Reducing Error in Context-Sensitive Crowdsourced Tasks

Daniel Haas*
AMP Lab, UC Berkeley
dhaas@cs.berkeley.edu
Berkeley, CA, USA

Matthew Greenstein, Kainar Kamalov, Adam Marcus†, Marek Olszewski, Marc Piette
Locu, Inc.
{matt, kkamalov, marcua, marek, marc}@locu.com
Cambridge, MA/San Francisco, CA, USA

Introduction

The recent growth of global crowdsourcing platforms has enabled businesses to leverage the time and expertise of workers world-wide with low overhead and at low cost. In order to utilize such platforms, one must decompose work into tasks that can be distributed to crowd workers. To this end, platform vendors provide task interfaces at varying degrees of granularity, from short, simple *microtasks* (e.g., Amazon's Mechanical Turk) to multi-hour, context-heavy tasks that require training (e.g., oDesk).

Most research in quality control in crowdsourced workflows has focused on microtasks, wherein quality can be improved by assigning tasks to multiple workers and interpreting the output as a function of workers' agreement. Not all work fits into microtask frameworks, however, especially work that requires significant training or time per task. Such work is not amenable to simple voting schemes, as redundancy can be expensive, worker agreement can be difficult to define, and training can limit worker availability.

Nevertheless, the same characteristics that limit the effectiveness of known quality control techniques offer unique opportunities for other forms of quality improvement. For example, systems with worker training modules also possess a wealth of context about individuals and their work history on specific tasks. In such a context-heavy crowd work system with limited budget for task redundancy, we propose three novel techniques for reducing task error:

- A self-policing crowd hierarchy in which trusted workers review, correct, and improve entry-level workers' output.
- Predictive modeling of task error that improves data quality through targeted redundancy. When workers complete tasks, we can allocate spot-checked reviews to the tasks with the highest predicted error. This technique allows us to capture 23% more errors given our reviewing budget.
- Holistic modeling of worker performance that supports crowd management strategies designed to improve average crowd worker quality and allocate training to the workers that need the most assistance.

*Research conducted while an intern at Locu, Inc.
†Contact author at Locu.

Redundancy outside the World of Microtasks

Locu's largest use of crowd work is in parsing the contents of price lists (e.g., menus) from unstructured documents (e.g., PDF, Flash, HTML) that can be discovered on local merchants' (e.g., restaurants') websites. Workers, in combination with automated machine learning extractors, are tasked with converting unstructured documents into structured, wikitext-like syntax to annotate elements like menus, sections, items, and prices. Such work is poorly suited for microtask decomposition: menu items and sections can be interdependent (e.g., a list of toppings at the top of a pizza menu might affect pricing options for a pizza at the bottom of the menu) and worker output is not a simple categorical answer (we process a complex structured document).

Assigning multiple workers to every task is expensive, and it is difficult to reconcile multiple responses. We do use redundancy, however, by assigning trusted workers to review a sample of other workers' output. Our hierarchical approach has three levels. *Data Entry Specialists (DES)* process incoming tasks to completion. *Reviewers* look at tasks completed by DES, make corrections, provide feedback, and ask the DES to make additional changes before approving the task. In a third step, Reviewers might review other Reviewers' work for continued education and quality. *Managers* serve a cross-cutting role by arbitrating disagreements and holistically training workers. An example task lifecycle is shown in Figure 1. We have not utilized our hierarchical design outside of business and price lists, but we believe our design can generalize to other complex workflows.

This hierarchy has several benefits. First, it's iterative: workers benefit from the experience of previous workers who have completed the task. Second, promoting the highest quality workers improves the odds that reviews will catch lingering errors. Finally, by giving workers higher in the hierarchy trust and responsibility, we increase their commitment to the system and motivation to perform well. Since we implemented this hierarchy iteratively and specifically to support Locu's workflows, we have not yet evaluated the system against other organizational strategies.

Reducing Task Error with Predictive Modeling

Ideally, with a hierarchical crowd like the one described above, every task would be reviewed at each level of the hierarchy. Unfortunately, the cost of assigning a trained worker

to a long-running, context-heavy task makes such an approach prohibitively expensive. As such, a crowd system must trade off budget expenditure against data quality by selecting a subset of tasks for review up the hierarchy. The optimal spending strategy given a budget of n dollars with a task cost of m dollars per task is to review the $\frac{n}{m}$ tasks with the highest error. (As a practical matter, any strategy to predict high-error tasks must also allocate a budget for randomly reviewing tasks to train on in the future.)

First, we must define *Task Quality*. Task quality is the fraction of lines that are incorrect (e.g., 1.0 represents a completely incorrect task, and 0.0 represents a perfect task). We can estimate this value for tasks that have already been reviewed: we take a diff between the output of the original worker and the output of the reviewing worker, and calculate the percentage of lines that were changed. Our challenge, however, is that we must estimate task quality in tasks before review, so that we can select the worst tasks for review.

To predict task quality of unreviewed tasks, we trained a supervised boosted random forest regression model (called the TaskGrader). We train using a sample of several hundred thousand previously reviewed tasks. Each task is featurized, with features broken down along two axes depicted in Table 1: *task-specific* vs. *worker-specific* features, and *generalizable* vs. *domain-specific* features. Task-specific features are those which are only dependent on the task itself, whereas worker-specific features are those which are built from the worker's profile. Generalizable features are those which could be used in any crowd system with basic bookkeeping about tasks and workers, whereas domain-specific features are specific to Locu's problem domain.

We are in the process of fully evaluating the TaskGrader to determine its impact on reducing task error. Preliminary results show that using such a model to select tasks for review results in significant improvement over a random selection strategy. Figure 2 shows the ROC curves generated by reviewing varying proportions of tasks (from 0% to 100%) for the TaskGrader's review strategy and a random sampling review strategy. The TaskGrader has a significantly better AUC than the random sampler, 0.73 to 0.50. Given our current review budget, implementing the TaskGrader at Locu has resulted in 23% more errors being caught.

Future Directions: Inferring Worker Quality

Although avoiding error at task output time is important, it only attacks the symptoms of poor data quality, not its causes. Workers who are unattached or under-trained are likely to perform poorly on tasks, resulting in systemic data quality problems. Reducing the rate of this type of error is more efficient than attempting to improve data quality at task output time, as improvements to the workforce will benefit all future tasks. A crowd's net worker quality can be improved by ranking the workers' quality, intervening with or demoting poor workers, and promoting good workers. The research question here, stated simply, is "How do we best model workers to determine their overall quality and to identify correctable behavior?" We are actively investigating two approaches to this problem: detecting workers with anomalous behavior to identify spamming or systematic biases, and clustering workers by performance to rank their quality. Improving our model of workers should inform the TaskGrader model described above, and vice versa.

Related Work

Most research on quality control in crowdsourcing systems has focused on the microtask setting, both on improving task quality (Callison-Burch 2009; Le et al. 2010; Snow et al. 2008) and worker quality (Downs et al. 2010; Ipeirotis, Provost, and Wang 2010; Le et al. 2010). Other work has described hierarchies of crowd workers, (Kochhar, Mazzocchi, and Paritosh 2010; Kulkarni et al. 2012) but there has been little published evaluating the approaches. Machine learning has been used to predict task quality, (Rzeszotarski and Kittur 2011) but such work is focused on features related to user click and event logging rather than rich worker profiles and history, and task-specific features.

References

Callison-Burch, C. 2009. Fast, cheap, and creative: evaluating translation quality using Amazon's Mechanical Turk. In *EMNLP '09: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Downs, J. S.; Holbrook, M. B.; Sheng, S.; and Cranor, L. F. 2010. Are your participants gaming the system?: screening mechanical turk workers. In *CHI '10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Request Permissions.

Ipeirotis, P. G.; Provost, F.; and Wang, J. 2010. Quality management on Amazon Mechanical Turk. In *HCOMP '10: Proceedings of the ACM SIGKDD Workshop on Human Computation*. ACM Request Permissions.

Kochhar, S.; Mazzocchi, S.; and Paritosh, P. 2010. The Anatomy of a Large-Scale Human Computation Engine. *Proceedings of the ACM SIGKDD Workshop on Human Computation* 10–17.

Kulkarni, A.; Gutheim, P.; Narula, P.; Rolnitzky, D.; Parikh, T. S.; and Hartmann, B. 2012. MobileWorks: Designing for Quality in a Managed Crowdsourcing Architecture. *IEEE Internet Computing ()* 16(5):28–35.

Le, J.; Edmonds, A.; Hester, V.; and Biewald, L. 2010. Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution. *Proc. SIGIR 2010 Workshop on Crowdsourcing for Search Evaluation* 21–26.

Rzeszotarski, J. M., and Kittur, A. 2011. Instrumenting the crowd: using implicit behavioral measures to predict task performance. In *UIST '11: Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM Request Permissions.

Snow, R.; O'Connor, B.; Jurafsky, D.; and Ng, A. 2008. Cheap and Fast — But is it Good? Evaluating Non-Expert Annotations for Natural Language Tasks. *Proc. ACL '08*.

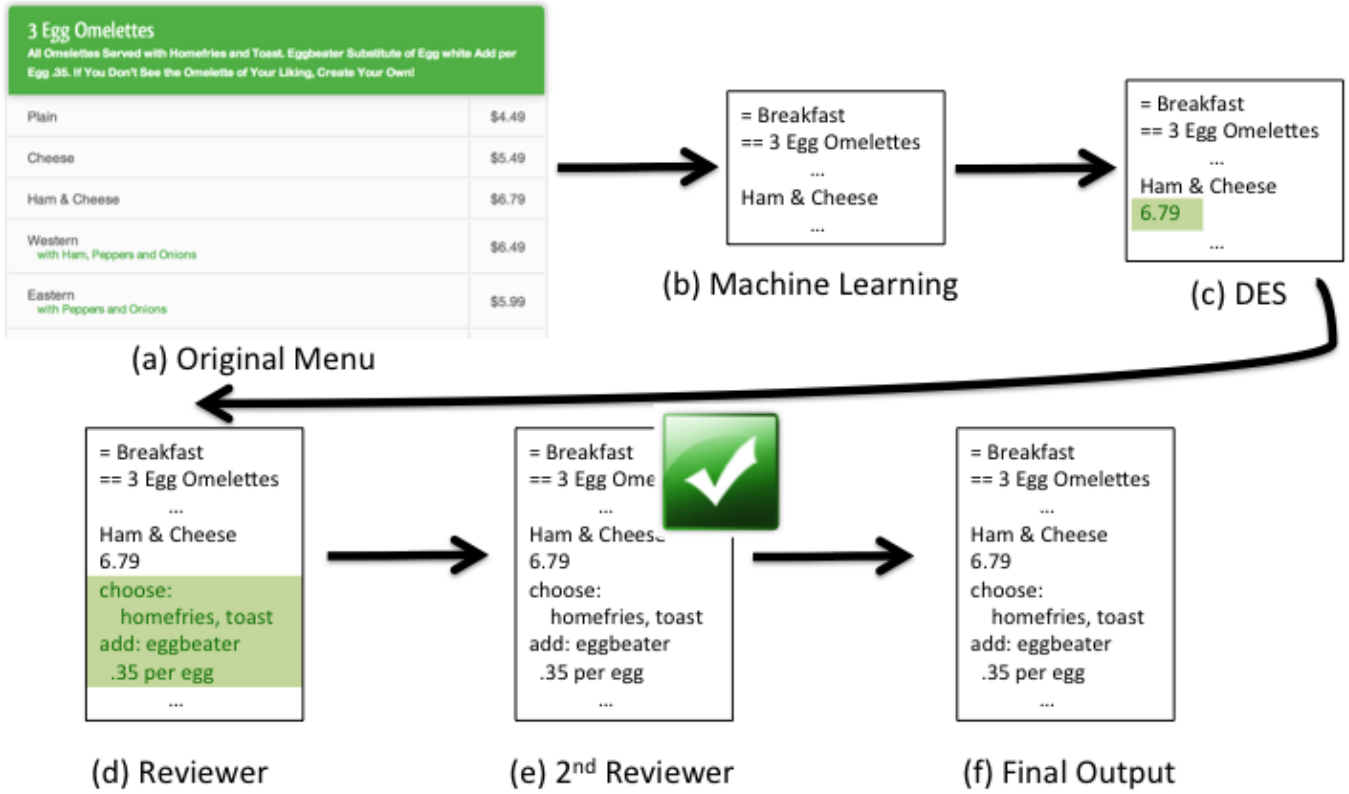

Figure 1: Example task lifecycle through the Locu crowd hierarchy. Menus are found online (a), then passed through machine learning extraction to discover menu sections, subsections, and items (b). Next, Data Entry Specialists attempt to fill in data that the algorithms miss (c), and Reviewers correct mistakes that were made by the previous workers (DES or lower-level Reviewer) before approving the task (d, e). The final output has the wikitext-like syntax shown in (f).

	Task-Specific	Worker-Specific
Generalizable	time spent on task, time-of-day task started, task length, etc.	worker hierarchy role, worker age in system, worker's past task quality, etc.
Domain-Specific	average items per menu section in task, task menu source url, price options per item in task, etc.	worker's past menu categories, worker's past spelling errors, etc.

Table 1: Examples of TaskGrader features, categorized by generality and source. Domain-specific features are examples of features used with Locu's Menu-parsing tasks.

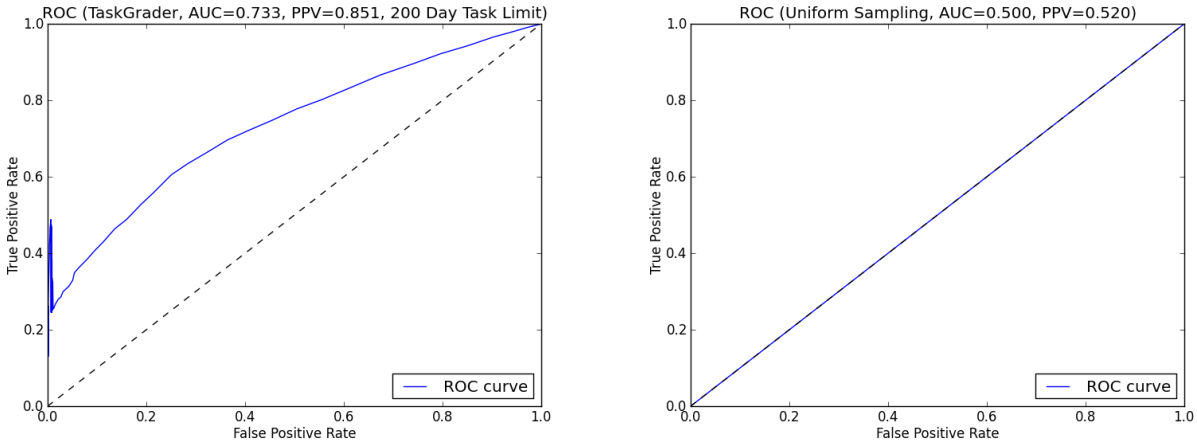

Figure 2: ROC curves for selecting tasks for review with varying thresholds of what makes a 'bad' task. The TaskGrader model has an AUC of 0.73 (left), while a uniform sampling strategy performs much more poorly (right).