REALIZATION OF SELFISH LEARNING

By
Wenxian Zhang

Senior Thesis in Computer Engineering
University of Illinois at Urbana-Champaign
Advisor: Lav Varshney

December 2018

Abstract

This thesis provides an overview and experiments/simulations to verify improved efficiency and accuracy of social learning through so-called selfish learning that revises traditional social learning patterns (Raman and Pattabiraman, 2018). We begin with introducing new concepts of social learning and the theoretical basis for utilizing selfish learning. Selfish learning is based on the concept that each agent will know decisions made by previous agents while receiving private signals. The agent can choose to declare and pass or declare and stop under the given information. There are two settings for selfish learning: 1) finite horizon, 2) infinite horizon. For finite horizon, we develop crowdsourcing experiments with a finite number of workers to test whether agents will respond according to reward maximization. The task is basically a ball game with different situations of varying combinations of private signal and public signal; workers give their choices in different situations. Additionally, we determine an appropriate experiment sample size by Monte Carlo simulation. For infinite horizons, we use reinforcement learning to simulate a Markov Decision Process formulation. We find that by informing decisions made by previous agents and receiving private signals, the accuracy of answers from crowdsourcing will be improved and the cost will be decreased.

Keywords: social learning, selfish learning, crowdsourcing, Markov decision process, reinforcement learning

Contents

1. Introduction

1.1 Motivation

Crowdsourcing, which involves publishing a set of questions on platforms to let internet users answer, is an important approach in different research areas to collect data sets. To improve accuracy of crowdsourcing, social learning [3] has been introduced.

In social learning, we hire a finite number of workers and let them make decisions in sequential order. Every worker can see the decisions made by previous workers. After all workers make decisions, we get the final worker's decision as the final result. This basic model, in the long run, will appear to conform in that everyone will observe each other's reactions and follow that reaction to be uniform. Therefore, when workers conform to a reaction, this reaction may be wrong [1].

In order to reduce effects of conformism, a new approach was introduced. The difference between original concepts of social learning and what we do—named "selfish learning" [7]—is that each worker will be payed slightly less than the previous worker if the worker declares and chooses to stop, and all workers—that answered before will get the same amount as this worker (which means his answer is the final result and we do not need to find any other workers), but if the worker is incorrect, every worker that answers will not be paid. To sum up, as the number of workers that answer increases, the pay of each worker will be decreased. And each worker can choose to declare or not declare. If the answer is right, all workers get payment at the amount of that worker who stops. If the answer is incorrect, all previous workers including this worker will not get payment. This is what we call "selfish learning".

The basic idea of this thesis is to see whether effects of conformism decrease and whether there is a relation between decrease of payment for increasing number of workers with accuracy and less cost. There is already mathematical basis, so we need to use mathematical basis to see whether it can be applied practically.

There are two parts to realize it: experiments and simulations. For experiments, what we think now is to assume a situation to let each worker answer that if they are in that situation, will they choose to stop and declare an answer, or choose to pass to the next worker to let next worker do decision. For simulation parts, we use reinforcement learning to simulate situations with infinite number of workers.

1.2 Mathematical theory

From concept of "selfish learning", there are mathematical theories dealing with each worker's decision and the reward function. Assume that for each worker, there is probability function H and noise Z with Gaussian distribution, where H is inference hypothesis $\{0,1\}$ and $Z \sim N(0,1)$.

So, the private signal of worker is $Y = H + Z$, and this worker will receive decision from earlier workers which is public signals. In other words, a worker will receive a private signal Y and a public signal which are decisions from the previous worker.

By using a non-increasing variable g, for worker i, the reward function is based on number of workers and workers' answers: $R_i = g_n \mathbf{1}$ {N = total number of workers answered this question, $H_n = H^*$, i < total number of workers}. Based on this function, g_n **,** the reward we give to each worker is the answer of last declared worker is correct**,** is non-increasing in n [7]. So, the longer the worker waits, the lesser the pay becomes. If the answer is incorrect, the payment will be 0.

In selfish learning, the worker who makes the final decision is aiming to maximize the reward due to expected utility theory [6]. Thus, the workers try and maximize the utility given by R_i and have to decide between passing and waiting (in which case g_n decreases but probability of being correct becomes higher) and the choice of stopping and declaring (in which case the g_n may be large, but there is a greater chance of being wrong and not getting anything).

There are three thresholds: $\alpha_n, \beta_n, \gamma_n$. When answering the question with two choices, workers can choose to stop and declare or pass to next worker and declare. Only when previous signals given to the current worker is strong, stopping and declaring will happen. So, for inference hypothesis {0,1} in margins of the scale between 0 to 1, there will be two part that fall in the situation that signal is strong: $Y_n < \alpha_n, Y_n > \beta_n$ where Y_n is the private signal for current worker. (when $Y_n < \alpha_n,$ the current worker will tend to choose to stop and declare $H_n = 0,$ when $Y_n > \beta_n,$ the current worker will tend to choose to stop and declare $H_n = 1.$)

If the signal is not strong, the worker will choose to declare and pass to next worker: $Y_n \in (\alpha_n, \beta_n)$ which means private signal cannot give current worker strong support. But as for passing and declaring, there is also two situations: passing and declaring hypothesis $H = 0$ or passing and declaring hypothesis $H = 1$. Here is the third threshold γ_n where $\alpha_n < \gamma_n < \beta_n$. When the given signal for current worker falls between α_n *and* γ_n, the worker will tend to choose to declare $H = 0$ and pass to next worker. When the given signal for current worker falls between γ_n and β_n, the worker will tend to choose to declare $H = 1$ and pass to next worker. By assuming each worker is selfish, there is a predicted phenomenon that worker will prefer to stop early due to non-increasing payment g_n.

Based on thresholds and private signal, given public signal as q_n:
Decision of nth worker receiving private signal Y_n:

$$H_n = 1\{Y_n > \gamma_n\}$$

Choose whether to stop or pass ($S_n = 0$ if choose to pass, $S_n = 1$ if choose to stop):

$$S_n = 1\{Y_n \notin (\alpha_n, \beta_n)\}$$

Posterior probability of nth worker's hypothesis is 0: $q_n = P[H^* = 0 \mid H_{n-1}, S_{n-1} = 0]$
Where H^* is given private signal and H_{n-1} is hypothesis from previous workers. $S_{n-1} = 0$ means the previous worker chose to pass to the current one.
From Bayes rule, posterior becomes $\frac{q^{n+1}}{1-q^{n+1}} = \frac{q_n}{1-q_n} \frac{P_0[Y \in (\alpha,\gamma)]}{P_1[Y \in (\alpha,\gamma)]}$

Based on theories in social learning system that *n*th worker use posterior $\frac{q^{n+1}}{1-q^{n+1}} = \frac{q_n}{1-q_n}\frac{P_0[Y\in(\alpha,\gamma)]}{P_1[Y\in(\alpha,\gamma)]}$ to estimate, so

$$\gamma_n = \frac{1}{2} + \log(\frac{q_n}{1-q_n})$$

Applying Bayes rule and posterior rule

$$\alpha_n = \gamma_n - \left[log\left[\frac{R_1(q_{n+1}^0)}{1-R_0(q_{n+1}^0)}\right]\right]^+$$

$$\beta_n = \gamma_n + \left[log\left[\frac{R_0(q_{n+1}^1)}{1-R_1(q_{n+1}^1)}\right]\right]^+$$

$R_h(q_n) = C(h) + \rho P_h[Y_n \in (\alpha_n, \gamma_n)]R_h(q_{n+1}^0) + \rho P_h[Y_n \in (\gamma_n, \beta_n)]R_h(q_{n+1}^1)$ where $\rho \in (\frac{1}{2}, 1)$

The formula is formula (11) from [7], and the induction process in included in [7]. As workers will try to maximize reward, the accuracy will be increased, and cost will be decreased partly due to non-increasing g_n.

2.Literature Review

2.1 Crowdsourcing

Crowdsourcing now is a significant part of solving problems that are hard for computers to solve. On a crowdsourcing platform, organizations or individuals publish their needs to the platform and get results from separate internet users. After collecting answers from different agents, organizations or individuals publishing their needs will receive the cumulative answers. A central application is to collect data for machine learning. Data sets are important to develop machine learning theories, testing accuracy, and debug models. Hence crowdsourcing will be applied to a lot of aspects of machine learning. Additionally, crowdsourcing can also be applied to other fields. Researchers can conduct their researches on crowdsourcing platforms [10].

2.2 Social learning

In order to increase accuracy of crowdsourcing, there is a model applied to collect data from crowdsourcing platforms.

To deal with situations that there are two choices and it is necessary to choose one, hiring agents to choose one of them and let agents know information from other agents is a term called “social learning” [3]. Based on concept of social learning, there are some models being studied and used: contagion process model which idea is random matching, homogeneous populations model that agent must choose better one of two choices, heterogeneous populations model which main idea is individually optimal choice, and so on [5].

2.3 Herding

Social learning can increase accuracy of crowdsourcing. It has been applied to different fields and extensively studied. However, because social learning is based on concept that each agent will receive signals from other agents, there will be conformism happening, which, in other word, is called “herding”. Herding is a common phenomenon in our daily life and already studies in some papers such as [8]. When belief becomes too strong and private signal is boundedly informative, herding will occur. People will always follow prevalent choice to be uniform. Reference [8] conducts an experiment on mechanical turk. They assume that there are two baskets, each with 100 balls, and one basket with 70 blue balls and 30 red balls and one basket with 30 blue balls and 70 red balls. They only present one random chosen basket to seven participants and let them draw 9 balls and guess which basket it is. Participants cannot share information except the final answers they give. Thus, participants will know public signals (i.e. decisions from others) and will know their own private signals (i.e. colors of 9 balls). The results of this experiment show that people tend to use all information even when they should not do that. Reference [9] also shows that even when making decision from private signal is optimal, human agents will still tend to be affected by public signals.

From public signals, the agent will prefer to follow most agents’ choice even though the private signal is different from public signals. With presence of herding, several agents choose incorrect answer will lead other agents to choose incorrect answer as well. The final results will also be incorrect because further agents will declare incorrect choice. Therefore, herding will lead undesirable results of social learning [2].

As effects of herding decreases, accuracy of social learning will definitely increase.

2.4 Human subjects to test Bayesian optimal theories of behavior

There are lots of studies about human subjects and Bayesian theories. For example, [8] utilizes human subjects to identify prior probabilities when it comes to decision making tasks of humans and this paper also studied human ability to perceive prior probabilities. In simple perceptual organization, to identify the role of sensory uncertainty, [11] also used human subjects to test whether they perform optimal Bayesian inference. It includes the experiment that they ask 8 participants to judge whether 2-line segments partially occluded belonged to the same line to show the importance of sensory uncertainty of decision making. What’s more, [4] let observers perform the embedded category task and non-parametrically estimates each observer’s category decision boundary to identify difference between Bayesian model and other fixed models. Ball game in [8], as well, applies internet users on mechanical turk to answer questions and collect data to verify their theories of behavior.

3. Experimental design

3.1 Main design of experiment

To verify the selfish learning, we are conducting an experiment by setting games rules and ask answers from workers in Amazon MTurk. We designed and elaborated the experiment by following experimenting methods used in research with team decision making [5, 6]. This experiment is a ball game with special rules. We set three different scoring rules: one with equal payment if final worker gets right answer, one with smoothly decreasing payment if final worker get right answer, and one with significantly decreasing payment if final worker get right answer. Suppose there are two boxes filling with balls of two different color that we can draw balls from them, and each worker can only know previous workers' answers. We ask them to answer that if they know previous workers' answer, how many numbers of balls of each color they see (if they draw nine balls from box) will make them choose which box they are drawing. After publishing this survey on MTurk, we can do experimental analysis to verify whether selfish learning is correct and where it is needed to be modified.

3.2 Mathematical theory of expected answer from workers

From our survey question, we set three workers to answer the ball questions. We need to acquire expected answers from workers to determine whether the collected data is in the range.
There are two boxes A and B. A has 70 red balls and 30 blue balls; B has 70 blue balls and 30 red balls. Drawing from one of the boxes randomly means that each box will have probability of 1/2 to be drawn. Hereinafter, set random variable X to represent box drawing, and represent boxes as A (70,30) and B (30,70).

Therefore, P (X = A) = P (X = B) = 1/2.

From the box which is randomly drawn, we draw total 9 balls from that box. Suppose that there are r red balls and b blue balls, the probability will be:

$$P(r, b \mid X = A) = P_A(r,b) = \frac{\binom{70}{r}\binom{30}{b}}{\binom{100}{9}}$$

$$P(r, b \mid X = B) = P_B(r,b) = \frac{\binom{70}{b}\binom{30}{r}}{\binom{100}{9}}$$

which means the probability that we get r red balls and b blues balls from 9 balls drawn from random one box.

Now we can do likelihood ratio test to find the proper value of b.

$$L(r, b) = \frac{P_A \text{ x } P(r,b \mid X = A)}{(1 - P_{A)} \text{ x } P(r,b \mid X = B)} = \frac{(\frac{1}{2})\frac{\binom{70}{r}\binom{30}{b}}{\binom{100}{9}}}{(1-\frac{1}{2})\frac{\binom{70}{b}\binom{30}{r}}{\binom{100}{9}}} = \frac{\binom{70}{r}\binom{30}{b}}{\binom{70}{b}\binom{30}{r}}$$

If r > b, L(r, b) will be greater than 1 and if b > r, L(r, b) will be smaller than 1. Therefore, b = 4.5 (which is at the middle) is the most proper value.

After we get the expected value of b which is 4.5, we can infer value of a and c. In order to find the expected value which we define as T_1 of the first worker, we should find the probability that the random box is box A (or box B) under the condition that there are r red balls and b blue balls in 9 balls.

Use box A to extend. If the random box is box A:
$T_1 = 10$ P(X = A | r, b) + 0 (1 - P(X = A | r, b)) = 10 P(X = A | r, b)
With expectation value, we can find optimal r and b.

By posterior probability:
P(X = A | r, b) = $\frac{P_A \text{ x } P_A(r,b)}{P(r,b)}$
We already know the value of $P_A(r, b)$, and P_A, so now we need to find P(r, b).
P(r, b) = P(X = A) P(r, b | X = A) + P(X = B) P(r, b | X = B) which we already calculate. Combined with above formula, we can find optimal value of a and b. And we have expected value of a, b and c in the game.

This game is designed to verify the selfish theory. We define the problem that there are two choices, so the answer H_n from workers will belong to $\{0,1\}$. But there is Gaussian noise that the final answer from workers will be $Y_n = H_n + Z \sim (0,1)$, where Y_n is observation and H_n is real answer. And we need to get what exact choice H_n is from Y_n.

Based on that, we have three thresholds in this problem: the first value a which the worker decides to stop and choose the answer as 0, the second value c which the worker decided to stop and choose the answer as 1, the third value b that from a to b the worker chooses 0 and passes, and from b to c the worker chooses 1 and passes. Only when the final answer is correct, workers answer this question will get rewards. So we design the ball game with this problem.

However, in this problem, we do the opposite. We need to know H_n from Y_n. But in the ball game, we tell workers others' choice and ask them to answer Y_n. And after we collect data of Y_n, we can analyze data to see if they are consistent of what we calculate from expectation, the value that we calculate before using likelihood ratios test and posterior inference.

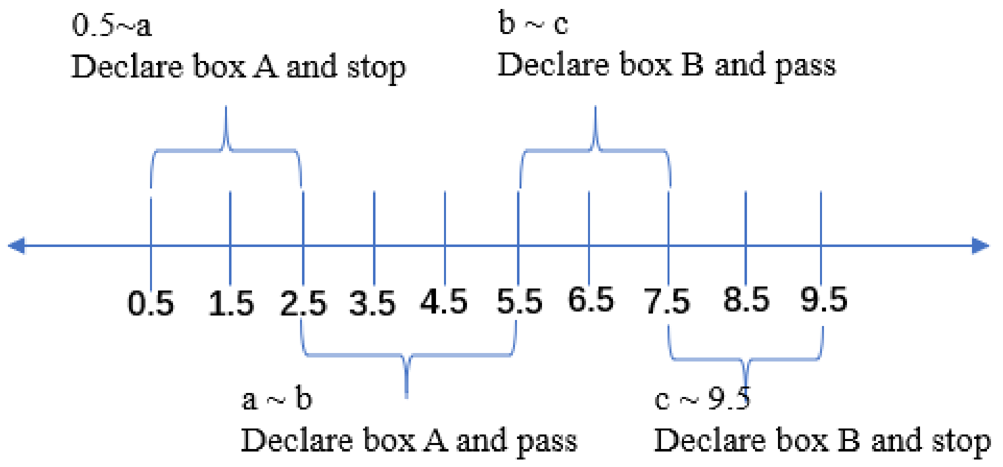

Figure 1: Thresholds and contestant decisions

3.3 Progress on Mturk implementation

On MTurk, we have already done the survey design with HTML and Javascript. Below is the ball game which will be published on MTurk.

Imagine you are in a game show. You are one of 3 contestants in the game. At each round, the contestants are ordered at random. There are two boxes labelled A and B. Box A has 70 blue balls and 30 red balls, and Box B has 30 blue balls and 70 red balls. There are 9 balls drawn from one of two boxes (but you do not know exactly which box is). The sequence of color for these 9 balls is invisible. Now, you need to decide which box these 9 balls belong to. You are the third one to guess the answer.

The host chooses one of the two boxes at random without revealing the identity to the contestants. The contestants are called up in the chosen order and are allowed to draw 9 balls from the box. The contestant can observe the balls and has to return them to the box. Then, the contestant has to declare what he thinks is the identity of the box aloud, and has the option to stop or to pass to the next contestant in the chain. The next contestant repeats the same process. Each contestant is aware of the answers declared by his predecessors.

Below is the rule to win:

1. If the first contestant stops and declares the correct answer, then he gets 10 points.
2. If the first contestant passes and the second contestant declared the correct answer and stops,

then both the first two contestants receive 9 points each.
3. If second contestant chooses to pass and the third contestant declares the correct answer, then each of the three contestants receive 8 points.
4. If the wrong answer is declared, no one receives any point.

NOTE: only contestants who answer in the game receive a reward.

Based on rules, you should answer the listed questions. All answers should be in range of 0-10 balls by following rules:

1. The first number, labeled by ‘a’, should be the maximum number of red balls that you see so that you will declare Box A and choose to stop this question. If you choose a = 0, it means that you will definitely choose box A and stop no matter what sequence of balls you see.
2. The second number, labeled by ‘b’, should be the minimum number of red balls you see so that you will declare Box B and choose to stop this question. If you choose b = 10, it means that you will definitely choose box B and stop no matter what sequence of balls you see.
3. The third number, labeled by ‘c’, is the threshold that you give an answer and decide to pass to next person. Specifically, if the number of red balls is in range of ‘a’ to ‘c’, you will declare Box A and pass to next one. If the number of red balls is in range of ‘c’ to ‘b’, you will declare Box B and pass to next one.

Here is the example of the game:
Nancy flipped the coin to choose a box (Box A or Box B) and let two contestants: Alex, Ben and Claire, play this game. Nancy firstly give this box to Alex and Alex drew 9 balls from that box without knowing which box it is. After seeing color of these 9 balls, Alex guessed that this box should be box A.

Alex is not so confident about his answer, so he decided to pass to Ben and only tell Ben that he thought it should be box A. And then Nancy ask Ben which box it is. Ben does not know what Alex has seen, but only knows that Alex thought it should be box A.
Now suppose you are Ben, depending on the answer from Alex, what is the largest number of red balls you see from that box will make you choose box A (number a) and choose to stop (not pass to Claire)? What is the smallest number of red balls you see from that box will make you choose box B (number c) and not pass to Claire?

Between number a and number c, you are not confident about which box it is, so you need to declare an answer, pass to Claire and tell your guess to Claire. At that time, what is the number of red balls --- number b, that you will choose box A if number of red balls is between a and b, or you will choose box B if number of blue balls is between b and c?
Note: you can only know the answer of previous contestants and you cannot know the color of 9 balls that previous contestants saw.

And then we ask the answer of different cases to collect data.

3.4 Sample size

3.4.1 Simulation to get sample size

We use Monte Carlo simulation to mimic worker behaviors and find proper size of the crowd. Monte Carlo simulation is a way to randomly simulate different cases and tackle those random samples to get final results we want. It can give us prediction of uncertainty and risk. It works by randomly choosing values for each task based on given parameters (the number we calculate and predict). After setting a chosen sample size n, it randomly generates n tasks and we can calculate average value of these n tasks. This is one-time simulation that we find an average value from n-time random generating. Monte Carlo simulation do hundreds of thousands of this step to simulate a huge number of results based on random values. Calculating the percentage of results out of range in total final results will tell us the error percentage and we can modify the parameters we set to see when the error percentage satisfies our requirement.

Here, we use Gaussian distribution to get random cases. By using python, the program generates a group of samples using three variables: the predicted sample size we set, the allowed error range and the correct answer we calculate. Firstly, it generates same samples with size of sample size we set, and then it will generate random Gaussian number and add them to those samples. So it will have n (sample size we set) different samples which have already been added with Gaussian random number. Secondly, calculating sum of those samples and getting average is a step to get one result of simulation. Repeating this simulation by one thousand times, we will have one thousand results of average of each simulation. So now, we have a Gaussian distribution of those average results, and then we can adjust the sample size we set to see which number will make the results fall into allowed error range. If it is, we can get final result of sample size.

Below is the pseudo code of simulation:

```
        Assume there is a pre-set sample size and correct answers of $\alpha_n, \beta_n, \gamma_n$.

        Create a Final result array

        Generate sample array with pre-set sample size, each element is list of correct
        answer $(\alpha_n, \beta_n, \gamma_n)$

    For i in range (1000):
            For i in range(sample size):

                // get different samples

                    Modified simulated signals based on correct answers $\alpha_n, \beta_n, \gamma_n$ by
                    adding noise to randomly created samples. There will be new
                    sample array with noise.
            // get the average values of those signals
            Average value = sum of sample array/sample size
            // add them to final result
            Add average value of final result array
        Check whether the final Gaussian distribution falls in allowed error range
```

The above pseudo code is one-time simulation of stable sample size. After this simulation, we will have an average value from randomization. This average value is what we want. And then we append this average value to final result array. After 1000-time simulation, there will be an array with 1000 average values.

With this final result array:

```
    Float epsilon;
    // Set epsilon value which is the range of error we allow. We only consider the average
    value which falls in range.
    int count;
    // count the frequency of signals which are out of error range we set
    For i in range(size of final result array):
        if final result array[i] out of range [correct answer–epsilon, correct answer
+epsilon]
            count++;
    return count/size of final result array
```

Hence, we can get probability of error with sample size we set. We can input any value of sample size and correct answer to see how probability of error varies. By outputting all probability of error, we can compare the statistics and find the most proper one. After adjusting pre-set sample size and error range, we can get the most suitable sample size to do the experiments.

3.4.2 Results of simulations

To simulate, we choose a = 5.5, b = 7.5 and c = 8.5 and epsilon = 0.1 to see the results. This MCMC is for one extreme case of answers for alpha, beta, gamma. The reason I chose this particular one was because the workers are fairly certain of the hypothesis before making their observation. That is why the thresholds are biased toward high values. In such an instance, the workers should necessarily be able to function well. Contextually it is one of the easier cases and hence we can utilize this check (sanity check).

At the same time, because the thresholds are close to the extreme, the chance of making errors is also more pronounced because of the quantization. For example, as graph attached below, line of variable c is higher than other two lines. This is because the answers can at most be 9.5, and the correct answer is 8.5. So most large errors are quantized sharply to lie within the range whereas that's not the case necessarily with the negative errors. That means we implicitly induce a bias in the worker responses and so the probability with which they are wrong is also more.

After choosing these four values of variables, we run MCMC simulation. In order to see results directly, we set x-axis as sample size we tried and y-axis as percentage or error for each variable after simulation by matching sample size. From example, (10,0.77) for blue line means that when we choose 10 as sample size, after we simulate MCMC by hundreds of thousands of times and calculate average percentage error, we got 77% error percentage for value of variable a.

Here is the graph we initially got:

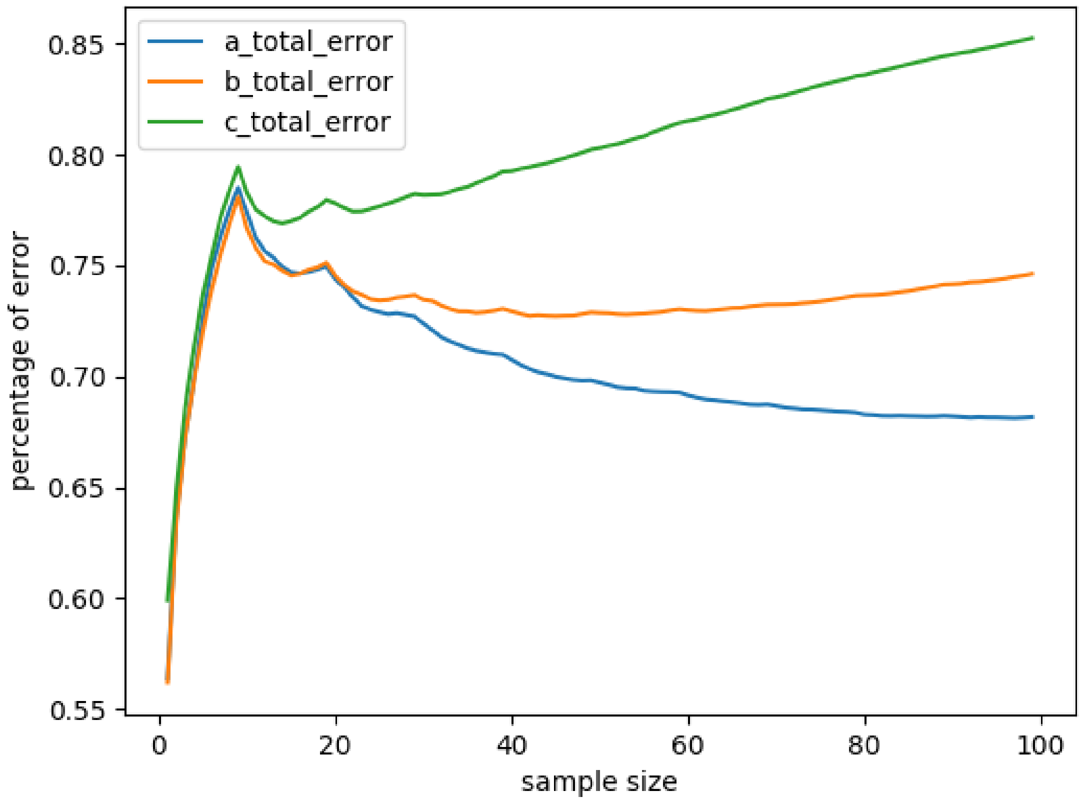

Figure 2. sample size with matching percentage of error (incorrect version)

This graph is not the graph we expected. At the beginning, three lines increase fast and begin to decrease about sample size = 10. What's more, percentage error of c decreases most smoothly but begins to increase firstly. Percentage value of b become stable and c keep decreasing. This is not we expect to see. Percentage of error should be decreased as number of sample size increases.

After analyzing, it is an effect of the quantization and significantly low epsilon value in comparison to the noise level. Since the levels are separated by 1, it suffices to use an epsilon of 0.5 on the average responses of the crowd.

Additionally, it also seems like there is some mistake with the average values computed for a,b,c as the values seem to be outside the range.

For a bit more rigor in the estimates from MCMC, in place of computing the mean of the (noisy and quantized) responses, we use the median of the responses instead of means of the responses. In general, the median is a little more robust in the presence of such non-linear operations (like quantization).

Therefore, after seeing results under condition {a = 5.5, b = 7.5, c = 8.5, epsilon = 0.1}, we modify the code by adjusting epsilon to 0.5 and change calculating average values of each variable as results for each time of simulation to calculating medians of each variable.

Here is resulted graph after modifying original code:

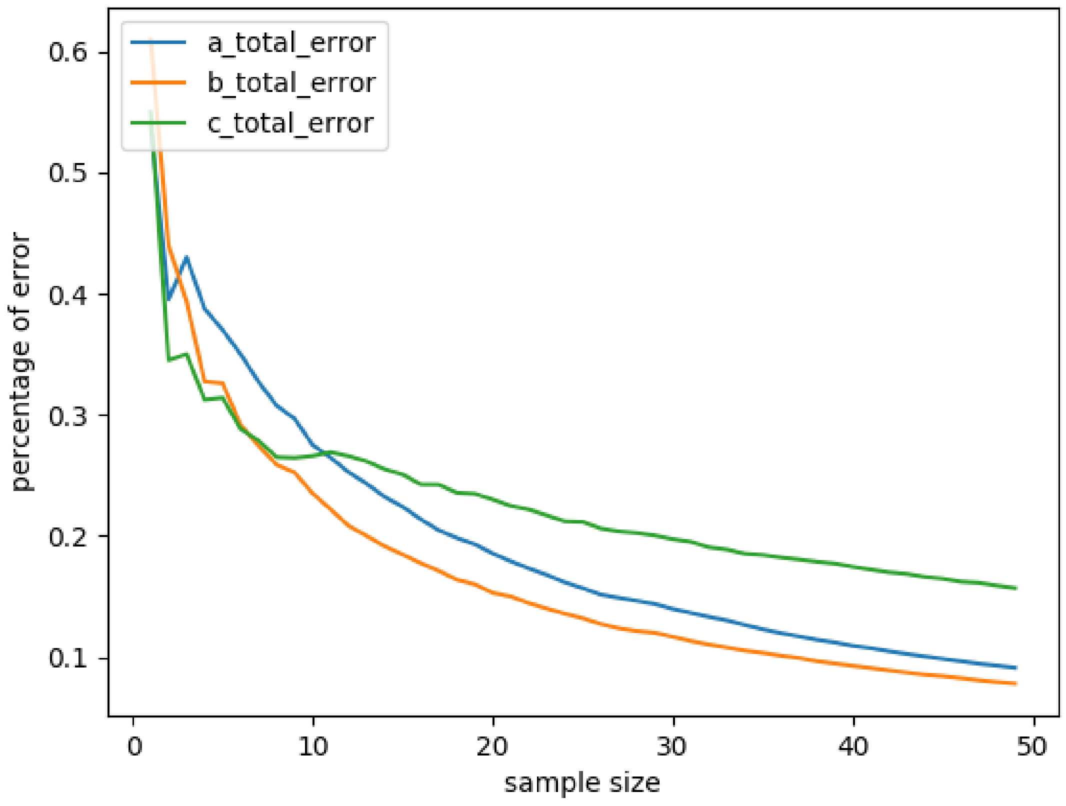

Figure 3. sample size with matching percentage of error (correct version)

To check, we discover that in histogram, values for each variable are accurate:

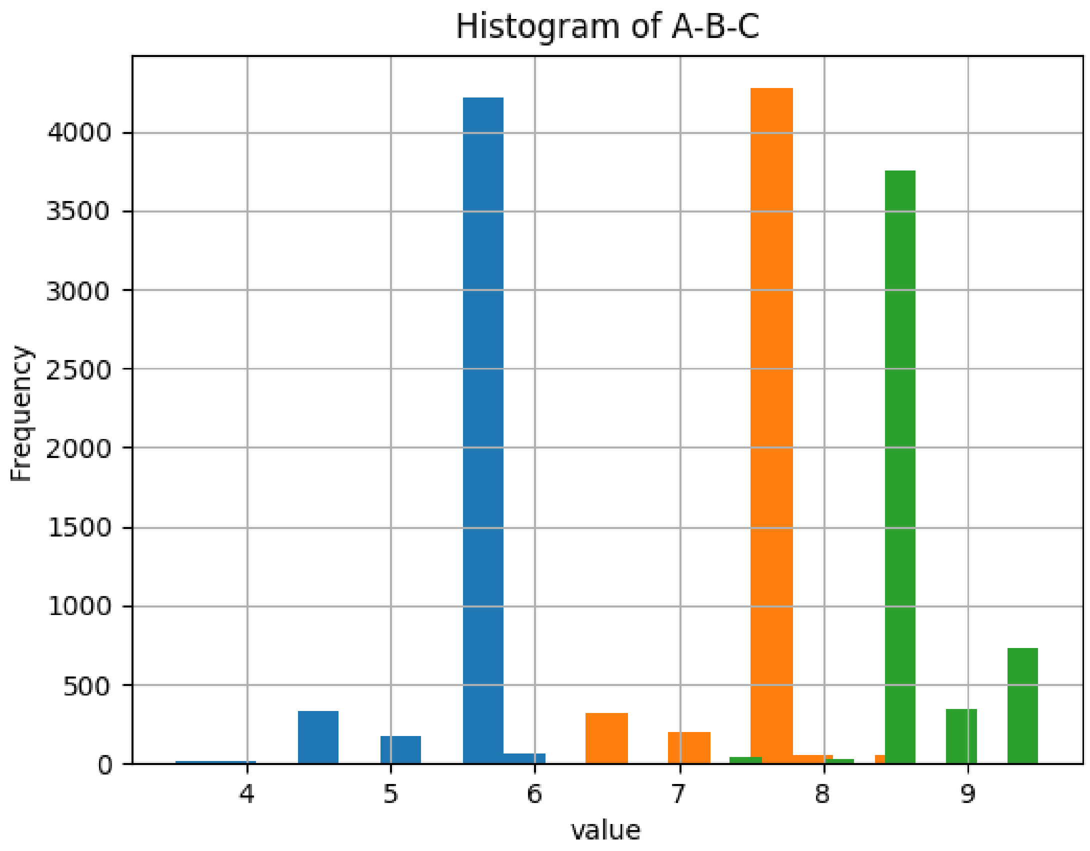

Figure 4. histogram of frequency for each a, b and c value

(x-axis is simulated value of the worker and y-axis is the frequency of that value)

The final result works. In practice, we would use this error graph as a benchmark, and set an error level, say 10%, and identify the number of samples to be obtained for this. This can also be used the other way around in that, given the statistics of the responses and the error probabilities of the workers, we can do some post-processing to the data to correct for such errors.

4. **Simulation of infinite horizon**

4.1 Mathematical basis

Agents respond in order to maximize rewards, and the reward is dependent on below functions:

public signal: $q_n = \mathrm{P}\,[\, H^* = 0 \mid \hat{H}^{n-1} = h^{n-1}, \hat{S}^{n+1} = 0 \,]$ which is action set we choose

$$\text{Threshold: } \gamma_n = \frac{1}{2} + \log\left(\frac{q_n}{1-q_n}\right)$$

$$\text{Variables: } \begin{cases} \alpha_n = \gamma_n - \left[log\left[\frac{R_1(q_{n+1}^0)}{1-R_0(q_{n+1}^0)}\right]\right]^+ \\ \beta_n = \gamma_n + \left[log\left[\frac{R_0(q_{n+1}^1)}{1-R_1(q_{n+1}^1)}\right]\right]^+ \end{cases}$$

$$\text{coefficient: } \rho \in \left(\frac{1}{2}, 1\right)$$

$$P_0 = \begin{cases} P_0[Y \in (\alpha,\gamma)], & if\ \frac{q'}{1-q'} = \frac{q}{1-q}\frac{P_0[Y \in (\alpha,\gamma)]}{P_1[Y \in (\alpha,\gamma)]} \\ P_0[Y \in (\gamma,\beta)], & if\ \frac{q'}{1-q'} = \frac{q}{1-q}\frac{P_0[Y \in (\gamma,\beta)]}{P_1[Y \in (\gamma,\beta)]} \\ P_0[Y \notin (\alpha,\beta)], & if\ q' = \emptyset \end{cases}$$

$$\text{Cost function: } C(h) = \begin{cases} P_0[Y \le \alpha_n], & if\ h = 0 \\ P_1[Y \ge \beta_n], & if\ h = 1 \end{cases}$$

By the above variables and coefficients:

$$R_h(q_n) = C(h) + \rho P_h[Y_n \in (\alpha_n, \gamma_n)] R_h(q_{n+1}^0) + \rho P_h[Y_n \in (\gamma_n, \beta_n)] R_h(q_{n+1}^1)$$

Here, we need to find q_n and q_{n+1} to maximize reward to see whether it acts as expected when in infinite horizon.

Reinforcement learning implies that workers can learn optimal function by playing the game a few times and observing rewards they get. By this simulation, we can ideally get the numerical characterization of optimal worker behavior (i.e. how should workers behave).

4.2 Realization of mathematical part

4.2.1 reinforcement learning with fixed number of states

We utilize Lagrange multipliers and constrained optimization to optimize rewards with those constraints and get matching actions responding to maximized rewards.

In Lagrange multipliers and constrained optimization, there are two functions *f* and *g*, and there must be a point that *g* touches with *f*. This point is the optimal point we want to find.

$$L(x, \lambda) = \Delta f(x) + \lambda \Delta g(x) = 0$$

Where *f* is the reward function, and *g* is the constraint function.

Because q_n^h = P (h = 0) ∈ [0,1], we divide it to 9 parts:

$$q_n^h \in (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9),$$

For each q_n^0 and q_n^1, we use Lagrange Multiplier to find the best q_{n+1}^0, q_{n+1}^1 and the maximized matching rewards.

Here is pseudo code of reinforcement learning code:

```
$$C(h) = \begin{cases} P_0[Y \le \alpha_n], & \text{if } h = 0 \\ P_1[Y \ge \beta_n], & \text{if } h = 1 \end{cases}$$
    f($\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1$) = -($C(h) + \rho P_h[Y_n \in (\alpha_n, \gamma_n)] R_h(q_{n+1}^0) + \rho P_h[Y_n \in (\gamma_n, \beta_n)] R_h(q_{n+1}^1)$)
    constraints :=
        $[\alpha_n - \gamma_n + \left[log\left[\frac{R_1(q_{n+1}^0)}{1-R_0(q_{n+1}^0)}\right]\right]^+ = 0,$
        $\beta_n - \gamma_n - \left[log\left[\frac{R_0(q_{n+1}^1)}{1-R_1(q_{n+1}^1)}\right]\right]^+ = 0,$
        $\frac{q'}{1-q'} - \frac{q}{1-q}\frac{P_0[Y \in (\alpha,\gamma)]}{P_1[Y \in (\alpha,\gamma)]} = 0,$
        $\frac{q'}{1-q'} - \frac{q}{1-q}\frac{P_0[Y \in (\gamma,\beta)]}{P_1[Y \in (\gamma,\beta)]} = 0]$
    So, we have constraint function:
        $g_1(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1) = [\alpha_n - \gamma_n + \left[log\left[\frac{R_1(q_{n+1}^0)}{1-R_0(q_{n+1}^0)}\right]\right]^+,$
        $g_2(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1) = \beta_n - \gamma_n - \left[log\left[\frac{R_0(q_{n+1}^1)}{1-R_1(q_{n+1}^1)}\right]\right]^+,$
        $g_3(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1) = \frac{q'}{1-q'} - \frac{q}{1-q}\frac{P_0[Y \in (\alpha,\gamma)]}{P_1[Y \in (\alpha,\gamma)]},$
        $g_4(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1) = \frac{q'}{1-q'} - \frac{q}{1-q}\frac{P_0[Y \in (\gamma,\beta)]}{P_1[Y \in (\gamma,\beta)]}$
    L($\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1, \lambda_1, \lambda_2, \lambda_3, \lambda_4$) = $\Delta$f($\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1$) + $\lambda_1 \Delta g_1(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1)$
            + $\lambda_2 \Delta g_2(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1)$ + $\lambda_3 \Delta g_3(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1)$ + $\lambda_4 \Delta g_4(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1)$
    For each $q_n^0$:
        Solve L($\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1, \lambda_1, \lambda_2, \lambda_3, \lambda_4$) to find optimal $(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1)$
    For each $q_n^1$:
        Solve L($\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1, \lambda_1, \lambda_2, \lambda_3, \lambda_4$) to find optimal $(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1)$
    After we get the matching optimal next-action set and optimal rewards, we update reward matrix and repeat optimization again. After the
    $(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1)$ and f$(\alpha_n, \beta_{n,}, q_{n+1}^0, q_{n+1}^1)$ are converged, we get the final optima results.
```

And we can finally get best q_{n+1}^0, q_{n+1}^1 , $\alpha_n and\ \beta_n$, and the maximized matching rewards of each pair of q_n^0 and q_n^1. It will give us clearer understanding of what is the correct selfish behavior, and obtain further insights on how to work with workers who are selfish

4.2.2 results of reinforcement learning (not as expected)

After we run the reinforcement learning code for huge number of iterations, results look weird (all lines of rewards and $\alpha_n, \beta_{n,}, q^0_{n+1}, q^1_{n+1}$are not stable):

Reward graphs after running the code, which is obviously unstable:
(blue line is reward when h = 0 and orange line is reward when h = 1)

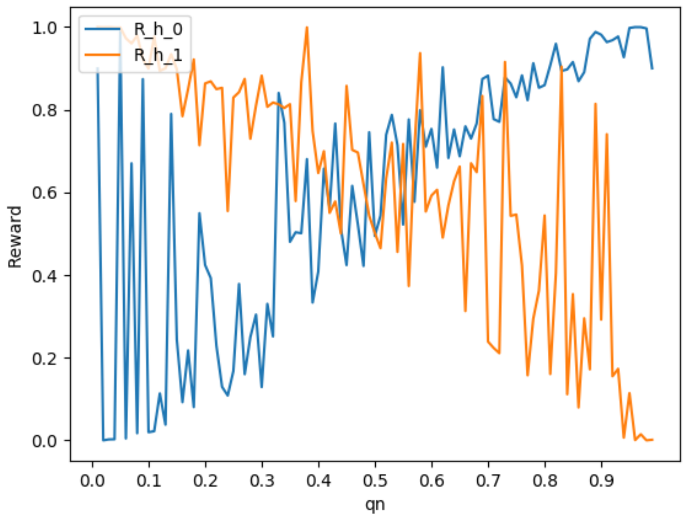

Figure 5. reward of h = 0 and h = 1 versus value of q_n

$\alpha_n, \beta_{n,}\gamma_n$ *when* $h = 0$, which is obviously unstable:

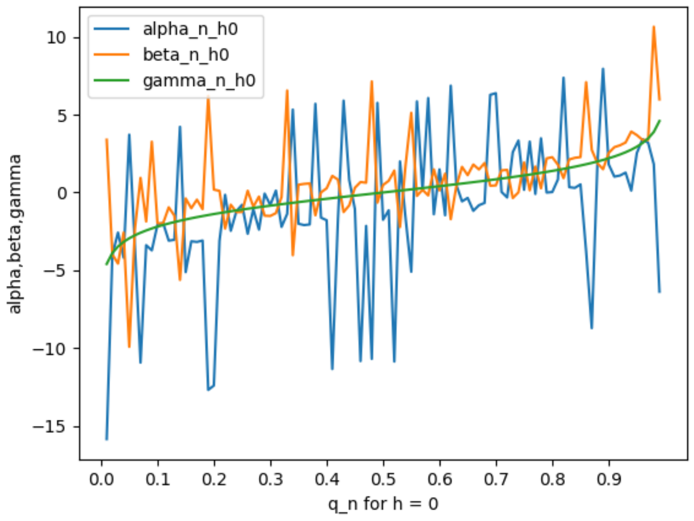

Figure 6. $\alpha_n, \beta_{n,}\gamma_n$ vs. value of q_n when h = 0

$\alpha_n, \beta_{n,}\gamma_n$ $when\ h = 1$:

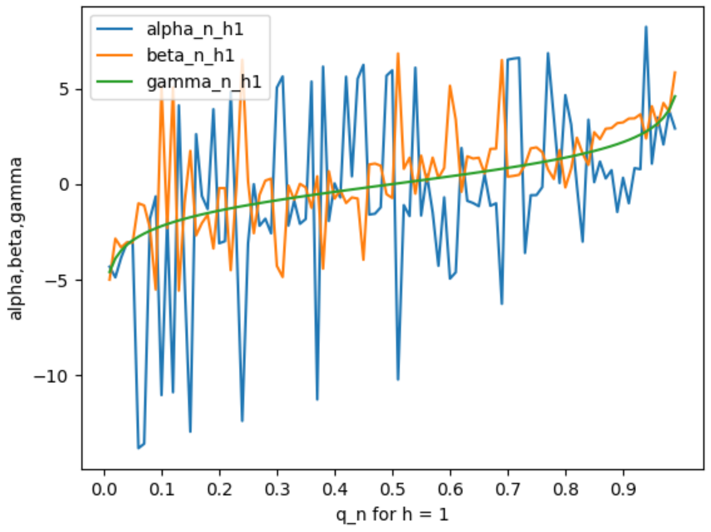

Figure 7. $\alpha_n, \beta_{n,}\gamma_n$ vs. value of q_n when h = 1

Matching next state, which fluctuates dramatically and shows nothing:
(blue line is next state for h = 0 and orange line is next state for h = 1)

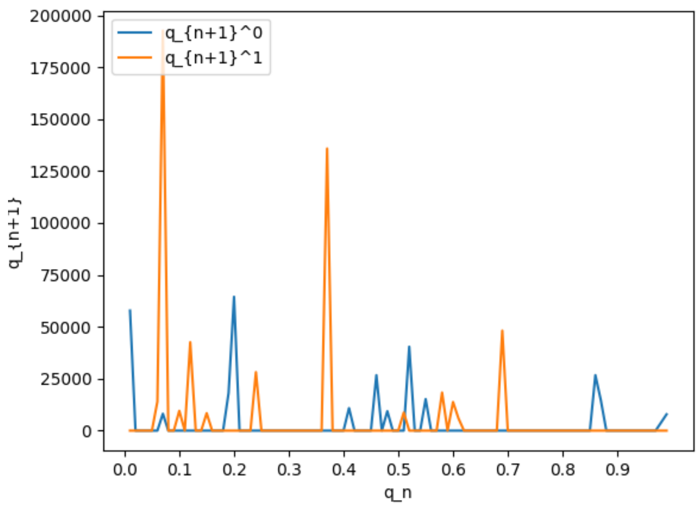

Figure 8. q_{n+1} vs. q_n

4.2.3 Q-learning

In reinforcement learning, we use fixed state $q_n^h \in (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)$ with h = 0 and h = 1 to find best next state with max reward. This learning exploit fixed number of combinations and find the best one. After analyzing the graph and the code, we think that the unstable results are from some tricky numerical calculation in optimization.

To avoid unstable results from previous section, we change reinforcement learning to Q-learning by exploring Q table to avoid optimizing steps with fixed number of states.

We set ranges of available values of $\alpha_n, \beta_{n,}\gamma_n$ and make them as action sets matching with each state.

After firstly initializing $\alpha_n, \beta_{n,}\gamma_n$ with random values and setting q = $\frac{1}{2}$, we use function $\frac{q'}{1-q'} = \frac{q}{1-q}\frac{P_0[Y\in(\alpha,\gamma)]}{P_1[Y\in(\alpha,\gamma)]}$ and $\frac{q'}{1-q'} - \frac{q}{1-q}\frac{P_0[Y\in(\gamma,\beta)]}{P_1[Y\in(\gamma,\beta)]}$ to calculate q'(next state) and use calculated q' to get matching reward value with function

$$R_h(q_n) = C(h) + \rho P_h[Y_n \in (\alpha_n, \gamma_n)]R_h(q_{n+1}^0) + \rho P_h[Y_n \in (\gamma_n, \beta_n)]R_h(q_{n+1}^1)$$

The reward value is what we need to fill in Q table and we get $R_h(q_{n+1}^0)$ and $R_h(q_{n+1}^1)$ from Q table when we calculate current reward value. Q table is initially filled with random values and then updated with process of Q-learning.

After running a certain amount of iterations, we get a fully updated Q-table and can know best reward and action for each state.

4.2.4 Results of Q-learning (expected)

Here is reward values when ρ = 0.55 (h = 0), ρ is non-increasing payment for worker. As we can see, the reward is increasing with value of q_n

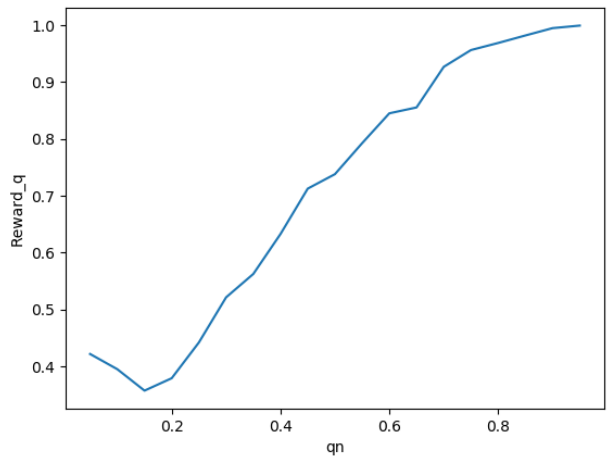

Figure 9. reward vs. q_n when h = 0, ρ = 0.55

Below is reward values for ρ = 0.80 (h = 0). Compared with figure 9, the line looks smoother.

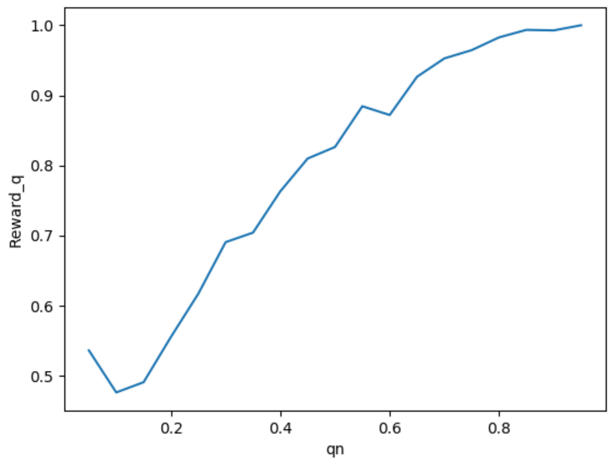

Figure 10. reward vs. q_n when h = 0, ρ = 0.80

Reward values for ρ = 0.95 (h = 0):

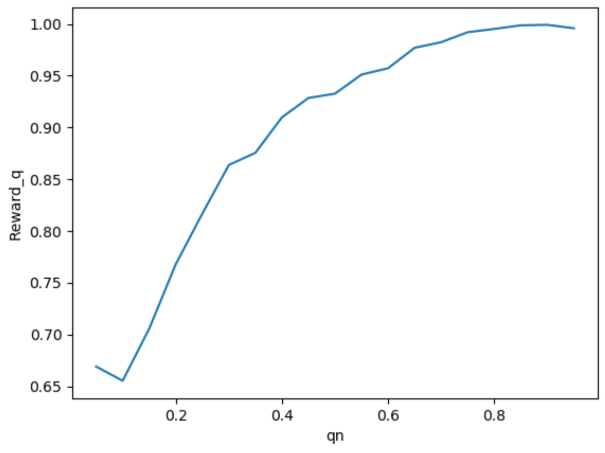

Figure 11. reward vs. q_n when h = 0, ρ = 0.95

Best actions (three threshold) for ρ = 0.80 （h = 0）:

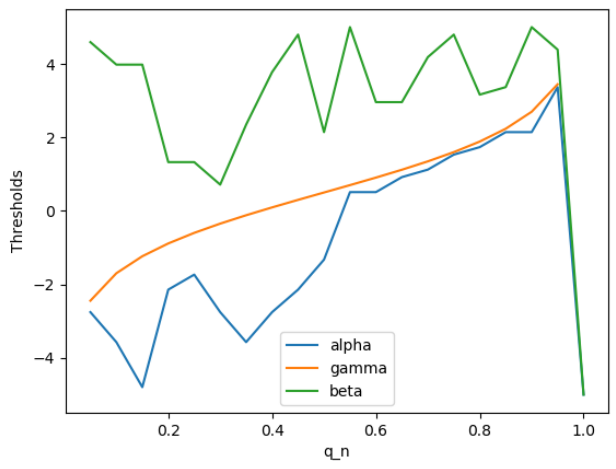

Figure 12. $\alpha_n, \beta_{n,}\gamma_n$ vs. value of q_n when h = 0

Reward values for ρ = 0.99 (h = 1):

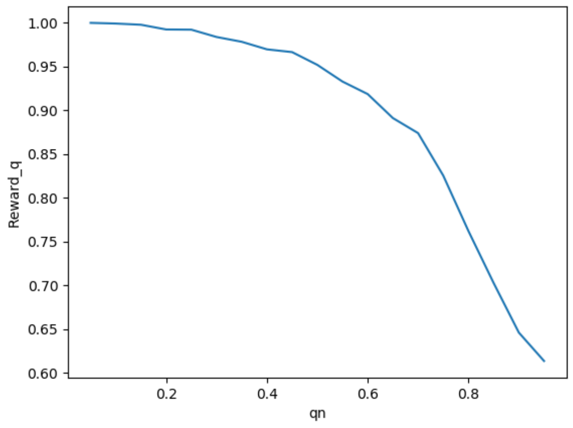

Figure 13. reward vs. q_n when h = 1, ρ = 0.99

Best actions (three threshold) for ρ = 0.99 (h = 1):

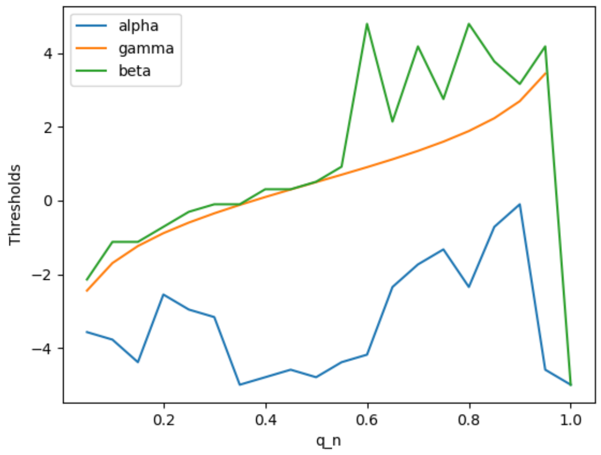

Figure 14. $\alpha_n, \beta_{n,}\gamma_n$ vs. value of q_n when h = 0

Final reward graph for h = 0:

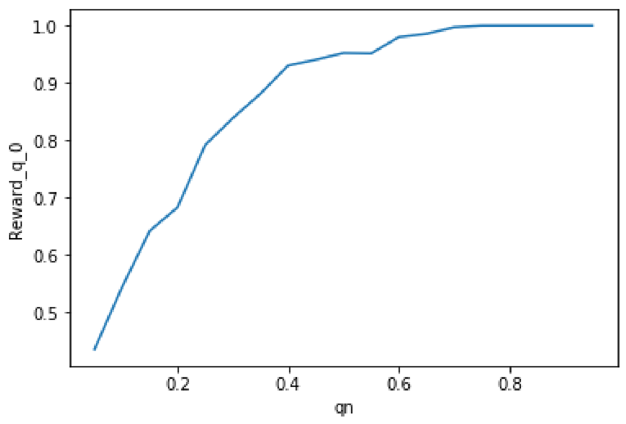

Figure 15. Final reward vs. q_n when h = 0

Final reward graph for h = 1:

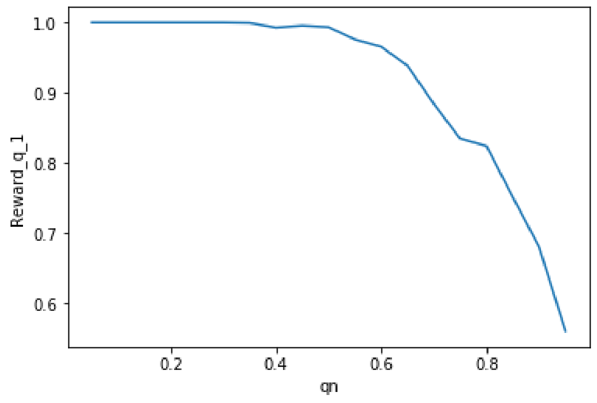

Figure 16. Final reward vs. q_n when h = 1

After getting two Q-tables of h = 0 and h = 1, we combine two Q-tables to one with function:

$$\text{Q-table} = q_n^0 * Q - table_{h=0} + (1\text{-}q_n^1) * Q - table_{h=1}$$

The new Q-table stored matching average reward values of each state corresponding to each action (thresholds) $\alpha_n, \beta_{n,}\gamma_n$. By finding optimal reward for each state and running a certain amount of iterations, we can get optimal action graph:

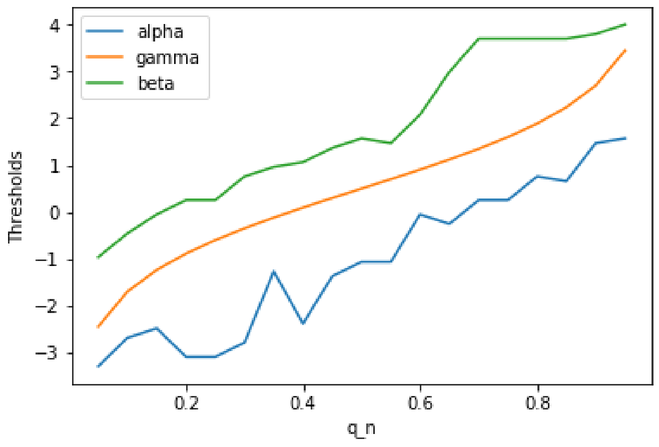

Figure 17. Final $\alpha_n, \beta_{n,}\gamma_n$ vs. value of q_n when h = 0

Now we are confirmed that the simulation is correct. So, we ran more iterations on workspace to get elaborated graph.

Reward graph when h = 0 with three different payoff values:

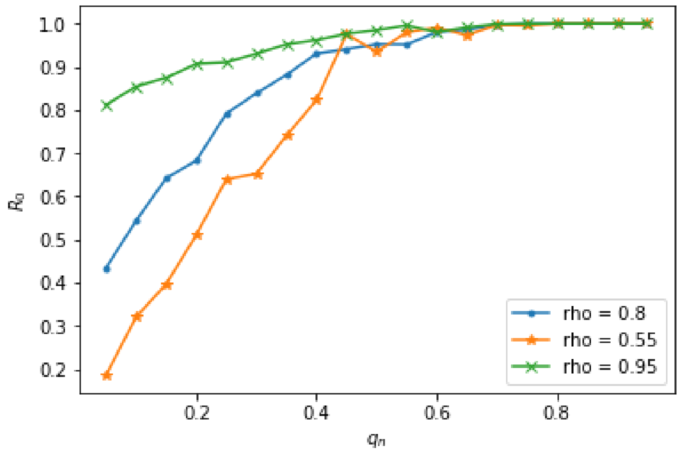

Figure 18. Final reward vs. q_n when h = 0 with different value of ρ

Reward graph when h = 1 with three different payoff values:

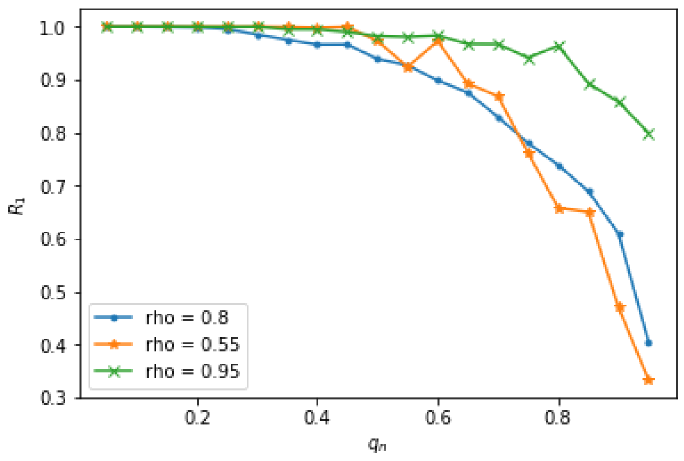

Figure 19. Final reward vs. q_n when h = 0 with different value of ρ

Average payoff comparison (average of two rewards):

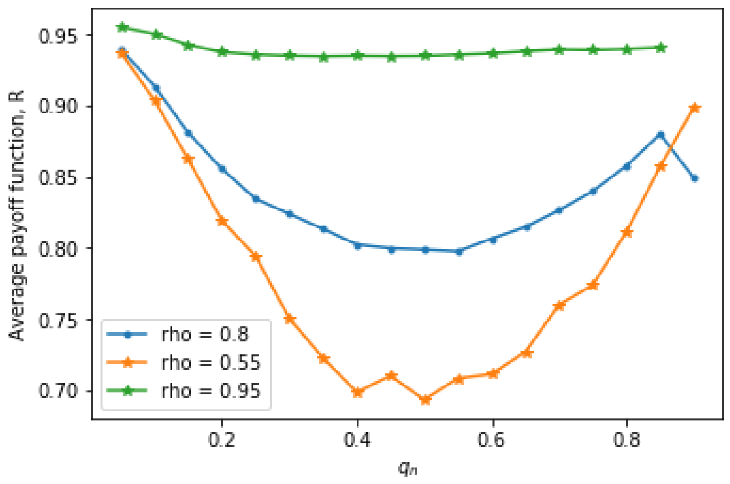

Figure 20. Average final reward vs. q_n for h = 0 and h = 1 with different value of ρ

Threshold comparison:

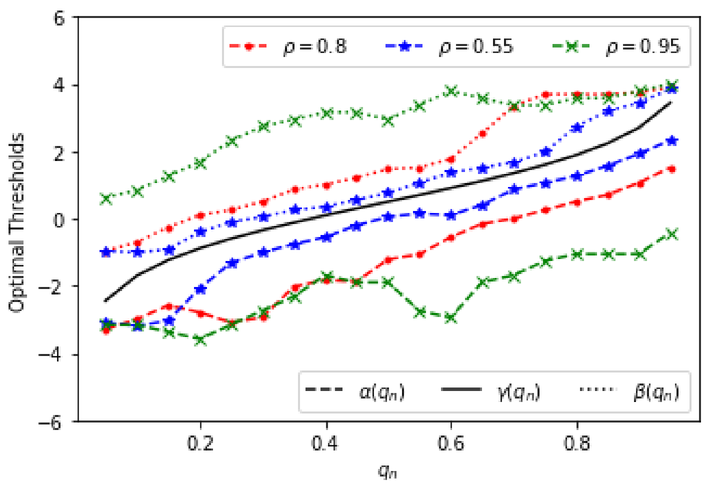

Figure 21. Final $\alpha_n, \beta_{n,}\gamma_n$ vs. value of q_n when h = 0 with different value of ρ

5. Conclusion

The graphs we get from Monte Carlo simulation and Q-learning can give us three thresholds $\alpha_n, \beta_{n,}\gamma_n$in different situations (different q_n) and sample size in Mturk experiment by setting three thresholds and q_n.

When we do Q-learning, we have different states q_n and fix ρ which represents non-increasing payment rate for each worker. If one worker chooses to declare and pass to next worker, payment will be decreased. It means that with number of answered workers increasing, payment for each worker will be reduced and will be zero if answer if incorrect. Under this condition, it is intuitively that if passing to next worker will lead to significant decrease of payment, such as 0.9 to 0.1, current worker will prefer to stop and declare because it is not worth of passing to next one. Oppositely, if payment just decreases slightly, it is safe for current worker to declare answer and pass question to next one. Therefore, as payment for each worker increases, the reward curve will be smoother at the end because each worker intents to get maximized reward. Current worker will prefer to pass to next one due to less risk and slightly different payment. The last half part of q_n represents bigger probability that h = 0. As workers prefer to pass to next one, the possibility to get big rewards will be increased, as figure 18 shows.

Therefore, when h = 1, as possibility of h = 0 decreases, reward will be increased. So, reward graph of h = 1has opposite trend compared with reward graph of h = 0, as shown in figure 19.

For three thresholds, as can be seen in the comparative thresholds plots (figure 21), if the payoff for the agents decays fast, then the thresholds for alpha and beta are close to gamma and go further away when rho increases. This is because, if the agents loose payoff fast upon passing, they prefer to stop more often instead. On the other hand, if the payoff does not decrease by much, then they prefer to pass and wait for the correct answer. Also, the alpha values converge to gamma when q_n is close to 1 and beta converges to gamma when q_n is close to 0. This is because, if the agent is confident that the truth is hypothesis 0 ($q_n$close to 1), then they are more willing to stop and declare 0, and so alpha is close to gamma. On the other hand, if they are confident of it being a 1 (q_n close to 0), then they stop and declare 1, and so beta is close to gamma in this case.

Next, the average payoff received by the agent is higher when $q_n$is close to 0 or 1 as the agent is fairly confident of the true hypothesis. It is the worst when q_n= 0.5 (perfect uncertainty about the true hypothesis). Also note that the average payoff is more when the rho values are larger as the agents are more willing to wait for the right answer and don't loose much money in this waiting period.

To do the experiment to conduct whether human thinks the same as we expect in mathematical theories, we can calculate q_n in the ball game mentioned in section 3.3 and get matching $\alpha_n, \beta_{n,}\gamma_n$ from figure 17.

After we get $\alpha_n, \beta_{n,}\gamma_n$, we have parameters we need in Mturk experiment. By using three thresholds to run the code with pre-set epsilon (percentage of error), we can get am image like figure 3. If we choose epsilon = 0.2, we find the situation that maximized epsilon for thresholds is 0.2 and then get matching sample size.

Therefore, we get three thresholds from Q-learning and get sample size from three thresholds. With those parameters, we can publish survey on Turk to get response. Setting matching thresholds for exact state as correct number, if answers from workers are close enough to these thresholds, workers can get bonus, otherwise not. And thus, we can conduct the experiment and to check whether human acts as what we think.

References

[1] Bala, Venkatesh, and Sanjeev Goyal. "Conformism and diversity under social learning." *Economic theory* 17.1 (2001): 101-120.

[2] Banerjee, Abhijit V. "A simple model of herd behavior." *The quarterly journal of economics* 107.3 (1992): 797-817.

[3] Bandura, Albert, and Richard H. Walters. "Social learning and personality development." (1963).

[4] Denison, Rachel N., et al. "Humans incorporate attention-dependent uncertainty into perceptual decisions and confidence." *Proceedings of the National Academy of Sciences* 115.43 (2018): 11090-11095.

[5] Ellison, Glenn, and Drew Fudenberg. "Rules of thumb for social learning." *Journal of political Economy* 101.4 (1993): 612-643.

[6] Quiggin, John. "A theory of anticipated utility." *Journal of Economic Behavior & Organization* 3.4 (1982): 323-343.

[7] Raman, Ravi Kiran, and Srilakshmi Pattabiraman. "Selfish Learning: Leveraging the Greed in Social Learning." *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* (2018).

[8] Rhim, Joong Bum. *Aggregation and influence in teams of imperfect decision makers*. Diss. Massachusetts Institute of Technology, 2014.

[9] Rhim, Joong Bum, and Vivek K. Goyal. "Team Decision Making with Social Learning: Human Subject Experiments." *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018.

[10] Vaughan, Jennifer Wortman. "Making Better Use of the Crowd: How Crowdsourcing Can Advance Machine Learning Research." *Journal of Machine Learning Research* 18.193 (2018): 1-46.

[11] Zhou, Yanli, Luigi Acerbi, and Wei Ji Ma. "The Role of Sensory Uncertainty in Simple Perceptual Organization." *bioRxiv* (2018): 350082.