Notes on Design Considerations and Implementation of Revolt: A Collaborative Crowdsourcing System with Synchronized Stages (Draft)

Joseph Chee Chang (josephcc@cs.cmu.edu)

March 2017

This article describe some of the design decisions and patterns, implementations details, and interesting strategies we found useful while implementing the Revolt collaborative crowdsourcing system for labeling machine learning datasets. We hope these can be helpful for people who are interested in developing real-time collaborative crowdsourcing systems. Some of the implementation details described below are particular to the TurkServer library, but the design patterns and strategies should be general enough for researchers using other libraries.

For details about the Revolt system, please refer to the CHI 2017 paper available here: `http://joseph.nlpweb.org/blog/2017/05/06/CHI-revolt/`

Joseph Chee Chang, Saleema Amershi, and Ece Kamar. 2017.
Revolt: Collaborative Crowdsourcing for Labeling Machine Learning Datasets.
In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems.
ACM, New York, NY, USA, 3180-3191. DOI: http://dx.doi.org/10.1145/3025453.3026044

1 Synchronized Stages

Revolt uses a Synchronized Stages pattern for facilitating real-time collaboration on crowdsourcing platform with three stages: Vote, Explain, and Categorize. The system employs multiple small ad-hoc teams of crowdworkers working in parallel on different parts of the input dataset. Within each stage, crowdworkers make independent judgments to be revealed to others in subsequent stages for collaboration. Using this pattern, Revolt can still benefit from the common crowdsourcing mechanisms of verification through redundant independent judgments (within stages), but can also capture and utilize the diverse crowd perspectives through collaboration (across stages). To the best of our knowledge this is the first crowd-based system that utilized this pattern.

As an running example, the scenario we set out to investigate is when crowdworkers are assigning predefined labels to items in a dataset for training machine learning models (e.g., labeling images with "cat" or "not cat" labels). However, in many cases, the labeling guidelines may not be comprehensive and leave rooms for alternate interpretations for some items (e.g., should an image of a tiger be labeled as "cat" or not, or how to judge if a web search result is relevant enough, see the Revolt paper for more examples). To address this problem, Revolt utilized crowd collaboration to find concepts in the dataset that are unaware by the requesters. In the first stage of the Revolt system, crowdworkers label a subset of the images independently the same way traditional image labeling tasks are done on crowdsourcing platforms. In the second stage we compare and reveal their independent judgements to each other in real-time when an item received different labels from different crowdworkers. We then ask the crowdworkers to explain their labeling choices and discuss with each other to generate a new label (e.g., "tigers") for the ambiguous items. Afterwards, the requesters can review these concepts discovered by the crowdworkers who collectively reviewed the entire dataset, and freely assign items associated with these concepts to one of the predefined labels (e.g., assigning all "tiger" items as "cat", and all "cartoon cat" items as "not cat").

The independent nature of the first Vote stage is crucial for capturing all ambiguous items. Since crowdworkers are typically optimizing for generating an acceptable labels as quickly as possible, if the judgements from other crowdworkers are visible during the first stage, workers might be swayed to pick the label others have already picked out, and converging on a single label even for items that are ambiguous and open to debate. On the other hand, the synchronized nature between the first and second stages is also crucial to reduce the amount of work. By identifying the items that received different labels in real-time, workers only need to work on the small subset of ambiguous items in the following Explain and Categorize stages.

2 Awareness

We made the decision of allowing the crowdworkers to be fully aware of the collaborative nature of the tasks, and found it to be beneficial. In general, crowdworkers who chose to accept our HIT found collaboration to be enjoyable, citing "it was fun to work with other people" and that they "discovered interesting things from others" in the post-HIT survey. We've also received higher quality data while workers are collaborating with each other. In the Explain stage of Revolt, where crowdworkers are providing short explanations for their labels on items that received different labels, we received better explanations when the workers are aware that these explanations will be shown to others. For example, explanations that contained detail description of the items such as "this is a tiger, which sometimes are referred to as big cats" instead of just "big cats are cats".

With the Synchronized Stages pattern, it is very apparent to the crowdwork-

ers that there are other people working on the task with them, since they have to wait for others in the lobby and between synchronized stages. However, there are still some additional information in each stage that we exposed (or chose not to expose) to the workers that:

- Peer comparison: In the Vote stage, workers were informed that others are labeling the same items, and that their labels will be compared in the following stages. The idea is that this can serve as a incentive for paying more attention and providing quality work.
- Peer pressure: In the Explain stage, workers were informed which items received conflicting labels, but we did not expose the distribution of these labels. We found that exposing the distribution will cause some of the workers to quickly agree to the majority instead of providing their reason for picking the different labels in the first place.
- Helping others: In the Explain stage, we asked the workers to provide explanations that focus on describing the items to help others understand their judgement, so they the crowdworkers don't get overly defensive about their judgements.

3 Controlling Dropouts

In traditional crowdsourcing settings, workers returning a HIT without completing the tasks does not cuase serious harm to the system. The data are discarded by the Mturk platform, and payments are not made. However, in the collaborative settings, if one worker in the group left before the tasks is completed, the collected data would be generated less collaboration. Further, in our opinion the workers should not be punished by other crowdworkers leaving early, and they have also expressed their disappointments in our surveys when they were unable to finished the task due to other workers leaving early. Therefore, in the Revolt system if some crowdworkers decided to return their HITs, the system will still allow the remaining workers to continue with their remaining stages and receive full payments. Due to these reasons, controlling dropout rate is crucial to both lowering cost and increasing the diversity of the collected data for collaborative crowd systems employs small ad-hoc teams working in parallel. During development, a lot of efforts was put into iterative design to figure out how to lower the proportion of crowdworkers leaving early, and we have successfully lowered the dropout rate of the system from around 50% to less than 5% using the following strategies:

- Length of stages: Intuitively, the longer your tasks are, the higher the chance of some workers leaving before the task ends. In addition, workers should receive the same amount of work in the same stages to minimize the wait time between the synchronized stages.

- Awareness of collaboration: In the task preview screen, make clear the collaborative nature of the task, providing time estimates while explaining how dropping out early may effect other crowdworkers.

- Awareness of the tasks: In the task preview screen, provide example tasks so crowdworkers knew what to expect if they accepted the HIT. For example, some workers might be willing to label cat images, but not news articles, and we do not want them to only find out in the middle of the task and thus returning the HIT.

- Progress and Notification: During the tasks, we gave clear progress indicators which showed the amount of bonus earned so far, and how much they can earn by completing the remaining stages. When the system moved crowdworkers to the subsequent stage synchronously, desktop and audio notifications were sent to all workers so that if they can come back to the browser tab and continue working on the next stage.

- Payment Structure: Revolt consists of three stages, and we pay 1USD for each stage. We designed each stage to be of similar difficulty and work duration. The idea behind this payment structure is that if workers are satisfied with the workload and payment of each stage, they should be motivated enough to continue and finish the next stage.

- Lobby: To form small ad-hoc groups, crowdworkers who have already accepted the HIT still need to wait until there are enough workers in the Lobby to start the experiment. In our experience, many workers will rush to accept our HITs soon after posting forming multiple ad-hoc teams of three workers. However, subsequent workers may need to wait for a short period of time in the lobby before enough workers have arrived. One issue we had encountered is that some workers may be idling when the experiment automatically starts, causing them to timeout quickly in the first stage. Therefore, instead of automatically starting new experiments whenever there are enough workers in the lobby, we implemented a ready button that expires every 3 minutes, and only started new experiments whenever there are enough readied workers. The ready button is only activated whenever there are enough workers in the lobby. This ensures whenever a new experiment starts, all the workers in the experiment had pressed the ready button within the past 3 minutes. In addition, we also send each worker desktop and audio notification whenever a new worker arrived at the lobby, so that they do not have to be constantly check the status of the lobby.

4 HIT Posting Strategies

4.1 HIT Duration

HIT duration should be set to longer than the estimated lobby wait time plus the time to finish all stages and the exit survey, so that workers have enough time to complete the task. On the other hand, since many workers have the habit of accepting and queuing up lots of HITs they deemed worth doing, if you set an overly long HIT duration, you could end up with a lot of workers clogging up your lobby and intentionally not starting the experiment. You also want to make sure that you indicated how long the task will take in the preview screen, so workers know that if they queue up your HITs for too long after accepting them, they might not have enough time to finish the tasks.

4.2 HIT List Ranking

Since many crowdworkers frequently check the HIT list for most recently posted HITs, one way to increase your HITs' exposure is to post them in batches. For example, if you need 30 assignments, you can post 10 at a time over 10 minutes, that way your HITs are constantly in the first page of most recent HITs on mTurk. Another posting trick is to add assignments whenever you have a few people waiting in the lobby, that way you increase the chance of having enough workers to start the experiment. It would probably be a fun project to automatically learn the optimal posting strategy for real-time collaborative crowdsourcing, but for the Revolt system we basically post batches of assignments over a short period of time and was able to recruit enough workers in a reasonable amount of time for our tasks.

5 Implementation Details

In TurkServer terminology, an Instance (or a world) consists of three synchronized stages: 1) the "Lobby" where crowdworkers wait for the experiment to start, 2) the "experiment" where a group of crowdworkers collaborate to finish the task defined by the requester, and 3) the "exit survey" where crowdworkers answers a post-survey and submit the HIT independently. These three stages are "synchronized" in that all workers must enter each stages together. For example, workers wait until there are enough people in the lobby to start the experiment in a group, and finish the tasks in the experiment stage together before they can move on to the exit survey and submit the HITs.

To implement your own synchronized sub-stages during the TurkServer you simply need to keep track of each worker's progress in each stages, and move all workers to the next stage when all of them are done with the current stage. You might be tempted to use the Meteor.users object provided by TurkServer to keep track of user data, but there are security measures around the user object to prevent syncing additional attributes across crowdworkers. Simply create

an additional ‘TurkServer.partitionedCollection‘ to keep track of each workers progress.

For example, in the Revolt System there are three synchronized stages: Vote, Explain, Categorize. All workers are initialized with the "Vote "status which also inform their frontend code to render the Vote stage interface. Workers who completed the Vote stage are assigned the "Voted" status, which puts them in a waiting room where they can keep track of the progress of other workers. And finally, when all active workers are assigned the "Voted" status, the backend sets the status of all workers "Explain", which moves all workers to the next stage synchronously.

5.1 Asynchronized Survey Stage

In most TurkServer tutorial and example projects, all crowdworkers in a "experiment" stage move on the "exit survey" stage synchronously, i.e., the system waits until all the workers are done with the task and calls ‘TurkServer .Instance .currentInstance() .teardown()‘ which moves all the workers to the "exit survey" stage. In our case, since we have multiple synchronized sub-stages during the experiment stage (vote-explain-categorize), we did not require workers to wait for each other to finish in the last stage before they can move on to the exit survey and submit their HIT. In TurkServer, this can be done by calling ‘TurkServer .Instance .currentInstance() .sendUserToLobby(user._id)‘ for individual workers, since the built-in Assigner sends workers who entered the lobby for a second time to the exit survey. Note that you should check if all your workers in an Instance has left the experiment stage and call ‘TurkServer .Instance .currentInstance() .teardown()‘ so that the system knows the experiment has ended and the TurkServer backend admin interface can render and time the experiments correctly.

5.2 Controlling Redundancy

- Task Redundancy: For diversity, if you do not want a single worker to label all partitions of your dataset, you can limit the number of HIT a worker can accept by setting the TurkServer settings "turkserver.experiment.limit.batch" to the number of HIT a single crowdworker can accept.

- Global Redundancy: If you have multiple conditions, and do not want to use the same crowdworkers to be exposed to different conditions, you can assign seen workers with custom mTurk qualifications, and reject those workers in your future HIT postings. This is a general strategy commonly used when developing crowd based systems for testing different conditions.