, prior knowledge) when compared to his peers. We adopted the concept of learning curves which were used in industrial plants and factories to model real workers' performance improvement and showed with regression fitting that a modified hyperbolic learning curve can also be used to describe a crowd worker's learning process. We showed that using learning curve to project worker quality is more reliable and accurate than taking the average across all historical data or assuming all workers have a fixed quality which was used in many other researches. We also discussed some of the challenges we are facing with this learning curve model, e.g. relationship between accuracy of estimation and number of data samples, and the way to handle them.

Having a reliable estimation of worker quality gives us the possibility to use Bayesian Network based opinion aggregation approach rather than voting which was used in a lot of crowdsourcing consensus applications. This gives us a good idea of how accurate one prediction is and we are also enabled to take advantage of crowd workers' quality improvement, which directs us intuitively to reduce the number of workers hired when they become better and more accurate at predicting the underlying truth. We then proposed a dynamic hiring system which models the entire hiring process as a MDP and uses Monte Carlo Planning and backward evaluation to make adaptive hiring decisions for each task. Our system makes smart tradeoff between quality of answers and hiring cost, as well as workers' immediate quality and future quality improvement.

We presented two versions of this dynamic hiring system in this thesis. The first one assumes we have full control of the worker population and can let all potential workers to work on a set of tutorial tasks first and make the hiring decisions based on the information we have collected. We compared its performance to other common approaches used in crowdsourcing consensus tasks with simulation tests and it yielded equal or better performance with huge saving in hiring cost, as it was able to locate high quality workers and high potential workers and take advantage of them in the long run. For mixed worker population where workers have diversified learning process (a combination of fast and slow learners), the system was outperforming its peers as it can quickly locate better workers and stay with them. The validity of this mechanism was also verified by real world sleep spindle detection tasks. This system can be used in settings where we can enforce participation, e.g. sleep spindle tests where we only hire nurses from hospitals and clinics, where all workers' availability is known and we can mandatorily let all workers work on training set and only pick the ones we want for real tasks.

To handle more real world applications where workers' availability is unknown and we are not allowed to run free tutorial tasks to workers we are not going to hire, we proposed a modified version which incorporates randomization to make tradeoff between using existing workers and exploring new workers. From our simulation tests, it still shows a better performance accuracy wise, and significant saving in hiring cost when compared to bandit based hiring mechanism that pays workers to work on a subset of tasks first to collect quality information a.k.a. **TopK**. Similarly, for a mixed worker population where part of workers exhibit different learning speed or availability, the dynamic hiring system had a better performance in both accuracy and cost saving, as it could distinguish different types of workers and favoured the ones who are better learners or more available.

Overall, this thesis made contribution to recognize and study crowd workers' learning process in doing consensus tasks. We opened up a new door to devise dynamic and adaptive hiring mechanism that can take advantage of crowd workers' quality difference and improvement. Our system can help tasks assigners reduce their hiring cost in the long run without sacrificing performance, as well as locate high quality and fast learning workers. Of course, our work is still very preliminary and there are many possible directions for future research, which we discuss in the following section.

6.1 Future Work

First, it would be very interesting to test the performance of our proposed dynamic hiring system in a real world application that can run for a longer horizon of tasks. Sleep spindle detection tasks may be a good candidate, but for this thesis we were only able to get a small data set which verified the learning process of workers. If we can build a system on MTurk that can allow returning workers and provide thousands of tasks to work with, we can then verify the performance especially hiring cost saving of our system in the long run.

Second, we are currently only able to work on binary tasks where only two outcomes are possible for our hiring mechanism and simulation. In the future one can extend it to work on tasks with more possible outcomes, e.g. classification tasks where multiple labels are available. To deal with these more complicated tasks, we need to make some assumption about the relationship between all possible outcomes, as people may be more inclined to some outcomes but not others when they are making mistakes, which will complicate the problem dramatically. We may have to add more parameters to the system which we can fine tune according to the actual problem domain. One may also assume all outcomes are independent and analyze the performance of such system on data where this assumption doesn't hold.

Moreover, for crowdsourcing consensus tasks, some of the tasks can be divided into smaller sub-tasks, and the final outcome of a task depends on aggregated opinions on all of its sub-tasks. Our proposed dynamic hiring approach may fit this model nicely, as we may track and measure a workers' quality and learning progress on different types of sub-tasks separately, and make dynamic hiring decisions at a sub-task level.

Also as we have discussed in previous chapters, it would be very interesting to evaluate and compare the influence of external factors on workers' learning process. We experienced quality drop (especially for recall scores) for a few workers in our sleep spindle detection tasks on MTurk towards the end of the 55 windows. We were suspicious that it might be caused by workers getting tired, bored and therefore lowered their effort, but we didn't get the opportunity to setup a separate experiment to test our hypothesis. One may break tasks into shorter sections and compare workers' performance against ones who are working on longer batches. Additionally, we may test the effectiveness of different training materials and methods on workers' learning speed.

Last but not least, in our dynamic hiring system, we were setting a few system parameters based on understanding and knowledge of worker population and task domain, e.g. the weights for current and future quality, belief threshold, bias of task outcomes, and baseline confidence score. It would be interesting to compare our system's performance at some extreme settings where our estimated parameters are completely off or extremely accurate, in both real world experiments and computer simulations.

References

[1] Zooniverse: People-powered research. https://www.zooniverse.org/.

[2] Paul S. Adler and Kim B. Clark. Behind the learning curve: A sketch of the learning process. *Management Science*, 37(3):267–281, 1991.

[3] John R. Anderson. Acquisition of cognitive skill. *Psychological Review*, 89(4):369–406, 1982.

[4] Michel J. Anzanello and Flavio S. Fogliatto. Learning curve modelling of work assignment in mass customized assembly line. *International Journal of Production Research*, 45(13):2919–2938, 2007.

[5] Michel Jose Anzanello and Flavio Sanson Fogliatto. Learning curve models and applications: Literature review and research directions. *International Journal of Industrial Ergonomics*, 41(5):573–583, 2011.

[6] Linda Argote. *Organizational learning: Creating, retaining and transferring knowledge.* Springer, 1999.

[7] Daren C. Brabham. Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence: the International Journal of Research into New Media Technologies*, 14(1):75–90, 2008.

[8] Arthur Carvalho. *Advancements in the elicitation and aggregation of private information.* PhD thesis, University of Waterloo, 2014.

[9] Arthur Carvalho and Kate Larson. A consensual linear opinion pool. *In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 2518–2524, 2013.

[10] Robert T. Clemen. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5(4):559–583, 1989.

[11] Roger M. Cooke. *Experts in uncertainty: Opinion and subjective probability in science.* Oxford University Press, 1991.

[12] Morris H. DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345):118–121, 1974.

[13] Stéphanie Devuyst. The dreams sleep spindle database.

[14] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.

[15] Edward S. Epstein. A scoring system for probability forecasts of ranked categories. *Journal of Applied Meteorology*, 8(6):985–987, 1969.

[16] Enrique Estellés-Arolas and Fernando González-Ladrón de Guevara. Towards an integrated crowdsourcing definition. *Journal of Information Science*, 38(2):189–200, 2012.

[17] Guido Fioretti. The organizational learning curve. *European Journal of Operational Research*, 177(3):1375–1384, 2007.

[18] Fiorenzo Franceschini and Maurizio Galetto. Asymptotic defectiveness of manufacturing plants: an estimate based on process learning curves. *International Journal of Production Research*, 40(3):537–545, 2002.

[19] David Geiger, Stefan Seedorf, Thimo Schulze, Robert C. Nickerson, and Martin Schader. Managing the crowd: Towards a taxonomy of crowdsourcing processes. *In: Proceedings of the 17th Americas Conference on Information Systems*, pages 4–7, 2011.

[20] Christian Genest and Kevin J. McConway. Allocating the weights in the linear opinion pool. *Journal of Forecasting*, 9(1):53–73, 1990.

[21] Panagiotis G. Ipeirotis. Analyzing the Amazon Mechanical Turk marketplace. *Magazine XRDS: Crossroads, the ACM Magazine for Students - Comp-YOU-Ter*, 17(2):16–21, 2010.

[22] Robert A. Jacobs. Methods for combining experts' probability assessments. *Neural Computation*, 7(5):867–888, 1995.

[23] Ece Kamar, Severin Hacker, and Eric Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. *In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 467–474, 2012.

[24] Ece Kamar and Eric Horvitz. Light at the end of the tunnel: A monte carlo approach to computing value of information. *In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, pages 571–578, 2013.

[25] Ece Kamar and Eric Horvitz. Planning for crowdsourcing hierarchical tasks. *In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1191–1199, 2015.

[26] Ece Kamar, Ashish Kapoor, and Eric Horvitz. Lifelong learning for acquiring the wisdom of the crowd. *In: Proceedings of the 23rd Internation Joint Conference on Artificial Intelligence*, pages 2313–2320, 2013.

[27] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

[28] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, 4212:282–293, 2006.

[29] Chris J. Lintott, Kevin Schawinski, Anže Slosar, Kate Land, Steven Bamford, Daniel Thomas, M. Jordan Raddick, Robert C. Nichol, Alex Szalay, Dan Andreescu, Phil Murray, and Jan Vandenberg. Galaxy zoo: morphologies derived from visual inspection of galaxies from the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 389(3):1179–1189, 2008.

[30] C. D. Manning, P. Raghavan, and M. Schütze. *Introduction to information retrieval.* Cambridge University Press, 2008.

[31] James E. Mazur and Reid Hastie. Learning as accumulation: A reexamination of the learning curve. *Psychological Bulletin*, 85(6):1256–1274, 1978.

[32] Allan H. Murphy. A note on the ranked probability score. *Journal of Applied Meteorology*, 10(1):155–156, 1971.

[33] David A. Nembhard and Napassavong Osothsilp. Task complexity effects on between-individual learning/forgetting variability. *International Journal of Industrial Ergonomics*, 29(5):297–306, 2002.

[34] David A. Nembhard and Mustafa V. Uzumeri. An individual-based description of learning within an organization. *IEEE Transactions on Engineering Management*, 47(3):370–378, 2000.

[35] Shanthakumar Pananiswami and Ronald C. Bishop. Behavioral implications of the learning curve for production capacity analysis. *International Journal of Industrial Ergonomics*, 24(1-2):157–163, 1991.

[36] Mahbubur Rahman, Brenna Blackwell, Nilanjan Banerjee, and Dharmendra Saraswat. Weed infestation identification using hierarchical crowdsourcing. *Elsevier Computers and Electronics in Agriculture*, 2013.

[37] Mahbubur Rahman, Brenna Blackwell, Nilanjan Banerjee, and Dharmendra Saraswat. Smartphone-based hierarchical crowdsourcing for weed identification. *Journal of Computers and Electronics in Agriculture*, 113(C):14–23, 2015.

[38] Leonard J. Savage. Elicitation of personal probabilities and expectations. *Journal of American Statistical Association*, 66(336):783–801, 1971.

[39] Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. *In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 614–622, 2008.

[40] James Surowiecki. *The Wisdom of Crowds*. Random House LLC, 2005.

[41] Charles J. Teplitz. *The learning curve deskbook: A reference guide to theory, calculation, and applications*. Praeger, 1991.

[42] Long Tran-Thanh, Sebastian Stein, Alex Rogers, and Nicholas R. Jennings. Efficient crowdsourcing of unknown experts using bounded multi-armed bandits. *Artificial Intelligence*, 214:89–111, 2014.

[43] Jeroen Vits and Ludo Gelders. Performance improvement theory. *International Journal of Production Economics*, 77(3):285–298, 2002.

[44] Carl-Axel S. Staël von Holstein. A family of strictly proper scoring rules which are sensitive to distance. *Journal of Applied Meteorology*, 9(3):360–364, 1970.

[45] Maja Vuković. Crowdsourcing for enterprises. *In: Proceedings of the 2009 Congress on Services*, pages 686–692, 2009.

[46] Simon C. Warby, Sabrina L. Wendt, Peter Welinder, Emil G. S. Munk, Oscar Carrillo, Helge B. D. Sorensen, Poul Jennum, Paul E Peppard, Pietro Perona, and Emmanuel Mignot. Sleep-spindle detection: crowdsourcing and evaluating performance of experts, non-experts and automated methods. *Natural Methods*, 11(4):385–392, 2014.

[47] Jacob Whitehill, Ting fan Wu, Jacob Bergsma, Javier R. Movellan, and Paul L. Ruvolo. Whose vote should count more: optimal integration of labels from labelers of unknown expertise. *Advances in Neural Information Processing Systems*, pages 2035–2043, 2009.