Self-Improving Crowdsourcing: Near-Effortless Design of Adaptive Distributed Work

Jonathan Bragg

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Daniel S. Weld, Chair

Mausam, Chair

Ece Kamar

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Self-Improving Crowdsourcing:
Near-Effortless Design of Adaptive Distributed Work

Jonathan Bragg

Co-Chairs of the Supervisory Committee:
Professor Daniel S. Weld
Computer Science & Engineering

Associate Professor Mausam
Computer Science & Engineering

Systems that coordinate work by many contributors have enormous potential to solve many of the most pressing problems facing society today. In particular, crowdsourcing systems enable data collection at scale, critical to accelerating scientific discovery and supporting a host of new machine learning applications. However, ensuring these systems are cost-effective and produce high-quality data remains a key challenge, and one of central importance to downstream applications. Creating an efficient, successful crowdsourcing task requires significant time investment and iteration to optimize the entire task pipeline: recruiting workers who have the requisite knowledge, defining and communicating the task requirements, training and testing workers on those requirements, routing tasks to workers in a skill-aware manner, and prioritizing tasks to avoid wasted effort. These design costs underlie nearly every reported crowdsourcing success, yet are seldom acknowledged and therefore often underestimated. This initial investment makes crowdsourcing impractical for all but the largest tasks; in many cases, it may actually be less costly for the task designer ("requester") simply to perform the task herself.

The central thesis of this dissertation is: *high-quality, efficient crowdsourcing tasks can be created at low cost through self-improvement meta-workflows combining algorithms, workers, and minimal requester involvement.* Toward this end, I present methods for automating or semi-automating the design of many stages of the task pipeline, thus reducing the burden of the task designer:

- Chapter 3 presents algorithms for efficiently recruiting workers with the requisite knowledge based on their digital footprints, reducing the number of recruiting requests the requester needs to issue.

- Chapter 4 provides algorithms for managing recruited workers by optimizing the amount of training or testing they receive. These algorithms outperform common ad-hoc requester policies and require no tuning by the requester.

- Chapter 5 presents algorithms for routing tasks to all trained workers in parallel in a skill-aware manner. These methods also outperform baselines policies and do not require hand-tuning.

- Chapter 6 provides algorithms for prioritizing tasks to reduce wasted effort on multi-label classification tasks, a common type of task. These methods use less than 10% of the labor of previously-used requester policies.

- Chapter 7 presents a tool that helps the task designer rapidly specify and improve the task design and instructions, by enabling prioritized navigation of the dataset (through worker-surfaced ambiguous categories of questions) and semi-automated workflow creation (through suggested questions for training and testing workers). This work fully closes the loop, with algorithms, the requester, and workers all contributing to task self-improvement.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

ASSIGNMENT: A matching between a task instance and a worker.

CROWD: A group of people or workers commonly assembled by a platform (see "crowdsourcing platform") or as part of an online community..

CROWDSOURCING: In this dissertation, a method of completing work using a "crowd" of contributors, each of whom is responsible for performing a portion of the work.

CROWDSOURCING PLATFORM: A web application or marketplace that supports posting and performing work. The platform manages how worker are recruited and find work.

GATED INSTRUCTION: A method for screening workers, which involves training and testing them.

GOLD QUESTIONS: Questions with known answers, commonly used for worker testing.

IMPROVEMENT TASK: A type of meta-task aimed at improving a workflow.

META-WORKER: A worker tasked with an improvement task, rather than a participant in the primary workflow.

META-TASK: A task outside the primary workflow, e.g., issued by a meta-workflow.

META-WORKFLOW: A workflow (algorithm) that operates on a crowdsourcing workflow (e.g., to improve it).

ONLINE COMMUNITY: A group of people digitally assembled for a purpose (e.g., Wikipedia). A particular type of crowd.

QUESTION: See "task instance."

REQUESTER: The issuer of tasks. Commonly, the requester is also the task designer, and I use these terms interchangeably.

SELF-IMPROVING CROWDSOURCING: A new paradigm proposed in this dissertation, where a crowdsourcing workflow improves itself through a meta-workflow that coordinates the efforts of algorithms and workers, with minimal requester involvement.

SUBTASK: A task that is part of a workflow aimed at solving a different, primary task. Usually, subtasks are determined by breaking down a task into easier problems in a process known as *task decomposition*.

TASK: A particular class of questions to be crowdsourced.

TASK INSTANCE: A particular instantiation of the task, for example, a single image in an image-labeling task. Also referred to as a *question*.

TASK PRIORITIZATION: A partial ordering (possibly time-varying) of tasks or task instances that should be completed first.

TASK ROUTING: A task prioritization method that also takes into account assignments of task instances to specific workers.

WORKER: A contributor on an online work platform.

WORKER TESTING: A process that tests worker performance, usually involving inserting "gold" questions into the worker's stream of tasks.

WORKER TRAINING: Methods to improve worker skill or understanding of the task.

WORKFLOW: An algorithm for solving a crowdsourcing task. This algorithm can create instances of subtasks, and use worker (or machine) answers to those instances in later parts of the algorithm (e.g., as inputs to other subtasks), with the ultimate goal of solving the task.

ACKNOWLEDGMENTS

First, I would like to thank my advisors Dan Weld and Mausam for their unwavering support, dedication, guidance, inspiration, and friendship. I am so grateful for the tools—equally useful in and out of the lab—you have helped me to develop. I could not have asked for better mentors during my Ph.D.

I have been fortunate to have had many superb research mentors. I would like to thank Elaine Chew, Stuart Shieber, and Hyeong-Ah Choi for their early research support and encouragement; Ece Kamar, Dieter Fox, Andy Ko, Jeff Bigham, Haoqi Zhang, and Matt Lease for high-level feedback on this dissertation research; and Ed Chi, Peng Dai, and Igor Karpov for support while on internship.

While I have had many influential teachers, I would like to recognize Ben Taskar in particular. I only took one course with Ben, but he was exceptionally inspiring, kind, and generous with research advice.

I would like to thank my friends and colleagues, especially the ones I have met during grad school: Ofra Amir, Lydia Chilton, Emad Soroush, Chris Lin, Xiao Ling, Andrey Kolobov, Eunsol Choi, Svet Kolev, Vikash Kumar, Shih-Wen Huang, Gagan Bansal, Mandar Joshi, Quan Ze Chen, Adam Marcus, Adish Singla, Pao Siangliulue, Darren Kuo, Anand Kulkarni, Anand Sriraman, Tracy Tran, Laci Lovasz, William Agnew, Vlad Murad, and many others. A very special thank you to Aileen Granstrom.

I am grateful to the (many) people who have participated in my experiments, as well as the funding agencies who have made this research possible, in particular the National Science Foundation and the Office of Naval Research.

Finally, I would like to thank my family for their unconditional love: my father for my introduction to computer science, my mother for her loving encouragement, and my sister Danielle Bragg for being the best lifelong friend and colleague.

DEDICATION

To my family.

Chapter 1
INTRODUCTION

1.1 Motivation

Much progress throughout history has been enabled by people working together. The advent of communication technologies leading to the internet in the 20th century allowed for geographically distributed groups of people to collaborate. More recently, digital work platforms such as Zooniverse[1] and Amazon Mechanical Turk[2] have enabled scientists and engineers to collect data from people around the world at unprecedented scale. This method of data collection, known as "crowdsourcing," has produced numerous scientific discoveries and is critical to building the next generation of machine learning systems.

The effectiveness of crowdsourcing depends to a large extent on iterative design and experimentation. Failure to sufficiently iterate can lead practitioners to conclude falsely that crowdsourcing is not helpful [110, 169]. Conversely, investing heavily in process improvements can pay large dividends. A particularly notable example of these dividends is FoldIt, a crowdsourcing community that has enabled several scientific breakthroughs [29]. FoldIt's success can be attributed to getting many critical design elements right, including providing a sophisticated tool that helps people find solutions by combining their efforts with algorithms, recruiting an active community, training new members through effective skill-appropriate tutorials, and creating a feedback loop that improves the tool's usefulness through solutions and macros provided by the community. The long-term vision of this dissertation is to lower the

[1] `https://www.zooniverse.org/`

[2] `https://www.mturk.com/`

barrier to creating tools like FoldIt, which effectively combine the efforts of humans and machines and improve over time.

Unfortunately, coordinating the efforts of many distributed workers (the "crowd") remains a costly process. In order to obtain useful (high-quality) data from crowdsourcing, one must try a large number of crowdsourcing task designs. Even for experienced crowdsourcing task designers ("requesters"), best practice task design involves prototyping and iterating on possible designs, and progressively trying out designs first on oneself, then on one's friends, and finally on crowdsourcing platforms. Design guidelines published by crowdsourcing platforms and researchers may prove useful starting points, but general design principles are not well understood, and novice requesters may be guided by intuition alone with varying degrees of success [130]. Even once one arrives at a well-performing crowdsourcing task, tuning the parameters of that task, such as the amount of training or testing workers receive, is difficult and the optimal settings may result in a complex policy that requires expertise in artificial intelligence (AI) to program [162].

1.2 Self-Improving Crowdsourcing

In order to reduce the large, often prohibitive cost of solving new problems using crowdsourcing, this dissertation proposes *self-improving crowdsourcing*. Traditionally, the requester is the sole entity responsible for crowdsourcing process (workflow) improvements. Self-improving crowdsourcing is a more general paradigm that shifts much of the burden of task design and improvement from the requester to a meta-workflow (a workflow that operates on a workflow), which also uses algorithms and workers to improve the crowdsourcing process (see Figure 1.1). Moreover, this paradigm seeks to minimize effort by the requester, whose primary job is to provide the task specification detailing what kind of outputs should be produced (since only the requester may know the context in which the outputs will be used). It also seeks to minimize the overall improvement cost, to maximize the usefulness of the approach.

This dissertation posits that *high-quality, efficient crowdsourcing tasks can be created at low cost through self-improvement meta-workflows combining algorithms, workers, and minimal requester involvement.* Crowdsourcing is a joint effort performed by the requester (who traditionally designs and manages the task), workers (who complete instances of the task, but can also serve as "meta-workers" that assume some of the requester's traditional responsibilities), and algorithms (that can assist either the requester or workers); thus, the process can "self-improve" (e.g., by reducing cost or improving answer accuracy) through the efforts of any of these parties. In the true spirit of crowdsourcing, which seeks to recruit the efforts of many people other than the requester, this dissertation shows that workers and algorithms can perform central self-improvement roles.[3]

This dissertation demonstrates the feasibility of self-improving crowdsourcing with experiments that focus primarily on labeling tasks. Labeling tasks are useful to study for several reasons: (1) they represent the most common task type [72], (2) worker answer correctness can be verified if the true answer is known (required for common worker testing procedures (Chapter 4)), and (3) worker answers can be aggregated using automated methods to support estimation of the true question answers and worker abilities. While we focus on labeling tasks for the above reasons, the proposed methods may be applicable to other task types, e.g., the recruiting methods presented in Chapter 3 are immediately useful for other task types.

Further, this dissertation considers task improvement problems that do not require searching through many possible ways of decomposing a task into subtasks. In the crowdsourcing literature, workflows often refer to algorithms that solve complex problems through the use of subtasks [12, 27, 126]. Deciding how to define and combine subtasks is a difficult design problem [130]. While this dissertation *does* present meth-

[3]Self-improvement refers to improvement of the crowdsourcing process, not specific crowdsourcing participants, though in certain cases, participants may improve the process by improving themselves (e.g., in the case of a machine-learning control algorithm that improves its own performance).

ods for improving workflows—for instance, deciding when to test workers (Chapter 4) and how to prioritize questions for multi-label classification (Chapter 6)—it does not seek to solve the general workflow optimization problem, which requires reasoning over a much larger design space that also includes task decomposition and remains beyond the state of the art except in restricted settings [42, 105]. This dissertation takes a step toward this larger goal in the sense that general workflow improvement benefits significantly from task-level improvement (since workflows are composed of subtasks), and may also benefit from techniques similar to those proposed in this dissertation.

1.2.1 Benefits of Self-Improving Crowdsourcing

Self-improving crowdsourcing has many benefits:

- **Reduced requester effort.** The requester does not need to spend as much time defining the task or experimenting with the process to optimize task performance.

- **Reduced cost.** Optimization reduces the need for redundant work, significantly increasing the amount of useful data generated for a given budget. This increased productivity can make crowdsourcing attractive for new applications, or increase impact for larger budgets.

- **Higher quality answers.** Effective design can have a major impact on data quality and downstream applications that make use of that data. For instance, improving the worker training process on an NLP (relation-extraction) task led to significant increases in both crowdsourced data quality and performance of classifiers trained on that data [110].

- **Better task throughput.** Careful task design and effective training have the potential to increase the pool of useful contributors, thus allowing tasks to be

completed more quickly. In contrast, simply filtering workers, by qualification tests or other means, without first performing task improvement eliminates many workers who may have been useful contributors had they understood the task better or received better instruction.

- **Lower risk to workers.** Based on a recent survey of Amazon Mechanical Turk workers, McInnis et al. [119] found that unfair rejections of work have a major impact on worker livelihoods and identify seven major risk factors for rejection. Task improvement helps to address the first two of these risk factors: flaws in the task design and unclear evaluation criteria. While requesters often blame workers for unsuccessful crowdsourcing efforts, inadequate attention to task design and other human factors is often to blame [98].

1.2.2 Workflow and Meta-Workflow Optimization

The varying roles of workers and the requester in self-improving crowdsourcing can be understood in a reinforcement learning AI framework (see Figure 1.1). In this framework, a self-improving crowdsourcing agent (algorithm controlling the process) takes actions by interacting with workers (e.g., by issuing a question) and observing their answers. In addition to performing actions for the primary task, the agent can also perform *meta*-actions, which allow the agent to subgoal on improving the process rather than actually performing the end task. For example, a meta-action could have a worker suggest a change to the instructions, which the agent can use to help the requester construct new actions that improve the agent and enable it to collect higher-quality answers that more closely match the requester's reward function (agree with the requester's definition of what is a correct answer).

Designing a self-improving crowdsourcing agent is more challenging than many traditional reinforcement learning problems in several ways. First, the agent can observe worker answers but not the true rewards, since the system does not know in

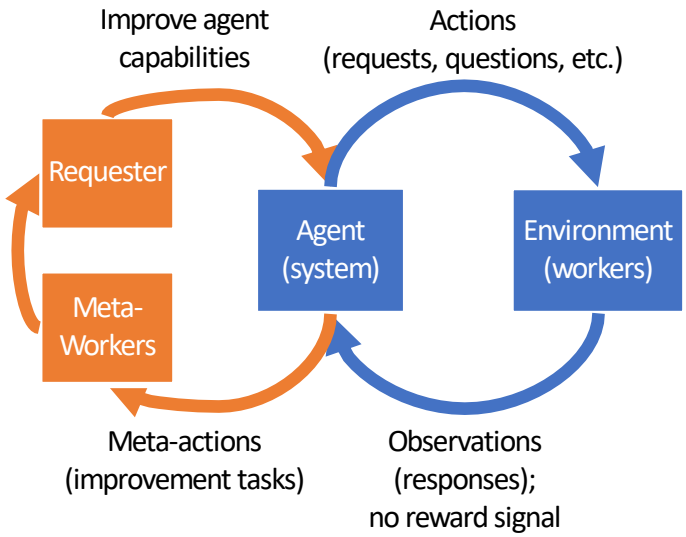


Figure 1.1: Self-improving crowdsourcing, viewed in a reinforcement learning framework. The right side of the figure (in blue) is the traditional RL agent-environment interaction, where the agent performs self-improvement through interaction with workers. Different from standard RL settings, rewards are not observable, since crowdsourcing seeks to answer questions with *unknown* answers. The left side of the figure (in orange) depicts the self-improvement roles of the meta-workers and requester.

general if worker answers are correct. Second, it is difficult (or impossible) for the requester to communicate her true reward function, and the requester's own understanding of her reward function may change as she sees more data from the task. Third, the action space of the agent often grows exponentially in the number of questions or workers involved, since the agent must assign questions to workers. Finally, the agent must learn in a complex, high-stakes human environment without access to a simulator (if one could simulate human performance on the task, crowdsourcing would be unnecessary), so it is essential that the agent learn in a safe, efficient manner.

In order to design an agent that can succeed in this challenging setting, I make use of several strategies. The first strategy is to formulate submodular optimization problems [91], which can be solved near-optimally using myopic algorithms that handle large, combinatorial action spaces well (Chapters 3, 5, and 6). The second strategy is to formulate problems using model-based representations that support planning with longer horizons, when action spaces are smaller (Chapter 4). The final strategy is to use humans to design new actions that improve outcomes (Chapter 7). These strategies use compact representations that enable efficient learning when there is no simulator and tasks may not involve a large number of questions.

Table 1.1 summarizes the action spaces available to the agent at each stage of the task pipeline (see Figure 1.2), as well as the primary actors in the self-improving feedback loop that are evaluated in this dissertation. While the table only notes the primary actors, this dissertation proposes several additional supporting roles that could be performed by requesters (or perhaps even workers) to further help address some of the challenges described above. For instance, requesters could perform feature engineering to improve the agent's model of the environment (Chapter 3 presents experiments demonstrating that features can predict whether workers will respond to recruiting messages). Similarly, requesters could specify an initial policy based on external knowledge or prior experience, to encourage safe exploration (Chapter 4 presents an agent that can make use of this initial policy). Finally, requesters can provide new actions (beyond instructions) that can improve performance of the agent system (Chapter 3 describes design experiments that enable the agent to recruit workers more effectively).

Achieving self-improving crowdsourcing requires innovation in both AI and Human-Computer Interaction (HCI). For instance, an HCI research methodology benefited my tool for helping the requester improve the task specification and instruction workflow (Chapter 7). Finding the right design involved experimenting with different ways to combine the efforts of workers, algorithms, and the requester in a meta-workflow,

Chapter	Topic	Agent Action Space	Self-Improving Actors
3	Recruiting	Designs × Workers × Items	Algorithms
4	Management	{*Work*, *Test*, *Train*, *Replace*}	Algorithms
5	Routing	Workers × Items	Algorithms
6	Prioritization	Labels	Algorithms
7	Improvement	{*Work*, *Filter*, *Organize*}, {*Diagnose*, *Clarify*, *GenTest*}, {*Refine*}	Algorithms, Meta-Workers, Requester

Table 1.1: The agent (crowdsourcing system) considers a different action space, or set of actions for each stage of the task pipeline. The action spaces here are either primary workflow actions (Chapters 3– 6) or meta-actions (Chapter 7; see Figure 1.1). Constraints on these action spaces are described in later chapters. Self-improving actors are the primary participants involved in the self-improvement feedback loop.

and to support nonlinear task exploration and improvement with the right user interface for the requester. AI, on the other hand, is particularly useful for optimizing parts of the task pipeline where one can define a tractable space of possible solutions,. In these settings, I present adaptive algorithms that learn from experience (e.g., by observing which test questions workers get right (Chapter 4)) and improve without requester involvement.

1.3 Dissertation Roadmap and Contributions

The remainder of this dissertation is structured as follows. Chapter 2 surveys previous and related work. Chapters 3– 7 comprise the body of the contributions toward self-improving crowdsourcing (described in more detail shortly). Chapter 8 concludes by

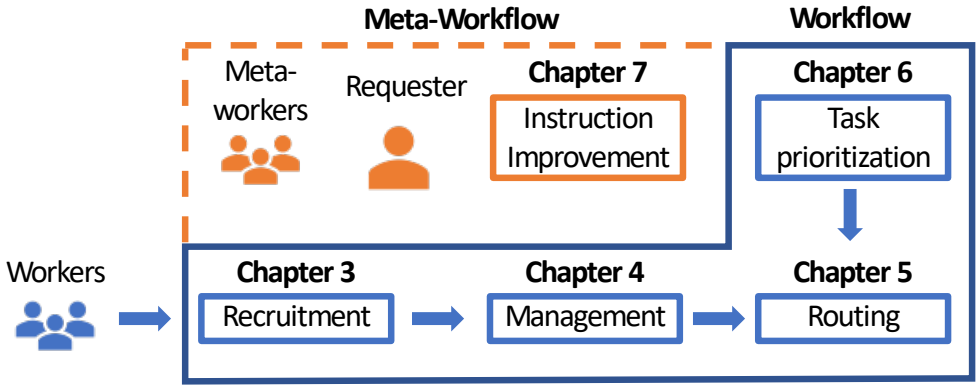


Figure 1.2: Chapters 3–6 deal primarily with algorithmic self-improvement. In Chapter 7, the requester and meta-workers take on more direct workflow improvement (through a more explicit meta-workflow).

summarizing contributions, offering takeaways for crowdsourcing system designers, and identifying possible future work.

In the following subsections, I describe the contributions of Chapters 3– 7. Figure 1.2 depicts the organization of these chapters and contributions visually. All the work in this dissertation was the result of collaborations, so I use *we* extensively to acknowledge my collaborators.

1.3.1 Chapter 3: Requesting Work from Outside Contributors Using Data Mining

Identifying people who are best suited to contribute to tasks is critically important to task success. Often, systems may need to look beyond the current pool of contributors; attracting contributions from these people is especially important to new online communities (e.g., StackOverflow), which must generate initial value before they can gain critical mass and become self-sustaining. Chapter 3 demonstrates how data mining experiments can inform the first decision-theoretic mechanism for requesting a set of outside contributions with the highest expected utility to a community. This method uses data mining to identify a set of people who are likely to contribute to tasks of

value (e.g., people who have tweeted something related to the task) and to estimate how likely they are to respond to a contribution request. By showing that a natural class of utility functions is submodular, we provide an efficient algorithm for selecting a provably near-optimal set of requests over the set of possible request designs and task assignments. Experiments with a young online community[4] demonstrate that this algorithm can generate as much community utility as baselines while issuing only 55% as many requests, by predicting high-value requests from textual features of how potential contributors have written about topics related to tasks and from initial estimates of effective request designs (e.g., we discovered asking contributors to make a small contribution first improved response rates). This work initially appeared in [71] and is joint work with Shih-Wen Huang, Isaac Cowhey, Oren Etzioni, and Daniel S. Weld. The open-source implementation of these algorithms is available at `https://crowdlab.cs.washington.edu/bootstrapping-online-communities.html`.

1.3.2 Chapter 4: Optimal Worker Testing and Training

An ideal system for supporting crowd work would notice when workers need additional training, provide opportunities for training, and disqualify workers who do not eventually produce high quality work to ensure successful outcomes for the system. However, the current standard for requesting work in online labor markets like Amazon Mechanical Turk consists of intermixing a flat 10–30% of test questions with known answers and disqualifying workers who answer too many incorrectly. Noting that the placement of test questions should instead *adapt* to actual performance, in Chapter 4, I present our approach, which formulates the problem of balancing between (1) testing workers to determine their accuracy and (2) actually getting work performed by asking questions with unknown answers as a partially-observable Markov decision process (POMDP) and applies reinforcement learning to dynamically

[4] `http://airesources.org/`

calculate the best policy. Evaluation with Mechanical Turk workers show that my agent learns adaptive testing policies that produce up to 111% more reward than the standard static policies. Furthermore, I present versions of the agent that also *train* workers and only disqualify workers who are unresponsive to training. This work initially appeared in [16] and [15], and is joint work with Mausam and Daniel S. Weld. The open-source implementation of this work is available at `https://crowdlab.cs.washington.edu/optimal-training-and-testing-for-crowd-workers.html`.

1.3.3 Chapter 5: Routing Tasks to All Available Workers

An optimal approach to coordinating work not only prioritizes work and monitors contributors, but also considers *who* should do what work. In Chapter 5, I present the first distribution mechanism that assigns tasks in parallel to all available contributors in a manner that is sensitive to worker skill and question difficulty. We first show that computing an optimal matching between workers with varying skill levels and work with varying difficulty is NP-hard for arbitrary utility functions, even if the skills and difficulties are known. Given this hardness result, we developed an approximation algorithm for computing a near-optimal matching, by showing that information gain is submodular in this setting, too. In live experiments, this algorithm uses less than half of the labor of previously-used algorithms. I also present extensions to the algorithm that partition similar workers and tasks so that task routing can be performed in real time in large-scale deployments. A key contribution of this work is characterizing the space of possible distribution mechanisms, and noting that ours is the first that does not assume that workers are always immediately available, or that only the best workers can provide value to the system. This work initially appeared in [14] and is joint work with Andrey Kolobov, Mausam, and Daniel S. Weld. The open-source implementation of this work is available at `https://crowdlab.cs.washington.edu/task-routing.html`.

1.3.4 Chapter 6: Prioritizing Tasks for Multi-Label Classification and Taxonomy Creation

Many hard problems require global understanding of large amounts of information, making them too difficult for a single person to solve alone. In response, researchers have developed algorithms to distribute work across multiple people, and to aggregate this work to achieve high-level understanding. For example, Chilton et al. [27] created an algorithm for organizing an unstructured dataset into a taxonomy that has crowd workers repeatedly generate possible labels and decide which labels apply to which items in the dataset in order to infer relationships between labels (such as whether a label should be the parent of another label in the taxonomy). This inference step was extremely expensive because it required $(k \times |\text{items}| \times |\text{labels}|)$ judgements, where k is the number of workers who are asked each question ($k > 1$ required for quality control).

Building on the observation that this inference step is an instance of a more general machine learning problem, Chapter 6 presents DELUGE, the first decision-theoretic algorithm for solving multi-label classification by collecting answers from workers, DELUGE has two components: (1) a probabilistic model to infer which labels apply to an item given noisy responses from workers, and (2) a decision-theoretic algorithm that uses this model to ask workers about labels that will maximize information gain about the latent label values for an item. By showing that information gain is *submodular* in our setting—it satisfies a mathematical definition of diminishing returns—we prove that a greedy algorithm is guaranteed to select near-optimal batches of questions to ask. Live experiments in an online labor market demonstrate that DELUGE solves multi-label classification problems using less than 10% of the labor of previous approaches, enabling taxonomy creation (or other downstream applications) for a fraction of the cost. This work on DELUGE previously appeared in [13], and is joint

work with Mausam and Daniel S. Weld. The open-source implementation is available at `https://crowdlab.cs.washington.edu/crowd-multi-label.html`.

1.3.5 Chapter 7: Efficiently Specifying Tasks and Improving Instruction Workflows

The work described up to this point uses AI to optimize the division of labor and to manage contributors, but makes a critical assumption: the task is well defined to begin with. If tasks are poorly specified and communicated, contributors will not produce consistent work, even if they are trained and managed with sophisticated policies. However, task design is hard because it involves a costly iterative process: identifying the kind of work output one wants (often not apparent before inspecting large amounts of data), conveying this information to contributors, observing contributor performance, understanding what remains ambiguous, revising the instructions, and repeating the process until the resulting output is satisfactory.

To facilitate this process, in Chapter 7, I propose SPROUT, a novel meta-workflow that interleaves tasks performed by contributors and the task designer for specifying tasks and improving their design. SPROUT achieves this by (1) eliciting points of confusion from contributors, (2) enabling task designers to quickly understand these misconceptions and the overall space of questions, and (3) guiding task designers to improve the task design in response. SPROUT lets task designers embed illustrative examples in their instructions, and provides additional assistance by recommending questions related to those examples that it then compiles into *gated instructions* [110], a training and testing workflow that we previously proposed to improve contribution quality by ensuring new contributors understand the instructions. We conducted a user study with requesters with varying amounts of crowdsourcing experience, demonstrating a strong preference for SPROUT over current best practices for creating labeling tasks. Based on our observations and discussion with requesters during the study—and our experiences conducting pilot studies with task designers and contributors while building the tool— we also formulated a set of design recommendations

for future task authoring and debugging tools. This work will appear in [17], and is joint work with Mausam and Daniel S. Weld. The open-source implementation of this work is available at `https://crowdlab.cs.washington.edu/task-design.html`.

Chapter 2
PREVIOUS WORK

In this chapter, I discuss previous and related work that is relevant to self-improving crowdsourcing. I begin with a discussion of applications of artificial intelligence to crowdsourcing. The remaining sections focus on the various aspects of the task creation pipeline: recruiting workers (Chapter 3), managing workers (Chapter 4), and routing and prioritizing tasks (Chapters 5 and 6, respectively). I conclude this chapter with a discussion of general task and workflow design (Chapter 7).

2.1 Artificial Intelligence for Crowdsourcing

The research in this dissertation fits into the broad theme of using AI techniques and decision theory for the optimization of social processes and crowdsourced tasks [162].

2.1.1 Decision Theory for Interface Optimization

Numerous researchers have applied decision theory to control interface behavior. For example, the BUSYBODY system mediates incoming notifications using a decision-theoretic model of the expected cost of an interruption, which is computed in terms of the user's activity history, location, time of day, number of companions, and conversational status [69]. LINEDRIVE illustrates driving directions using optimization to find the optimal balance between readability and fidelity to the original shapes and lengths of road segments [1]. SUPPLE [56] renders interfaces using decision theory to optimize the ease of a user's expected behavior. These systems are a small sample of decision theory applications, which are too numerous to enumerate here.

2.1.2 Decision Theory for Crowdsourcing Control

A popular decision-theoretic framework, which we use for managing workers (Chapter 4), is based on Markov Decision Processes (MDPs) and Partially-Observable Markov Decision Processes (POMDPs). These methods have been previously used for deciding whether to hire another worker or to submit a crowdsourced binary answer [40, 80, 132]. POMDPs are also used for controlling more complex workflows such as iterative improvement [39], switching between multiple workflows for the same task [100], and selecting the right worker pool for a question [136]. Similarly, MDPs are used for optimal pricing for meeting a deadline of task completion [57].

In the literature exploring the aggregation-based, cost-quality tradeoff, POMDP decisions are taken based on the needs of the *question* at hand: "Should one take another vote or instead submit one's best estimate of the answer?" Here, the interactions with a worker are quite passive — following a *pull model*, where once a question is sent to the marketplace, any worker may answer it. In practice, a worker usually has a longer relationship with a requester, since she answers several questions in succession. This sequence of answers can be used for actively allocating test or work questions (*push model*). In Chapter 4, we use POMDPs to decide actively when to test (or train) a worker and whether to replace them, in case they are not performing to expectation.

Self-improving crowdsourcing as described in this dissertation is focused on optimizing crowdsourcing task workflows to produce high quality data, but objectives other than data quality have also been considered. A number of researchers have extended the active learning framework to include noisy crowdsourcing labels when optimizing classifier accuracy [103, 104, 106, 149]. In this line of work, an optimization procedure must reason about not only how worker answers to individual questions may change the agent's belief about the true answers, but also how an altered belief may affect the performance of a machine learning algorithm trained on those beliefs.

Other researchers have designed workflows for many other goals, including debugging machine learning models [5, 6], or training sequential decision algorithms [113, 167].

A large body of work has optimized simple tasks such as classification with noisy workers [40, 80, 139, 149, 161, 163, 164]. Relatively less research has gone into optimizing more complex workflows (such as our work optimizing the CASCADE workflow in Chapter 6) that have a much richer space of possible outcomes. See Section 2.5.5 for more discussion on general workflow optimization.

2.1.3 Modeling Workers over Time

Modeling workers over time is critical for worker management (Chapter 4). Other researchers have also explored the temporal dimension of worker behavior, for example Toomim et al.'s empirical analysis of worker survival curves [157]. Inspired by this work, our model in Chapter 4 uses a parameter to model the probability that a worker is going to leave the system at the next step. Worker retention can be improved by bonuses, making tasks more engaging, or other incentives, but we have not yet tried to optimize these factors. Some recent related work has used worker-centric MDPs for optimally placing an intermediate goal [88] or deciding whether to provide a bonus to encourage worker quality or retention [168].

Other researchers have developed more complex time-series models of worker reliability, e.g., using hidden Markov models [45, 168], or autoregressive models [76]. Since our focus is test question insertion, we use a simpler model for temporal reliability, including a parameter to account for the phenomenon that a worker's accuracy may decrease over time,[1] either due to fatigue, boredom, or deceit. Extensions to more complex models is a topic for future work.

[1]We do not model increase in accuracy in Chapter 4, since most fielded crowdsourcing workflows do not provide continued feedback (after an initial tutorial) that would cause workers to improve over time.

2.2 Recruiting Workers and Contributions

This section discusses recruiting workers and contributions, focusing on online communities where finding contributors is critical to community success.

2.2.1 Starting New Online Communities

Online communities are virtual spaces where people can interact with each other. Many new online communities fail because they are unable to carve out a useful niche, to provide enough value to accrete a community, or because they lose to competition from other communities [93].

There are several ways to help a community reach critical mass. One popular method is to leverage existing members to recruit new members. Previous research has shown that a person is more likely to join a community if he or she has friends that are already members [7]. Companies, such as Dropbox, exploit this principle by providing incentives for users to refer their friends [93].

Bootstrapping the content of online communities is a complementary approach to the cold-start problem and especially useful when resulting content is long lived. Seeded content can increase the utility for initial members to join the community [93]. Therefore, many online communities bootstrap by copying content from 3rd parties. For example, MovieLens imported a database of ratings from another movie rating website (EachMovie.com), which was no longer operational. Resnick et al. [140] show that using paid staff to prepopulate the forum made it more attractive for other people to post to and read the board. Seeded content not only increases the utility of users, encouraging them to join the community, but also can be used to direct the behavior of the new users, encouraging them to contribute similar content [153].

2.2.2 Encouraging Contribution through Task Routing

To make a volunteer online community thrive, the community designer needs to find ways to encourage contributions from its members. One approach, called *intelligent task routing*, models each member's interests, determines a task of potential interest, and sends them a personalized request [34]. As one example, Cosley et al. [35] utilized the edit history of Wikipedia users to model their interests. Their system used information retrieval and collaborative filtering techniques to suggest tasks, significantly increasing the contribution rate. Another approach utilized the rating history of MovieLens users to find those whose ratings differ the most. Their system used this information to send personalized suggestions encouraging users to reply to forum posts of users with opposite opinions; this significantly increased the reply rate [65].

These exciting results show that intelligent task routing can be used to encourage community members to contribute additional content. However, it is not clear whether one can use intelligent task routing to bootstrap content from non-members, who have not generated activity logs that help with targeting. Chapter 3 presents alternative methods for incentivizing contributions from non-members.

2.2.3 Encouraging Contribution Using Request Design

Research in social psychology has shown social influence or persuasive techniques can be used to make people more likely to comply with requests [28]. Since the success of online communities relies heavily on the contributions of community members, many studies have been done to examine how request design can be used to encourage member contributions [93]. For instance, Burke et al. [18] show that asking more specific questions can increase the response rate by 50%. Also, using the social psychology theory of social loafing, Beenen et al. [10] show that requests stressing the uniqueness of the member's contribution significantly increased the contribution rate. Moreover, Lopez and Brusilovsky [112] suggest that the system should adapt the design based on

user demographics. Segal et al. [146, 147] show that well-timed motivational messages can improve engagement in citizen science contexts. All these studies focus on designing requests to get contributions from community members. Users that have not yet committed to the community might be less likely to accept the requests because they are less invested in the community. Therefore, we might need to find another theory in social psychology to motivate request designs that are more suitable for encouraging contributions from non-members.

Research has shown that people are more likely to respond to a large request after they have accepted a smaller request because they want to maintain the consistency of their self-perception [28]. Therefore, one compliance technique that is often used in industry is to hide the large request and present the smaller request to the recipients first, a method called the "foot-in-the-door technique" [53]. Gueguen [62] showed that this method is not only useful in a face-to-face scenario, but can also be used in computer-mediated communication (e.g., email). However, to the best of our knowledge, no researchers have investigated whether an online community could use the foot-in-the-door technique to bootstrap contributions (Chapter 3).

2.3 Managing Workers

This section discusses testing and training workers, two important worker management problems, and Gated Instructons, a workflow that has been shown to effective in screening workers.

2.3.1 Testing Workers

Most practitioners of crowdsourcing use some kind of testing regimen to ensure worker quality. However, the exact policy is usually ad hoc. For example, CrowdFlower has two settings, the number of test questions and total number of questions per page, with the default (best practice) settings as 1 and 5, respectively (i.e., 20% gold). It also allows requesters to specify a minimum acceptable running accuracy for a worker,

whose default value is 80%. All these settings can be changed by a requester, but very little advice is offered on how to set their values [36].

Researchers also routinely insert gold questions in their training data collection. As an example, we survey some papers that used crowdsourcing to generate labeled data for the task of relation extraction from text. Zhang et al. [169] use a policy similar to CrowdFlower's — 20% gold questions and filtering workers under 80% accuracy. Gormley et al. [61] use three test questions every ten questions (30% gold) and filter workers under 85% accuracy. They also boot workers that answer three or more questions in less than three seconds each, and they show that the combination of these methods works better than either alone. Angeli et al. [4] insert two gold questions into a set of fifteen (13% gold) questions and filter the workers who have lower than 67% accuracy. They also filter the specific sets of fifteen questions on which a worker fails both test questions.

All these papers use a common strategy of gold question insertion and worker filtering, but choose very different parameter settings for the same task. Moreover, no paper describes a strategy for computing these parameter settings, which are usually chosen by gut instinct and, in rare cases, by limited A/B testing to assess cost-quality variations empirically for various parameter settings. More importantly, these settings are *static* and do not respond to any change in worker mix, or to individual worker characteristics. In Chapter 4, we develop a POMDP-based formalism for mathematically modeling this problem, present an algorithm for automatically learning model parameters, and hence produce a suitable adaptive policy targeted to a given task and labor market.

Previous work has studied the exploration-exploitation problem of finding the best workers for task routing purposes (Section 2.4.1), but that work predominately assumes that worker quality is fixed, obviating the need (or benefit) of continued testing with gold questions. Exceptions that do model workers over time [45, 76, 168] do not consider gold question insertion.

Finally, we note that methods that induce lower-quality workers to leave of their own volition are complementary to optimizing testing. Mao et al. [116] found that some workers self-police and stop working on tasks for which they are unsure. Carefully designed instruction and reputation systems may cause workers to recognize when they may be providing low-quality work and stop working. Other methods like worker review hierarchies [64, 96] can also serve diagnostic purposes.

2.3.2 Teaching Workers

Researchers have begun to investigate how best to improve crowdsourcing worker quality by actively selecting examples for instruction [9, 151]. Unlike our approach, this work focuses solely on teaching and does not consider the tradeoff between providing more instruction and assigning work that helps the system answer unknown questions.

Optimizing instruction for crowdsourcing shares many of the same challenges faced by intelligent tutoring systems. We make use of knowledge tracing [31], one of the earliest probabilistic models of student learning, but a number of more sophisticated models have also been proposed [89]. Researchers have developed techniques for optimizing instruction in intelligent tutoring systems, including using POMDPs to produce better teaching policies (e.g., [134]). However, our system must consider the long-term implications of learning gains (on future tasks), rather than simply minimizing the time to achieve those learning gains.

Our approach is also inspired by recent work on optimizing the order in which algebraic lessons are taught in educational mathematics games [111, 115, 129]. Like intelligent tutoring systems, these games seek to maximize student knowledge (or as a proxy, the time spent interacting with the game). In our case, worker knowledge is irrelevant in itself, since workers can leave at any time. Instead, we seek to optimize our accuracy over a set of questions that need answering.

2.3.3 Gated Instructions

The importance of instructions, training, and screening was also demonstrated by an analysis of several attempts to crowdsource training data for information extraction [110]. Our approach to semi-automated task design (Chapter 7) adopts *gated instruction* from this work (see Figure 7.2). Gated instructions is a quality control method that uses test questions to ensure that workers have read and understood the instructions. It differs from the common practice of mixing 10–30% gold standard questions into a work stream in the hope of detecting spammers [16, 127], since the former is intended to ensure understanding not diligence. It also has advantages over other approaches like instructional manipulation checks [128], which test attentiveness, not understanding; can be gamed [66]; and do not provide training.

A gated instruction workflow inserts two phases before the main task: an interactive tutorial, followed by a screening phase, which consists of a set of questions workers must pass in order to start work on the actual task (Figure 7.2).

2.4 Task Routing and Prioritization

2.4.1 Routing

Previous work has considered task routing in a number of restricted settings. Some approaches [81, 82, 83] assume worker error is independent of problem difficulty and hence gain no benefit from adaptive allocation of tasks.

Most adaptive approaches, on the other hand, assign problems to one worker at a time, not recognizing the cost of leaving a worker idle [23, 44, 45, 161, 166]. Other adaptive approaches assume that a requester can immediately evaluate the quality of a worker's response [67, 158], or wait for a worker to complete all her tasks before moving on to the next worker [68]. Some, unlike our work in Chapter 5, also assume that any given question can be assigned to just one worker [158]. Moreover, the adaptive approach that best models real workers optimizes annotation accuracy by

simply seeking to discover the best workers and use them exclusively [23]. Other related work seeks to formally characterize the tradeoff between exploring to better estimate an individual worker's accuracy versus the overall distribution of workers, with the goal of identifying a high-performing worker [73].

The Generalized Task Markets (GTM) framework [148] has much in common with the routing problem presented in Chapter 5; they seek to find a coalition of workers whose multi-attribute skills meet the requirements of a job. However, the GTM approach assumes a binary utility model (tasks are completed or not) – it does not reason about answer accuracy.

2.4.2 Prioritization

Chapter 6 presents methods for prioritizing tasks for multi-label classification. Closely related work on optimizing the categorization of items within a taxonomy takes a graph-theoretic approach [131], but does not consider a probabilistic framework for modeling noisy workers, a critical component of crowdsourcing systems. Moreover, that approach assumes labels are organized in a taxonomy that is known a priori, and does not model label co-occurrence, which our experiments in Chapter 6 show dramatically improves labeling efficiency and accuracy. Kamar and Horvitz [79] also consider asking related questions, but assume a hierarchy of tasks is given in advance.

Other related research investigates selecting the next best question from a set of known questions. Often, the goal is to use active learning to improve the accuracy of a classifier, by selecting questions based on label uncertainty or model uncertainty [149, 161]. Our approach to multi-label classification (Chapter 6) seeks to ask questions that optimize the value of information within a graphical model [92], rather than to optimize performance on an external task.

2.5 Task and Workflow Design

In this section, I describe related work on the importance of task and workflow design, tools and approaches to diagnosing and improving designs, fully automatic approaches to design, and alternatives to workflow improvement. I begin by briefly describing related approaches for workflows to create a set of categories (useful for creating taxonomies as in Chapter 6).

2.5.1 Workflow Design for Elucidating Categories

Our approach to label elucidation (for multi-label classification and taxonomization in Chapter 6) is related to work on collaborative and social tagging. [59] uses a Pólya urn model to explain why relative tag proportions tend to stabilize over time for bookmarks on the Delicious website. [25] investigates tagging on Delicious as well, using information-theoretic measures to model the developing vocabulary of tags and the effectiveness of the set of tags for document retrieval. In a crowd labor setting, [101] uses a Chinese Restaurant Process model to optimize free response question answering.

2.5.2 General Design Principles for Tasks and Workflows

There is a small but growing body of work elucidating best practices for task design. CrowdFlower, a major crowdsourcing platform, reinforces that tasks should be divided into discrete steps governed by objective rules; they also highlight the importance of clear instructions [37] and test questions [38]. Several studies of worker attitudes also point to task clarity problems as a major risk factor for workers [54, 119, 165]. Furthermore, large-scale analyses have found positive correlations between task clarity features like the presence of examples and task performance metrics like inter-annotator agreement and fast task completion times [72]. Other controlled empirical studies provide further evidence that examples improve task outcomes [165]. Some

work has sought to systematically understand the relative importance of various task design features [2, 165], but this work is limited to specific task types and general design principles remain poorly understood.

Emerging understanding of good workflow design suggests that investing in worker understanding is critically important to crowdsourcing outcomes. A large-scale controlled study compared the efficacy of different quality control strategies, concluding that training and screening workers effectively is more important than other workflow interventions [123]. Providing feedback about a worker's mistakes has also been shown to be very helpful in improving their answer quality [47].

While there has been more emphasis on understanding worker behaviors, Papoutsaki et al. [130] instead study behaviors of novice requesters designing workflows for a data collection task; they distill several helpful lessons for requesters. As a whole, the above studies demonstrate the strong need for tools like Sprout (Chapter 7) to help requesters clarify the task, include illustrative examples, provide training with feedback, and screen workers.

2.5.3 Tools for Understanding Workflows and Worker Behavior

While this dissertation presents methods to minimize requester effort by helping to quickly improve the instruction (Chapter 7) and automating other parts of the process (Chapters 3– 6), an alternative approach is to develop tools that assist the requester in managing crowd workers.

Several tools support understanding of task behaviors. CrowdScape provides visualizations to help requesters understand individual worker behaviors and outputs [144]. Noting that experimenting on different versions of task instructions, rewards, and flows is time-intensive, CrowdWeaver provides a graphical tool to help manage the process and track progress [87]. Cheng et al. [24] propose methods for automatically determining task difficulty. Kairam and Heer [78] provide methods for

clustering *workers* (rather than questions, as in the task-improvement methods of Chapter 7).

In the MOOC domain, Glassman et al. [58] provide a user interface that enables a teacher to provide feedback for common errors on programming questions at scale. Similar approaches could be used for scaling worker feedback provided by the requester (or skilled workers).

2.5.4 Tools for Task and Workflow Design

The SPROUT task improvement tool (Chapter 7) extends a line of research on tools that support designing and debugging workflows. We are inspired by Turkomatic [95], which proposed having workers themselves decompose and define a workflow to solve a high-level task description provided by a requester. Both systems embody a meta-workflow with crowd workers acting in parallel with the requester. While Turkomatic was only "partially successful" [95], the vision is impressive, and we see SPROUT as diving deeper into the task specification aspect of the greater workflow design challenge. SPROUT also leverages the reusability of instructions across many instances of a task, while Turkomatic considered one-off tasks where reusability is limited. Fantasktic [63] was another system designed to help novice requesters be more systematic in their task instruction creation via a wizard interface, but did not incorporate worker feedback or aid requesters in identifying edge cases like SPROUT. Developed in parallel with our work, WingIt [114] also has workers make instruction edits to handle ambiguous instructions, but does not provide a requester interface and relies on the requester approving or modifying each individual edit (which could be very time-consuming).

Forward-thinking marketplaces, such as CrowdFlower and Daemo [55], already encourage requesters to deploy prototype tasks and incorporate feedback from expert workers before launching their main task. These mechanisms demonstrate the feasibility and value of worker feedback for improving tasks. SPROUT makes this

paradigm even more useful with a meta-workflow that produces structured task feedback, does not require expert workers, and enables requesters to efficiently resolve classes of ambiguities via a novel user interface.

2.5.5 Fully Automatic Workflow Improvement

Researchers have attempted to remove the requester from workflow optimization with varying degrees of success. Turkomatic [95] attempted to have the crowd itself dynamically create and execute workflows via task decomposition. However, it was unsuccessful at having the crowd perform quality control; their solution requires the requester to continually monitor the crowd and intervene where necessary. On the other end of the spectrum, de Boer and Bernstein [41] propose using Bayesian optimization to recombine a library of workflow primitives to find the best-performing workflow. Lin et al. [102, 105] describe a vision for a system that enables a requester without expertise in AI to specify a workflow and have it benefit from AI optimization [40]. These approaches show promise, but general workflow optimization is very difficult and remains unsolved.

Notably, none of these approaches seeks to improve the quality of the workflow primitives themselves, which are assumed fixed. As discussed above, optimizing these primitives is a very important part of general workflow improvement. Moreover, the requester cannot be fully removed from this process, since they must resolve ambiguities in the initial task description. Chapter 7 presents methods for optimizing workflow primitives.

2.5.6 Alternatives to Workflow Improvement

Chang et al. [22] argue that striving for unambiguous task guidelines may be counterproductive, because of the inherent ambiguity of most annotation tasks. An alternative approach is to cluster the dataset so that ambiguities can be reviewed and

resolved in various ways. Kulesza et al. [94] propose structured labeling, which outputs a clustered dataset with each cluster labeled by a single person akin to the requester. Chang et al. [22] propose Revolt, a system that outputs structured labels created by the crowd, so that the requester does not need to perform this clustering and can instead resolve ambiguities post-hoc based on the clusters. It is unknown how useful these clusters actually are for requesters; Revolt did not provide a requester tool and evaluated with simulated requesters.

Structured labels may add reusable benefits beyond the requester's immediate needs,[2] but performing task improvement is more scalable for data collection. If a large amount of data must be annotated, Revolt will ask workers to explain and categorize *every* unresolved item, requiring a very large budget. Our approach to task improvement (Chapter 7) instead has workers cluster a *sample* of the data and aids the requester in improving the task. This method enables requesters to examine a larger and more diverse sample of the data than previously possible when designing crowdsourcing tasks.

Once the requester has improved the task, the task can be re-run many times to collect high-quality data at lower cost. Indeed, entire companies are based on perfecting workflows: Mighty AI (labeling for self-driving cars), LeadGenius (lead generation), B12 (website creation), etc. Chapter 7 presents the first requester tool for semi-automated task improvement, and validates the approach in a user study with requesters.

[2]E.g., by allowing machine learning practitioners to quickly relabel a dataset while performing feature engineering experiments [94].

Chapter 3

REQUESTING WORK FROM OUTSIDE CONTRIBUTORS USING DATA MINING

In order to realize the full potential of self-improving crowdsourcing, recruiting workers—the first step in the task pipeline (Figure 1.2)—must be efficient and require minimal requester effort. Crowdsourcing marketplaces provide one method of recruiting workers. In these marketplaces, requesters post an open call for work, and workers search for tasks they want to complete based on keywords, etc. However, this recruiting model does not work for all crowdsourcing tasks and applications. For instance, some tasks may require workers with specialized expertise or experience not available in the worker pool. In these cases, it may be beneficial to solicit workers outside of the marketplace to join the task. Workers outside the marketplace may also have other desirable characteristics, such as being more likely to join and become active contributors to specialized *communities* (e.g., FoldIt or Wikipedia).

In this chapter, I present methods for identifying outside contributors through digital footprints indicating they may have useful knowledge or be more likely to contribute. These methods are useful for targeted recruiting that helps to reduce cost and unwanted solicitation. In order to deal with a combinatorial action space, we formulate targeted recruiting as a submodular optimization problem. Further, we show the benefit of experimenting with features (to improve the self-improving crowdsourcing agent's model of the environment) and request designs (to improve the agent's action space); future crowdsourcing system designers should consider supporting requesters or meta-workers in these activities.

3.1 Introduction

The Internet has spawned communities that create extraordinary resources. For example, English Wikipedia's 34 million users have created over five million articles, a resource with over 60 times as many words as the next largest English encyclopedia.[1] Similarly, StackOverflow has become a top resource for programmers with over 25 million answers to 16 million questions,[2] while Yelp users generated more than 155 million reviews.[3]

In reality, however, most online communities fail. For example, thousands of open source projects have been created on SourceForge, but only 10% have three or more members [93]. Furthermore, more than 50% of email-based groups received no messages during a four-month study period [20]. Since network effects sustain successful communities, the key challenge for designers is kindling initial community activity such that it leads to a tipping point [49, Section 17.3].

Previous research has focused on methods for encouraging existing community members to contribute additional content. For example, SuggestBot used the edit history of Wikipedia editors to recommend articles for them to edit [35]. Beenen et al. conducted an experiment on MovieLens [33] and showed that designing requests based on social psychology theories can better motivate users to contribute [10]. Burke et al. [19] show that the community can encourage contributions from newcomers by showing the contributions of their friends. While these results provide insights on how to cause existing community members to increase their activity, they do not address the community "cold-start" problem. Without enough user-contributed content to attract a critical mass, there might never be enough value to recruit initial members to join the community [93].

---

[1] `https://web.archive.org/web/20180803222637/https://en.wikipedia.org/wiki/Wikipedia:Statistics`

[2] `https://web.archive.org/web/20180815183920/https://stackexchange.com/sites`

[3] `https://web.archive.org/web/20180805222543/https://www.yelp.com/factsheet`

This chapter examines methods for solving the cold start problem by bootstrapping community content from the contributions of non-members. Several challenges make this a difficult problem. First, since the community doesn't have an activity log for non-members, it is hard to model their interests and recommend tasks accordingly. Second, non-members have no existing commitment to the community, so they might not be inclined to make a contribution; it is unclear which social psychology theory one could use to encourage contributions in this case. Finally, there are a huge number of possible non-members and many candidate tasks to suggest; determining which requests should be sent to which users represents a combinatorial optimization problem.

Specifically, we identify a class of communities, which we call *datamining bootstrappable*, where an external resource provides a means of identifying potential members and estimating their interests and expertise. For these communities, we define the bootstrapping process as a decision-theoretic optimization problem. Previous research has shown decision-theoretic optimization is useful in similar social computing contexts such as crowdsourcing [39]. Applying the decision-theoretic framework to model the bootstrapping problem allows us to estimate the utility of different operations and find a set of operations that are near optimal for the community.

In addition, we conducted a field experiment on *Open AI Resources* (Open AIR)[4], a website which launched in July 2014, three months before our study. In collaboration with the Allen Institute of Artificial Intelligence (AI2)[5], we were allowed to access the user data of the site. This provided us a unique opportunity to study the community bootstrapping problem because Open AIR hadn't accumulated much reputation or user-generated content when we conducted our study.

---

[4] *Open AI Resources*, `http://airesources.org/`, is an online community that allows users to comment and discuss Artificial Intelligence (AI) related open-source datasets and software.

[5] `https://allenai.org/`

By text-mining information from the Google Scholar citation graph and linking to author homepages, we identified individuals who might be willing to join the Open AIR community. We considered a range of strategies to get these individuals involved and measured their response rates. The results from these experiments inform the parameters of a decision-theoretic model that can control the community bootstrapping operation. Our study is an initial step toward building an automatic system that can bootstrap the contents of online communities. In summary, this chapter makes the following contributions:

1. We characterize a class of online communities, which we call "datamining bootstrappable communities," where an external resource provides a means of identifying potential members and estimating their interests and expertise. We then define the problem of efficiently bootstrapping such a community in terms of decision-theoretic optimization, and propose a greedy algorithm that efficiently solves this problem with performance guarantees.

2. Using the *Open AI Resources* community as a case study, we identify a set of informative, text-minable features and estimate the probabilities that parameterize the actions in our model.

3. We demonstrate that request design, such as the *foot-in-the-door technique* [53], and the estimated level of interest of non-members are important features in the model that can significantly affect the probability of contribution.

4. We ran an experiment using synthetic data generated with parameters learned from the real data we collected to show that our decision-theoretic optimization algorithm can achieve comparable utility for bootstrapping online communities while issuing only 55% as many requests, compared to the strongest baseline.

3.2 Decision-Theoretic Community Bootstrapping

3.2.1 Applicable Communities

We start by specifying a class of online communities where our bootstrapping methods are appropriate and formally define the problem of eliciting contributions. Since the very notion of community is amorphous, we assume that there is a set of humans $\mathcal{H}$ who are potentially willing to make contributions $\mathcal{C}$ to different tasks $\mathcal{T}$. For example, for Yelp, $\mathcal{T}$ might be the set of restaurants, and $\mathcal{C}$ could be a contributed review. For AirBnB, there might be a single task (list a house) and each contribution would correspond to a rental property. We note that many communities have complex (two-sided) market dynamics [141], which we are ignoring; however, from a practical point of view, a single side tends to dominate most markets. For example, AirBnB focused on sellers, since their contributions (available rental inventory) were durable (led to repeated transactions); renters came easily. Similarly, Wikipedia was bootstrapped with an early focus on authors, even though the community would fail without readers as well.

We say a community is potentially *datamining-bootstrappable* if there are mechanisms, likely using external websites or similar resources, for satisfying the following conditions:

1. Identifying the humans who are potentially interested in a given task, and estimating the probability that they will contribute.
2. Sending a request to a those humans (e.g., via a data-mined email address or other communication channel).
3. Estimating the quality of their contribution.

If these conditions hold, our method is applicable. However, we caution that these conditions don't guarantee that our method will bring the community to the tipping

point. This depends on the specific parameters, e.g., number of humans, response rate, and actual utility of contributions, including their durability.

3.2.2 Request Model

Since we are targeting non-members who are not committed to the community, we assume that the community can send, at most, one contribution request to each human. In addition, since the request design greatly affects the response rate, a system can explore different requests in the design space $\mathcal{D}$. Thus the space of possible requests is $\mathcal{R} = \mathcal{H} \times \mathcal{T} \times \mathcal{D}$. Should a request result in a contribution, the quality of that contribution will come from a set of possible qualities $\mathcal{Q}$. Our objective is to issue the set of requests $\mathbf{R} \subseteq \mathcal{R}$ with maximal expected utility, while satisfying the constraint that no human is asked to do more than one task. Letting $\mathbf{R_h} \subseteq \mathbf{R}$ denote the subset of requests given to human h, this constraint requires $\forall h, |\mathbf{R_h}| \leq 1$.

3.2.3 Probability and Quality of Contributions

The expected utility of a set of requests is defined in terms of the probability of the humans responding to the appeal and the utility of the resulting contributions. As discussed in Sections 2.2.2 and 2.2.3, previous work has shown that the probability of a request being honored is a function of two key factors: the human's preexisting interest in task t, which we denote $i_{h,t}$, and the request design d [10, 34, 35]. Therefore, we model the probability that human h will contribute to task t as $P(c_{h,t}|\ d,\ i_{h,t})$. Similarly, we condition the quality of a contribution on the human and their interest in the task: $P(q_{h,t}|\ h, i_{h,t})$.

In order to apply our model to a specific domain, one needs to specify a set of designs and a way of estimating interest levels and then measure the conditional probabilities. In our experiments (Section 3.4), we show how this may be done for our case-study domain, the Open AIR website. For example, our experiments show that

if an author has written a paper citing a resource, then this connotes a significantly higher level of interest and increased contribution rate (e.g., see Table 3.3).

3.2.4 The Utility of Contributions

In general, it is extremely difficult to estimate the quality of a given contribution [70]. As a result it is common to assume (*all other things being equal*) that more contributions are generally better than fewer. For example, Amazon emails all purchasers to solicit a review. This intuition can be formalized as monotonicity: Let A, B and N be sets of contributions. A utility function, $U : 2^N \rightarrow R$, is *monotone* if for every $A \subseteq B \subseteq N, U(A) \leq U(B)$.

Furthermore, some contributions are more valuable than others. For example, the first review of an open-source code library is probably more useful than the 100th. In general, the marginal value of a contribution to a task is smaller when the task has already received other contributions. This "diminishing returns" property is captured by the notion of submodularity. More precisely, a utility function is *submodular* if for every $A \subseteq B \subseteq N$ and $e \in N : f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$ [125].

For example, one possible monotone submodular utility function can be defined in terms of the utility of the contributions to task t as $\mathrm{U}_t(\mathbf{c_t}) = \alpha \log \left[\beta \sum_{c_{h,t} \in \mathbf{c_t}} f_q(c_{h,t})\right]$, where α and β are constants, and f_q is a measure of the quality (e.g., the length) of contribution $c_{h,t}$ made to task t. But this is just an example. For the rest of the chapter, we assume that the utility provided by a set of contributions to task t is a monotone submodular function U_t. For simplicity, we further assume that the system's overall utility is a linear sum of the utilities achieved on each task. This approximation is common practice [143, Section 16.4.2] and makes intuitive sense. For example, the utility Yelp receives for the reviews of restaurant A are roughly additive with those of restaurant B.[6] We seek to send out the following set of requests, which

[6]One might argue that diminishing returns might apply *across tasks* as well as within tasks; we hope to consider this and other elaborations in future work.

maximizes the total expected utility $\mathrm{E}[\mathrm{U}] = \mathrm{E}[\sum_{t\in\mathcal{T}} \mathrm{U}_t]$:

$$\mathbf{R} = \underset{\mathbf{R}\in 2^{\mathcal{R}}}{\operatorname{argmax}} \sum_{t\in\mathcal{T}} \sum_{\mathbf{Q_t}\in 2^{\mathbf{R_t}}} P(\phi_c(\mathbf{Q_t}, \mathbf{R_t})) \sum_{\mathbf{q_t}\in|\mathcal{Q}|^{|\mathbf{Q_t}|}} P(\phi_q(\mathbf{q_t}))\mathrm{U}_t(<\mathbf{Q_t}, \mathbf{q_t}>),$$

where $\mathcal{R}$ is the set of all possible requests, $\mathbf{R_t} \subseteq \mathbf{R}$ is the set of requests for task t, $\phi_c(\mathbf{Q_t}, \mathbf{R_t})$ denotes the event where $\mathbf{R_t}$ requests are made but only $\mathbf{Q_t}$ actually result in contributions, and $\phi_q(\mathbf{q_t})$ denotes the event where each of the requests in $\mathbf{Q_t}$ results in a contribution with a quality from $\mathcal{Q}$, the set of possible contribution qualities. The probability of these events are defined as

$$P(\phi_c(\mathbf{Q_t}, \mathbf{R_t})) = \prod_{<h,t,d>\in\mathbf{Q_t}} P(c_{h,t}|\ d,\ i_{h,t}) \prod_{<h,t,d>\in(\mathbf{R_t}-\mathbf{Q_t})} 1 - P(c_{h,t}|\ d,\ i_{h,t})$$

and

$$P(\phi_q(\mathbf{q_t})) = \prod_{<h,t,d>\in\mathbf{Q_t}} P(q_{h,t} \mid h, i_{h,t}).$$

Recall also that $\mathbf{R}$ must satisfy the constraint that $\forall h, |\mathbf{R}_h| \leq 1$.

3.2.5 Solving the Optimization Problem

Although there are various utility functions community designers can choose from, in a realistic setting, one needs to consider that the utility of each contribution depends on the contributions from other people. For instance, the utility of an additional contribution to a task might decrease as the number of contributions to that task increases. Therefore, to find the solution to this optimization problem, the system needs to enumerate all the possible allocations of the requests. However, this creates $|\mathcal{T}|^{|\mathcal{H}|}$ possible allocations. If the number of humans or tasks is reasonably large (e.g., in the hundreds), then the search space will be intractable. We must therefore consider approximate solutions.

While many methods for heuristic search have been proposed, few offer performance guarantees. However, the submodular nature of the utility function allows us to closely approximate the optimal value with the following method. Algorithm 1

Algorithm 1 A decision-theoretic approximation algorithm for issuing bootstrapping requests.

Input: $\mathcal{H}$, $\mathcal{T}$, $P(c_{h,t}|\ d,\ i_{h,t})$, $P(q_{h,t}|\ h,\ i_{h,t})$ $\forall < h, t, d > \in \mathcal{R}$

Output: $\mathbf{R}$

Compute the initial expected utilities for all possible requests r_{h,t,d^*} $\forall h \in \mathcal{H}$ $\forall t \in \mathcal{T}$, where $d^* = \text{argmax}_d\, P(c_{h,t}|d, i_{h,t})$

Initialize $\mathbf{R} \leftarrow \emptyset$

while There is a human $h \in \mathcal{H}$ with no assigned task **do**

Find the request r_{h^*,t^*,d^*} with highest expected utility

Add r_{h^*,t^*,d^*} to $\mathbf{R}$

Mark h^* as assigned

Recalculate the expected utility for the requests r_{h,t^*,d^*} for every unassigned human $h \in \mathcal{H}$

end while

first computes the expected utilities for all possible requests. Then, the system sends the request with the highest expected utility. Once the system assigns one human to a task, it adjusts the expected utilities for the requests of that task and all the unassigned humans based on the expected contributions of the task. By iterating this process until all humans in $\mathcal{H}$ are assigned, the system can make sure the community has requests with the highest expected utility at each point when a partial assignment is made.

3.2.6 Performance Guarantee

As we now show, our assumption that U_t is monotone submodular guarantees that our solutions will be good. Since each outcome is associated with a nonnegative probability and monotone submodular functions are closed under nonnegative linear combination, the expected utility of a set of requests $\mathbf{R}$ (Equation 3.1) is also mono-

tone submodular.[7] This enables us to provide the following optimality guarantee for our algorithm:

Theorem 1. *Given the constraint that at most one request may be sent to any potential contributor, Algorithm 1 obtains at least* $\frac{1}{2}$ *of the total achievable expected utility.*

Proof. The constraint that each human receive at most one task is naturally encoded as a matroid constraint on the set of total possible requests. Fisher et al. [52] show that subject to a matroid constraint, Algorithm 1 achieves at least this fraction of the optimal value for monotone submodular functions. □

Note that we have assumed that utility functions are defined on a per-task basis, so Algorithm 1 needs only recompute at most $|\mathcal{H}|$ values in each step, leading to a worst-case $O(|\mathcal{H}|^2)$ performance.[8]

3.3 Applying the Model to Open AIR

So far, our request-assignment model has been abstract and hence applicable to many different communities. To make it concrete, we consider Open AIR (Figure 3.1), an online community hosted by the Allen Institute of Artificial Intelligence (AI2) which allows people to research and review open source AI resources (e.g., datasets and software). To initialize the content of Open AIR, an administrator manually searched for resources on the Web and added their information to the website. Open AIR provides three ways for a user to contribute to the community: first, by submitting

[7]Moreover, this function is *adaptive monotone submodular* [60], since we assume contribution probabilities are independent. While this means that an adaptive version of Algorithm 1 is also near-optimal, adaptive requests are impractical due to delays between the request and contribution; we do not consider this setting further in this chapter.

[8]Computing utilities involves taking an expectation over the possible outcomes for the requests assigned to a task. We assume that the probability and value of human contributions come from a bounded number of classes (Table 3.3), which enables speedy utility computations.

a new resource; second, by updating an existing resource; third, by commenting on a resource.

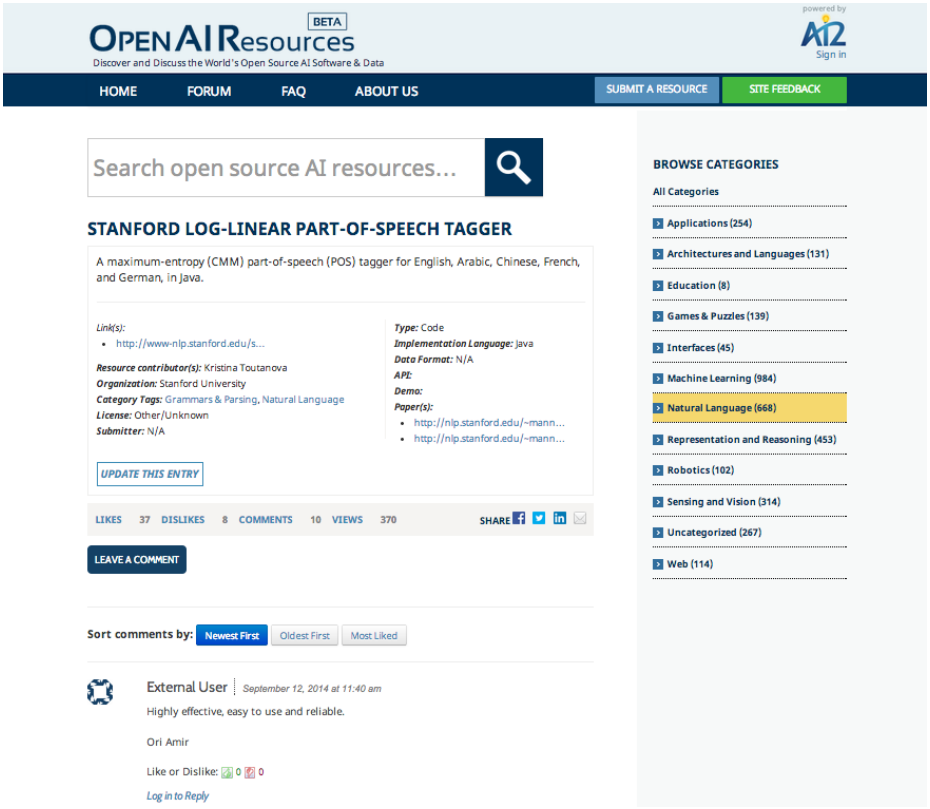


Figure 3.1: An example Open AI Resources (Open AIR) interface of an AI resource. The interface presents the basic information of the resource (e.g., a summary of the resource and its main contributor) and the comments of previous users. The users can update the entry or leave a comment by clicking the buttons on the interface.

We single out *commenting on a resource* for three reasons. First, the initialization process meant that Open AIR already had many resources in its DB. Second, commenting on a resource requires less effort than other kinds of contributions and should be easier to encourage. Third (and most important), reviews and opinions (comments) provide useful information for users who want to use the resources in

the future, which creates unique value for the community. Bootstrapping Open AIR, therefore, means encouraging enough non-members to come and review resources that the site reaches a tipping point and becomes self-sustaining.

In order to apply the decision-theoretic model we need a set of possible contributors, $\mathcal{H}$, whom we consider to be authors with a Google Scholar page. We define one task, $t \in \mathcal{T}$, for each resource. A contribution request also has a design $d \in \mathcal{D}$, which is an email template (described below) that we use when asking the non-member to contribute a review. The final requirement for the decision-theoretic model is a set of parameter values — specifically, the probability of contribution, $P(c_{h,t}|\ d,\ i_{h,t})$, for different designs, d, and different interest levels, $i_{h,t}$. These probabilities are estimated in our Experiments section. But before we can measure these probabilities, we need to define the features that we'll use for conditioning. The next subsection discusses the set of values we consider for $i_{h,t}$ and how text mining can extract these features from public information on the Web. The following subsection (Section 3.3.1) describes our request designs, d.

3.3.1 Text Mining Features to Predict Contributions

As we have mentioned, the interests of non-members are difficult to model because the system doesn't have logs of their activity in the community. Therefore, to estimate the interest level $i_{h,t}$ of non-member h responding to a task t about a resource, we propose that a system can use text mining of publicly available information on the Web. For a research-oriented community like Open AIR, non-members of particular interest (researchers) leave information traces in the form of publications. When authors cite a resource, they indicate basic knowledge or understanding of the resource and may be more likely to write comments for that resource.

Moreover, the text surrounding a citation may contain valuable information about the citation, its role in the paper, and the author's interest in the corresponding resource. To analyze these citation contexts, we manually examined 100 of them to see

TEXT-USE	**Citation Context**
True	We used the Stanford Named Entity Recognizer [4] in order to extract names of places, organizations and people names from the target.
True	We apply Stanford NER toolkit to extract named entities from the texts (Finkel et al., 2005).
False	In contrast, NER systems only categorize named entities to several predefined classes (typically 'organization', 'person', 'location', 'miscellaneous' [13]).
False	However, current NER systems such as Stanford NER that achieve F1 scores of 0.87 on news articles [21], achieve a significantly lower F1 score of 0.39 on tweets with a precision as low as 0.35.

Table 3.1: Examples of the two types of citation context that determine the **TEXT-USE** feature.

if there were any patterns. Preliminary analysis revealed two types of citation contexts: 1) the authors indicate using the software or dataset to conduct an experiment; or 2) the authors do not indicate using the resource, but list the paper to recognize previous or related work. For examples of the two types of citation contexts, see Table 3.1.

An orthogonal dimension of the citation context is the sentiment of the text. If an author expresses strong sentiment, either positive or negative, he or she may be more likely to respond to a request about the cited resource [43]. We enumerate the text

mining features that characterize the interaction $i_{h,t}$ of non-member h with a task t as follows:

- **TEXT-CITE**: *True* if a human, h, has cited the resource and *False* otherwise.
- **TEXT-USE**: *True* if h has cited the resource and indicated use and *False* if h has cited the resource without indicating use.
- **TEXT-SENT**: *Positive*, *Neutral*, or *Negative* based on the sentiment of h's citation of the resource.

TEXT-USE and **TEXT-SENT** are only defined when **TEXT-CITE** is True.

For each resource, we determine these features for non-members as follows. First, we extract the title of the paper that describes the resource. For instance, the Open AIR resource "Stanford log-linear part-of-speech tagger" has a link to the associated paper "Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger." Using the title of this paper, we search Google Scholar to retrieve the papers that cite the resource paper using the "Cited by X" link.[9] Finally, we parse the authors' email addresses in the citing papers to obtain a list of emails of non-members who have written papers that cite the resource paper. **TEXT-CITE** is *True* for a non-member / resource pair when the non-member's email appears in this list.

To extract the citation contexts, we built a parser to parse the text around citations in two common reference styles. The first style is the (*Author year*) format used in this paper. To find these citations, we used a regular expression to construct the (*Author year*) pattern with the author information and published year of the resource paper. The second reference style is the [*number*] format, where the *number* is the index of the citation in the paper's References section. To find these citations, we searched for the title of the resource paper in the References section to find its index,

[9]Our experiments adhere to the Google Scholar ToS, as the first author manually executed the queries.

and searched for the pattern [*number*] using that index. In each case, the citation context is the sentence containing the citation.

We analyze these citation contexts to determine the values of **TEXT-USE** and **TEXT-SENT**. To determine the value of **TEXT-USE**, we expanded the verb *use* with WordNet [121] to obtain a synset, or collection of synonym words.[10] **TEXT-USE** is *True* when any of the stemmed words in the citation context matches a word from the synset. To determine the value of **TEXT-SENT**, we perform sentiment analysis of the citation context using SentiWordNet [50]. SentiWordNet is a lexical resource that maps each synset in WordNet to scores that quantify the amount of positive and negative sentiment. To calculate a single average score for the sentiment of the citation context, we sum the positive scores and subtract the negative scores of each word, then divide by the total number of words. **TEXT-SENT** is *Negative* for average scores lower than -0.007 (1st quartile), *Positive* for scores higher than 0.018 (3rd quartile), and *Neutral* for scores between -0.007 and 0.018.

3.3.2 Designing Contribution Requests

Another parameter that affects the probability of contribution in our model is the request design d. In our case, this refers to the type of email message that was sent to non-members asking for a comment. When crafting these pleas, we expected that the response rate would be inversely proportional to the effort needed for the person to make the contribution, so we tried to make contributions as simple as possible.

Our initial design allowed non-members to comment on a resource by simply hitting "reply" to the request email; the body of their reply was automatically added to the Open AIR website and attributed to the person sending the email. Although this allowed people to enter reviews quickly and without leaving their email client, it was a failure. Not a single one of the 69 recipients submitted a review. Informal

[10]The words in the synset are *use*, *utilize*, *apply*, and *employ*.

interviews suggested that the problem was likely a lack of context — the non-members had neither a sense of the type of review that was expected nor how it would appear on the website. Based on this feedback, we switched to a different approach.

Our next designs brought the non-member directly to the site. A baseline design explained Open AIR and how their contribution would benefit the community, then provided a hyper-link labeled "Tell us about your experience using this resource." If the person clicked on the link the landing page presented two text boxes for 1) their name and 2) their comments on the resource.

Our final design utilized a method known as the "foot-in-the-door technique" [53], which showed that if one first asks a person to complete an easy task, they are more likely to later do a more time consuming task. With this method the candidate received a message identical to the baseline, but instead of a link inviting "Tell us about your experience using this resource." we presented a simple question which asked whether the recipient would like to recommend the resource to other AI researchers, and then presented two links, one for "Yes" and one for "No." Once the non-member clicked one of these links an Open AIR page would open in a web browser displaying an interface that invited them to optionally elaborate their opinion with more detailed information. In summary, we experimented with two designs, corresponding to the following feature:

- **REQ-FOOT**: *True* if a design, d, uses the *foot-in-the-door* techniques and *False* otherwise.

Although the comments induced in the **REQ-FOOT** = *True* condition require the same amount of effort as the baseline (clicking a link, entering a comment, and clicking submit), we conjectured that the the foot-in-the-door design would yield a greater number of reviews. The ability to contribute a useful bit of information with a single click might induce non-members to invest in the community, and once engaged they would more likely contribute a review.

Resource Name	**Associated Paper**
Scikit-learn	Scikit-learn: Machine learning in Python
WEKA	The WEKA Data Mining Software: an Update
Pascal VOC Dataset	The Pascal Visual Object Classes (VOC) Challenge
Stanford Named Entity Recognizer	Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling
Caltech 101	Learning Generative Visual Models from Few Training Examples

Table 3.2: The five resources that we asked the non-members to comment on in our study.

3.4 Experiments

To estimate the probability of contribution for different conditions, we conducted a set of controlled experiments with Open AIR. In our experiments, we focused on five different Open AIR resources (Table 3.2) and emailed contribution requests to 1,339 non-members who cited at least one of the resources. The emails were sent out on weekday mornings between 10/28/2014 and 11/10/2014. For reference, the email template for the request which applies the foot-in-the-door technique can be found in Appendix A. The email campaign was managed using MailChimp,[11] which allowed us to record whether the recipients opened the email and clicked the links in the email. If the non-members accepted the request and commented on the resources using the web interface, our program would record the information of the comments and automatically post the comments on Open AIR. To determine the quality of the

[11] `https://mailchimp.com/`

comments generated by these non-members, we manually examined all the comments resulting from the requests of our experiments. The average length of the comments is 26.4 words. The comments also provide different perspectives for the users to better understand the resources. For example, one comment mentioned that Scikit-learn is not only a useful resource, one can also learn about the algorithms from their website and documentation: *Scikit provides a wide variety of Machine Learning and data-processing algorithms, all interfaced through Python. Plus, their website is a great resource for concepts and details about the algorithms.* Also, another comment about PASCAL VOC dataset helps the users to understand why this dataset is so important to computer vision research: *Currently, it is the best computer vision dataset for evaluating object detection algorithms. It has had a long history and has been instrumental in greatly improving the state-of-the-art of object detection.* The results suggest that the comments generated by these non-members can be useful for other members in the community.

3.4.1 The Effects of Citing a Resource

In the previous section, we made the case for the following hypothesis.

H1a: *Non-members who wrote papers that cited a resource are more likely to accept a request to comment on the resource.*

To see if we can use the information of who cited the resource (obtained by text mining the Web) to bootstrap contributions from non-members, we conducted an experiment that sent requests to non-members who cited one of three Open AIR resources: Weka, Scikit-learn, and the Pascal VOC Dataset. The emails were sent under two conditions:

1. **Intelligent**: The system sent contribution requests only to non-members who have cited one of the three resources (**TEXT-CITE** = *True* for all h in this condition).

2. **Random**: The system randomly selected one of the three resources and sent requests asking the non-members to comment on that resource. We controlled this condition to ensure that the probability a recipient has cited the resource was at least 1/3 by including the non-members that we knew cited the resource ($P($**TEXT-CITE** = *True*$) \geq 1/3$ in this condition).

403 request emails were sent under the Intelligent condition and 382 request emails were sent under the Random condition. The results show that the requests sent under the Intelligent condition had significantly higher email-open rates (49.1% v.s. 40.8%, $\chi^2 = 5.45$, $p = 0.02$, $n = 885$, $df = 1$, effect size= 0.079), link-click rates (10.7% v.s. 4.7%, $\chi^2 = 9.71$, $p < 0.01$, $n = 885$, $df = 1$, effect size = 0.105), and comment rates (5.0% v.s. 1.3%, $\chi^2 = 8.49$, $p < 0.01$, $n = 885$, $df = 1$, effect size = 0.098) (Figure 3.2). One should note that the Random condition tested in this study is actually quite a strong baseline with 1/3 chance that the non-members cited the resource. Moreover, subsequent analysis showed that all of the non-members who ended up writing comments under the Random condition in fact *had* cited the corresponding resource in some paper. This result provides strong support for **H1a** and shows that authors that cited a resource (**TEXT-CITE** = *True*) were significantly more likely to open email requests, follow links, and contribute by writing comments about the resource. It also confirms our hypothesis that obtaining features by text mining the web can be used to help a community bootstrap content from the contributions of non-members.

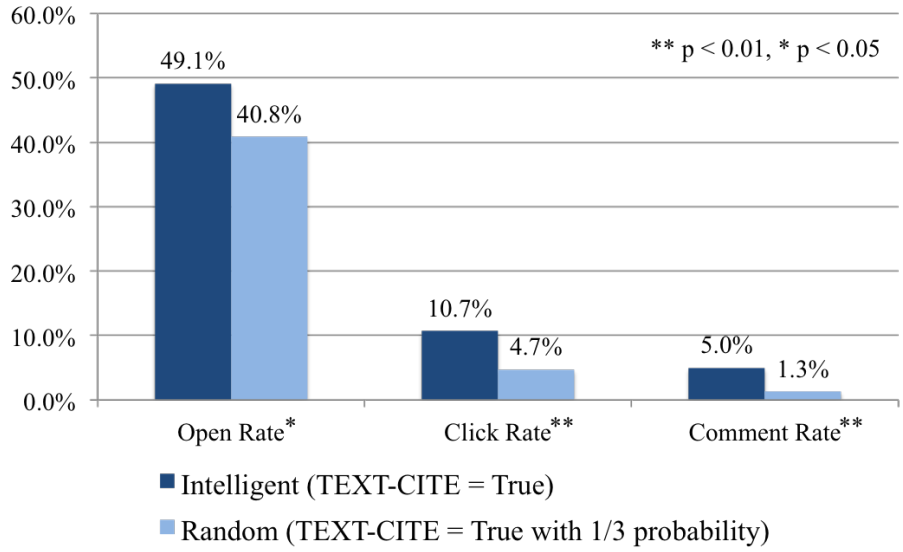


Figure 3.2: Comparison between the requests that perfectly match the non-member and the resource they cited (Intelligent), and the requests randomly assign one of the three resources to the non-member (Random). The results show that the requests that perfectly match the non-members and the resources they cited had a significantly higher open rate, click rate, and comment rate.

3.4.2 The Effects of the Context of a Citation

Since the context of a citation provides information about an author's relationship to a cited resource, we are interested in whether we can use text mining to determine which authors are more likely to accept contribution requests. Our first hypothesis is that the **TEXT-USE** feature will help us predict requests that will result in contributions.

H1b: *Non-members who indicate in the citation context having used a resource are more likely to comment on the resource.*

In this experiment, we focused on 340 non-members who cited one of the five resources listed in Table 3.2. We extracted the context of their citation, using the

method described in the previous section. We separated the non-members into two conditions, based on whether the citation context showed that they had used the resource to accomplish a task (**TEXT-USE** = *True*, $n = 153$) or merely compared to it (**TEXT-USE** = *False*, $n = 187$).

The results show that the email-open rates were similar for the two conditions (47.7% v.s. 50.3%, $\chi^2 = 0.22$, $p = 0.64$, $n = 340$, $df = 1$, effect size = 0.025). In terms of click rates, we can see that the click rate for the group who indicated using the resource is about 50% higher, but the difference is not statistically significant (11.1% v.s. 7.5%, $\chi^2 = 1.33$, $p = 0.25$, $n = 340$, $df = 1$, effect size = 0.063). The biggest difference between the two groups occurred in the comment rate. Non-members whose context indicated resource usage were three times more likely to provide a written review than the control group (5.9% v.s. 1.6%, $\chi^2 = 4.52$, $p = 0.03$, $n = 340$, $df = 1$, effect size = 0.116) (Figure 3.3). This supports **H1b** and shows that authors who indicate having used the resource in the citation context (**TEXT-USE** = *True*) have a significantly higher contribution rate.

Previous research has shown that people are more likely to leave highly positive or negative reviews because high valence experiences often motivate interpersonal communication [43]. Therefore, we hypothesized that this might also apply with respect to request acceptance.

H1c: *Non-members who expressed strong positive or negative opinions when describing the citation are more likely to comment on the resource.*

To test this hypothesis in the case of Open AIR, we analyzed the same 340 emails from the previous experiment, partitioning them into 3 groups: **TEXT-SENT** = *Negative* ($n = 83$), **TEXT-SENT** = *Neutral* ($n = 175$), and **TEXT-SENT** = *Positive* ($n = 82$), based on their average sentiment score.

The results suggest that there were no significant differences between the sentiment score groups in terms of email-open rate (44.6% v.s. 47.4% v.s. 57.3%, $\chi^2 = 3.09$, $p =$

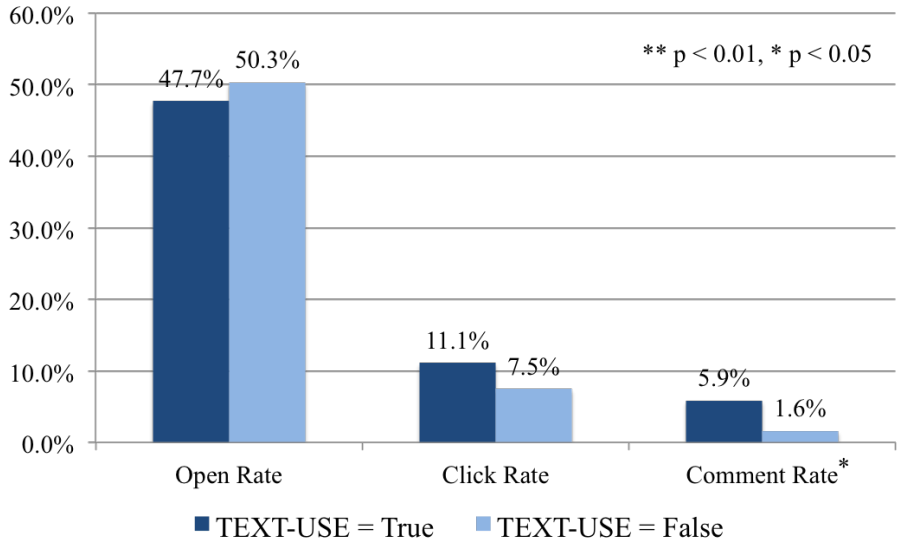


Figure 3.3: Comparison between the requests sent to non-members who expressed they used the resource in the citation context and those who didn't. The results show that the non-members who mentioned they used the resource in the citation context were significantly more likely to comment on the resource.

0.21, $n = 340$, $df = 2$, effect size = 0.096), link-click rate (6.0% v.s. 9.7% v.s. 11.0%, $\chi^2 = 1.38$, $p = 0.50$, $n = 340$, $df = 2$, effect size = 0.064), or comment rate (2.4% v.s. 3.4% v.s. 4.9%, $\chi^2 = 0.75$, $p = 0.69$, $n = 340$, $df = 2$, effect size = 0.047) (Figure 3.4). There are several possible reasons that we couldn't find significant differences across these conditions. First, our sample size may have been too small. Secondly, an author's sentiments may have changed since the paper was written. Furthermore, the sentiment analysis we performed was imperfect, and some contexts may have been mis-classified. We believe a follow-up study may be warranted. Nevertheless, at least in our current experiment, we were unable to find evidence for **H1c** and we conclude that a person's stated sentiment toward a resource (**TEXT-SENT**) might not be an important factor for contribution rate.

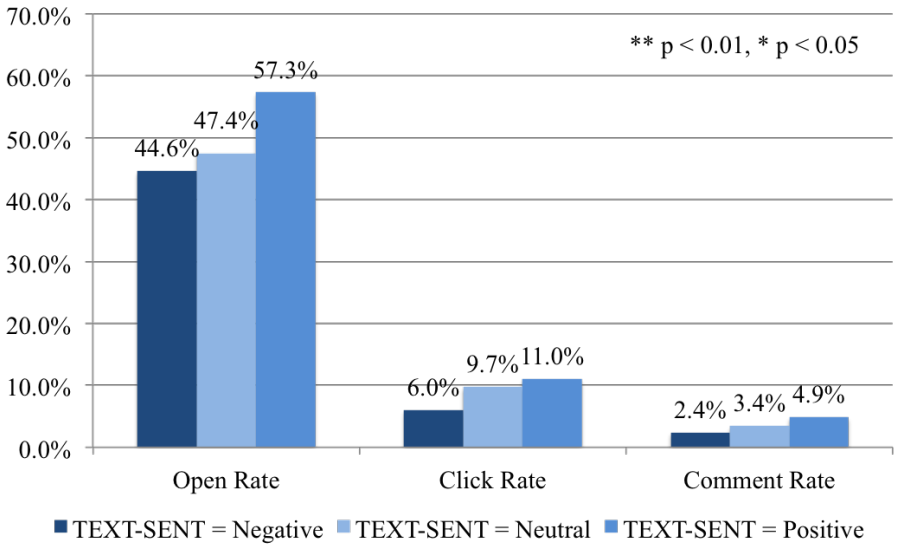


Figure 3.4: Comparison between the requests sent to non-members who expressed negative, neutral, and positive opinions in the citation context. The results show that there were no significant differences between the conditions.

3.4.3 The Effects of Foot-in-the-Door Request Design

Since the foot-in-the-door technique has been proven in a business context [53], we hypothesized that it might apply to online communities.

H2: *Non-members who initially receive a smaller "Yes/no" request are more likely to subsequently contribute a written review to the community.*

To test this hypothesis, we sent 403 request emails which asked a yes/no question first (see Appendix) and then invited a written comment (**REQ-FOOT** = *True*) and 407 request emails that directly asked the recipients to write a comment (**REQ-FOOT** = *False*). Recipients in both conditions were non-members who had cited either Weka, Scikit-learn, or the Pascal VOC Dataset. The email-open rates, link-click rates, and comment rates of the two conditions were compared.

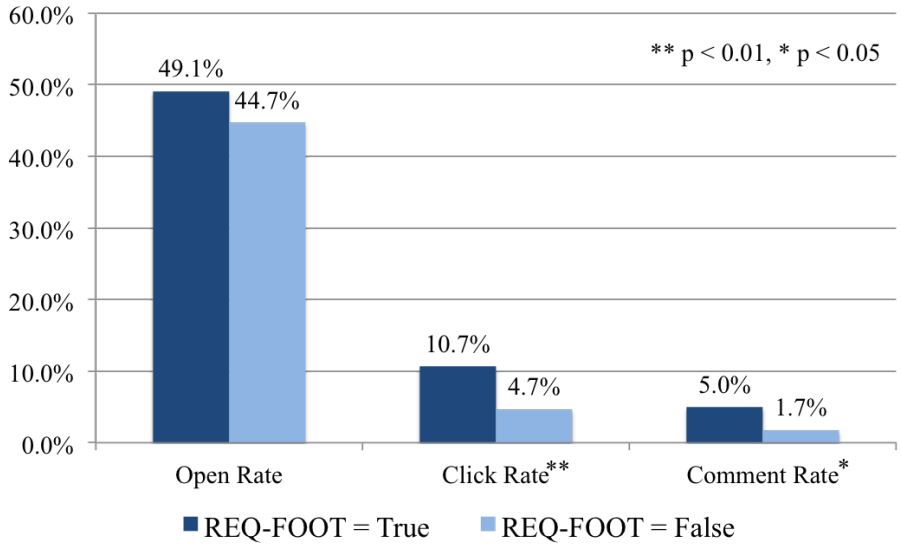


Figure 3.5: Comparison between the requests that asked a simple yes/no question first and the baseline. The results showed that requests that applied the foot-in-the-door technique lead to more clicks and more comments from the non-members.

The results showed that the email-open rates were similar between the two conditions (49.1% v.s. 44.7%, $\chi^2 = 1.58$, $p = 0.21$, $n = 810$, $df = 1$, effect size $= 0.044$), which makes sense because the subject lines were identical. However, the non-members who received foot-in-the-door requests were not only significantly more likely to click the link in the email (10.7% v.s. 4.7%, $\chi^2 = 10.32$, $p < 0.01$, $n = 810$, $df = 1$, effect size $= 0.113$), they were also significantly more likely to leave comments about the resource (5.0% v.s. 1.7%, $\chi^2 = 6.61$, $p = 0.01$, $n = 810$, $df = 1$, effect size $= 0.906$) (Figure 3.5). Thus, the findings support **H2** and show that the foot-in-the-door technique is an important tool for encouraging comments.

d	$i_{h,t}$	$P(c_{h,t}\|d, i_{h,t})$
REQ-FOOT	**TEXT-CITE**	0.050
REQ-FOOT	**TEXT-USE**	0.059
REQ-FOOT	**TEXT-CITE** $\wedge\neg$ **TEXT-USE**	0.016
REQ-FOOT	**TEXT-CITE** $\wedge$ **TEXT-SENT**=*Pos*	0.049
REQ-FOOT	**TEXT-CITE** $\wedge$ **TEXT-SENT**=*Neut*	0.034
REQ-FOOT	**TEXT-CITE** $\wedge$ **TEXT-SENT**=*Neg*	0.024
$\neg$ **REQ-FOOT**	**TEXT-CITE**	0.017

Table 3.3: The contribution probabilities with different design requests and the interest of a human in a task.

3.5 Simulation Experiment

To examine whether the decision-theoretic model we proposed really increases the utility of bootstrapping online communities, we conducted a simulation experiment using the synthetic data generated with the parameters learned from the previous experiments (Table 3.3) and real data collected from Microsoft academic search.

3.5.1 Method

To generate the citation graph which represents which authors cite which resources in their paper, we first parsed the publication information of all the AI researchers listed in Microsoft academic search[12]. This gave us a list of 266,101 authors, along with the number of publications each has produced. Then, we randomly sampled

[12] `http://academic.research.microsoft.com/`

400 authors[13] and generated the corresponding number of synthetic papers for each author. After that, we constructed the citation graph using the rich-get-richer model [49]. In this model, we first randomly sorted the papers; then, we created citations for the papers sequentially. We assumed each paper cites 29 papers (the mean number of citations for 10 randomly sampled papers was 28.8). For each citation, we randomly cite one of the previously processed papers with probability p_{cite} and randomly cite a paper cited by a previously processed paper with probability $(1 - p_{\text{cite}})$. We report experimental results with $p_{\text{cite}} = 0.5$.[14] This process ensures that the paper citations followed the power law. After the citation graph was generated, we randomly sampled 100 papers as the resources in the community. Based on data collected from our earlier experiments, we mark with 0.45 probability that the author of a citation really used that resource.

Based on the citation graph and the contribution probabilities we collected from our previous experiments (Table 3.3), we simulated the requests sent out by the community. We compare three methods for issuing requests:

1. **Random**: Send out requests that map the authors to the resources randomly.

2. **Greedy**: Based on the citation information, assign each author to the resource to which they are most likely to contribute.

3. **Decision-theoretic Optimization**: Issue requests using Algorithm 1.

We assumed the utility of the contributions of each task is $\log[100\mathbf{C}_t+1]$, where $\mathbf{C}_t$ is the number of contributions that are made to task t. We chose this utility function because it has the property of diminishing utility for each additional contribution, a reasonable assumption since our community does not benefit from having all the

[13] Our earlier experiments sent roughly this many emails.

[14] We found that our results improve as we decrease p_{cite}, so we chose 0.5 as a representative value.

contributions concentrated on only a few resources. We added 1 inside the log function to ensure nonnegative utilities. Since the expected utilities for many resources were less than one, we also multiply by 100 so that the utility is not dominated by the added constant factor.

We note that the Greedy and Random baselines are the strongest we could reasonably produce. For these baselines, we assign the authors with the highest probability of contributing to some resource first. Additionally, we break ties randomly, which has the effect of distributing contributions and dramatically improving the resulting utility.

3.5.2 Results

We generated 100 graphs using the method described in the previous section and simulated the requests sending in three different conditions: Random, Greedy, Decision-theoretic Optimization. The average expected utility of the three conditions on the five graphs are reported in Figure 3.6. The expected utility of the decision-theoretic algorithm is significantly higher than both baselines (using a two-tailed independent samples t-test). In particular, after issuing 400 requests, its expected utility is significantly higher than Random (58.9 v.s. 3.1, $p < 0.001$), and it also performed significantly better than a strong baseline which assigned the authors greedily to the resources they were most likely to contribute to (58.9 v.s. 54.4, $p < 0.001$). Importantly, the figure also shows that decision-theoretic optimization needs to issue only 55% (220) of the 400 requests in order to reach the maximum expected utility of the best (greedy) baseline.

In addition to this main result, we performed a sensitivity analysis that showed our results to be robust and unchanged when we are not given the true probabilities of contribution. In this analysis, we provided the algorithms with access to the expected probability values used in our experiment, but sampled the true values from normal distributions centered at those values (and truncated at 0 and 1). Increasing the

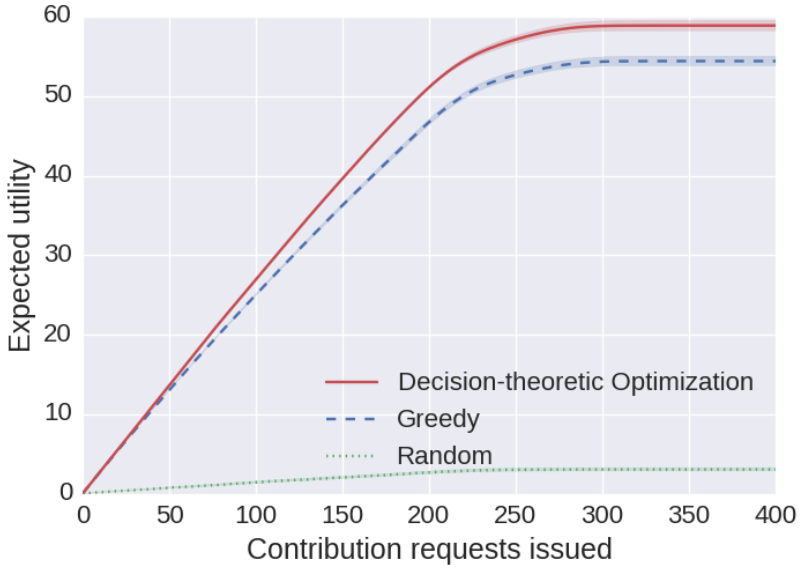


Figure 3.6: Decision-theoretic optimization achieves significantly higher expected utility and requires 55% as many requests (220) to match the maximum expected utility of the strongest baseline. Plot shows the mean expected utility over 100 simulations, with shaded 95% confidence intervals.

variance of these distributions until 2 standard deviations equaled the probability value itself did not alter our findings. Compared to the best (greedy) baseline, the decision-theoretic optimization method still achieved significantly higher expected utility after 400 requests (59.4 vs. 54.9, $p < 0.001$) and required only 56% as many requests to reach the maximum expected utility. This result is promising for a real-world deployment, where actual probability values would be drawn from a distribution rather than the expected value of that distribution.

3.6 Limitations and Implications

We are encouraged by the positive results from our experiments, but caution that there are several limitations to our study and our proposed method.

First, we only tested our method in one community. It may be the case that Open AIR is the best case for our datamining method, given the quality of corresponding data in Google Scholar. While more experiments are needed to demonstrate that our approach generalizes, there are other communities where the approach has worked or is worth considering:

- **AirBnB / Craigslist:** AirBnB reputedly bootstrapped their inventory of rental properties by crawling Craigslist for candidate homeowners who had listed properties, sending them emails from supposed AirBnB "fans." [142]. While this is a blackhat example that may have violated Craigslist terms of service, it illustrates the method's applicability.

- **SummitPost / CascadeClimbers:**[15] The SummitPost community maintains an online guidebook of mountain-climbing route descriptions for a worldwide audience. CascadeClimbers is a regional community website where climbers post pictures, accident updates and trip reports describing their outings. In contrast to the AirBnB / Craigslist example, these communities are complementary, not competitive. Since someone who has written a trip report has the knowledge to turn their tale into a more comprehensive and instructive route description, one might attract SummitPost contributions through the CascadeClimbers forum reply feature. For full coverage, one would wish to mine other regional sites as well.

[15] Respectively, at `www.summitpost.org` and `cascadeclimbers.com`.

- **500px / Reddit:**[16] 500px caters to a community of professional photographers who wish to showcase and sell their work to stock photo buyers. Reddit is a hierarchically-organized social news site; the photography subreddit features a wide ranging conversation about images and techniques. Someone who has posted several of their pictures on reddit is, therefore, a reasonable candidate member for 500px and might welcome a private message suggesting the site.

- **Movie Reviews / Twitter:** One might consider bootstrapping a movie review website by parsing Twitter posts for movie hashtags and using text mining techniques to see whether the author expressed strong sentiment indicating proclivity to posting a review. While this example satisfies our requirement for "datamining bootstrappability," Twitter is noisy enough that such an approach seems unlikely to work.

The success of our recommended bootstrapping approach depends not just on the three conditions in our definition, but also on empirically determined parameters, such as the number of candidate nonmembers that can be unearthed via datamining, the accuracy of targeting and resulting response rate, the quality of the contributions, and the utility derived by the community over time. Further empirical studies are needed to determine how general is our method.

In addition, the performance guarantee of Algorithm 1 is based on the monotone submodularity of the utility function. Our algorithm might not perform as well when the community has a utility function with different properties. For example, a community might want contributions to focus on a few resources so resource popularity can attract newcomers to the community. However, the goal of this chapter is not to provide a definite algorithm that can apply to every online community. Instead, we are trying to establish a decision-theoretic framework which allows the community designer to design their own algorithm that maximizes the community's utility. In

[16]See `https://500px.com` and `https://www.reddit.com/r/photography`.

the future, we plan to work with other online communities and come up with different optimization algorithms based on their individual utility functions.

3.7 Discussion and Future Work

In this chapter, we define bootstrapping an online community as a decision theoretic optimization problem. Although an optimal solution to the problem is combinatorially prohibitive, we present an efficient greedy algorithm and prove that it allocates requests within a constant factor of optimal. To demonstrate the practicality of our approach, we consider Open AIR, a newly created community for researching and reviewing open AI software and data resources. We show that text mining techniques, applied to Google Scholar pages, can extract several strong features that correlate with a person's interest in contributing a review.

Specifically, our results show that people who have authored a paper that cited an article describing a resource are more likely to comment on the resource than people who did not. Furthermore, by mining the context of these citations, our system can detect people who actually *used* the resource in their work; these people are significantly more likely to comment on the resource than people who simply acknowledge the resource as related work. Although we expected that strong positive or negative sentiment in the citation context would also signal a greater willingness to comment, the evidence did not support this conclusion.

Furthermore, our study shows that effective request design is an essential factor when encouraging non-members to contribute. Specifically, we found that first asking people a simple request (e.g., a binary "yes/no" question like "Would you recommend the resource?") significantly increased the likelihood that they would contribute the more time-consuming full-text review. This finding confirms the usefulness of the *foot-in-the-door* technique [53] in the context of bootstrapping an online community.

In the course of our experiments we were also able to learn parameter values for the conditional probabilities needed by our decision-theoretic model. Based on this

information, we ran a simulation experiment showing that decision-theoretic control achieves comparable expected community utility while issuing only 55% as many contribution requests, compared to a strong baseline approach.

These results suggest that using these methods, a self-improving crowdsourcing agent can efficiently recruit crowdsourcing workers with specialized expertise or experience not available in a crowdsourcing marketplace. Further, future crowdsourcing system designers should investigate ways to support requesters (or meta-workers) performing the type of feature engineering and request design experiments we conducted (which help to provide a self-improving crowdsourcing agent with a better model of its environment and more effective actions). It may be possible to design libraries of reusable features and actions that may help with multiple tasks, so that the do not need to be redesigned for each new task.

We are now ready to deploy the model on an even larger-scale to see if we can complete the bootstrapping process and bring Open AIR across the tipping point to self-sustaining traffic. Additional important directions for future research include adding explicit budget constraints to our model, considering a wider range of utility functions, experimenting with additional request designs and predictive features (e.g., the *age* of a citation may influence response rate), and applying our method to other online communities (Section 3.6 mentions several candidate communities). Since the pairing of Google Scholar and Open AIR may represent the best case for datamining-based bootstrapping, further experimentation will help demonstrate the generality of our approach.

Chapter 4
OPTIMAL WORKER TESTING AND TRAINING

Chapter 3 described methods for recruiting workers who are not part of the current community or marketplace. Once workers have agreed to participate in the task, supporting and monitoring their progress is the next critical part of the task pipeline (Figure 1.2). Toward this end, many crowdsourcing requesters include test questions to ensure workers are providing high-quality answers; training has also been shown to be important [110]. However, tuning the amount of testing or instruction requires expensive experimentation on the part of the requester.

This chapter presents methods that enable a self-improving crowdsourcing agent to improve how it manages workers once they have joined the task. These methods consist of control algorithms that improve agent performance over time (by interacting with workers and computing more accurate estimates of its environment model). No requester or meta-worker involvement is needed; the requester simply needs to specify a desired minimum accuracy for the data produced by workers. However, the requester can supply the agent with a starting policy that can help to focus the agent's exploration. Unlike the problem of recruiting workers (Chapter 3), the management problem can be formulated for individual workers over a smaller space of actions, allowing for model-based planning over longer sequences of actions.

4.1 Introduction

Ensuring high-quality results for crowdsourcing tasks is challenging, because of the high variability in worker skill. Accordingly, many AI researchers have investigated quality control algorithms. Two prominent techniques have emerged for the prob-

lem [162]: (1) periodically inserting gold questions (those with known answers) for each worker, in order to estimate worker reliability, and then firing workers who fall below an acceptable accuracy threshold, and (2) employing agreement-based approaches that do not use gold data, instead using expectation maximization (EM) or similar unsupervised methods to jointly estimate worker reliability and task output quality.

These two methods have complementary strengths and weaknesses. While the former is simpler and easy to understand, it puts the onus on the task designer to supply gold questions. Moreover, it is prone to 'bot attacks where, over time, workers identify all gold questions and create 'bots that answer those correctly while answering other questions randomly [127]. On the other hand, the unsupervised approach is technically more sophisticated, but requires significant amounts of data per worker and workers per task before low-quality workers can be reliably identified. Thus, it does not provide a quick filter for firing errant workers, nor does it work when workers do few jobs before quitting.

In practice, while unsupervised techniques have garnered significant research interest (e.g., [40, 163, 164]), the simpler technique of inserting gold questions is the norm in industry. CrowdFlower, a major crowdsourcing platform and consulting company, calls gold questions *"the best way to ensure high quality data from a job"* [38]. Furthermore, many research projects that use crowdsourcing to generate training data eschew EM-based quality control and simply insert gold questions [4, 21, 61, 169].

While insertion of gold questions is popular in practice, to the best of our knowledge, there is no formal model of when and how many gold questions to insert. CrowdFlower's rule of thumb is to have 10–20% of data as gold, and to insert one gold question per page [38]. Such a policy is arbitrary and may waste valuable budget, e.g. it may insert too many gold questions for simple tasks and too few for difficult tasks. Moreover, and more importantly, such a policy is static and does not adapt to individual workers or the current mix of spamming and diligent workers. The

percentage of testing questions should only be high if a large percentage of the labor pool are poor quality workers or spammers. Similarly, an adaptive policy that tests less once it is certain that a worker is diligent will likely perform better than a static one.

We formulate the problem of balancing between (1) testing workers to ensure their accuracy and (2) getting work done as a Partially Observable Markov Decision Process (POMDP). Our worker model captures the possibility that worker performance may degrade over time and workers may leave our system after any question. Our model also takes as input a desired accuracy of the final output, and a base testing policy. We apply reinforcement learning over our POMDP to dynamically improve the given base policy with experience. Evaluations on both synthetic data and real data, from Amazon Mechanical Turk, show that our agent is robust to various parameter settings, and typically beats the baseline policies used by most requesters, as well as the input base policy. Furthermore, our method is fully automated, easy to apply, and runs mostly out of the box. We release our software for further use by the research community.[1] Overall, we make the following contributions:

1. We use a POMDP model to formulate the problem of varying the number and placement of gold test questions in order to optimally balance the quality-cost tradeoff for crowdsourcing.

2. We present an adaptive reinforcement learning algorithm that simultaneously estimates worker reliability (by inserting gold questions) and assigns work to produce high-quality output.

---

[1] `https://crowdlab.cs.washington.edu/optimal-training-and-testing-for-crowd-workers.html`

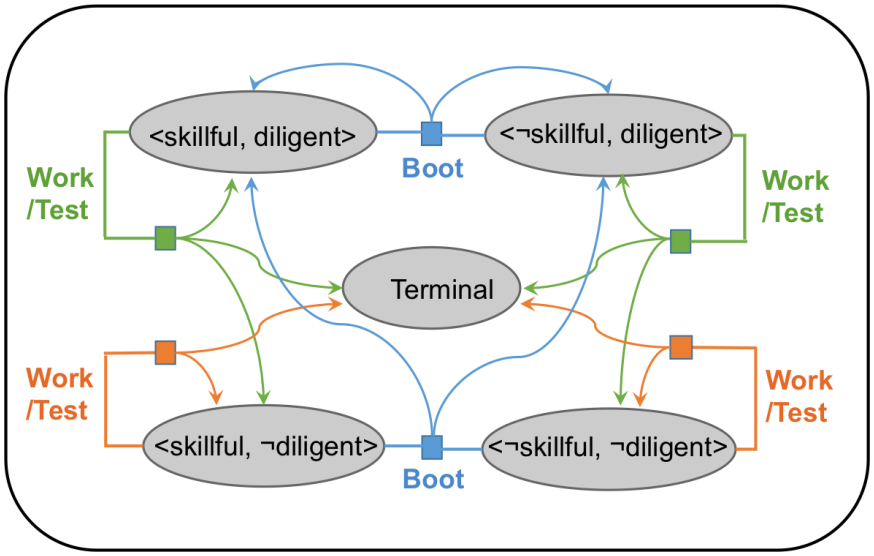


Figure 4.1: Transition dynamics of the Worker Controller POMDP. Note that the reward and observation model for Work and Test actions are different, but their transition models are the same, as shown.

3. We comprehensively test our model using simulation experiments and show that our algorithm is robust to variations in desired output accuracy, worker mix, and other parameters.

4. Additional experiments on three live data sets (collected using Amazon Mechanical Turk) show that our reinforcement learning agent produces up to 111% more reward than common policies found in the literature.

4.2 Worker Control

We propose a controller that automatically decides at each time step whether a specific worker should answer a test question, a work question, or whether this worker's performance is not up to the mark, and so this worker should be fired and a replacement worker hired. The actuators that actually perform these actions are part of a

web application, which selects new questions from a database of questions and interacts with a crowdsourcing platform to recruit new workers. Our controller should be adaptive, i.e., it should make decisions for each worker, based on their past behavior, instead of following some pre-defined static policy. The global objective of the controller is to get workers to answer as many questions as possible while obtaining a minimum target accuracy supplied by the requester. We first describe key design decisions in formulating this controller.

At a high level, this is clearly a sequential control problem, where the controller gets successive observations about a worker from test questions, and needs to take successive actions based on this history. This problem is *partially* observable because a worker's accuracy isn't known exactly and can change over time. However, the controller can maintain a probability distribution over each worker's accuracy based on its observations. This suggests a controller based on a Partially Observable Markov Decision Process (POMDP).

A POMDP [77] is a popular AI formalism for modeling sequential decision making problems under uncertainty. In a POMDP the state is not fully observed, although observations from information-gathering actions help the agent to maintain a *belief*, which is a probability distribution over the possible world states. A POMDP's objective is to maximize the long-term expected reward minus cost. The solution of a POMDP is a policy that maps a belief into an action.

One natural modeling choice for our worker controller POMDP would be to include the exact accuracy of the worker, a number in $[0, 1]$, in the world state. This accuracy could then be estimated over time based on the test questions. Unfortunately, solving POMDPs over a continuous state space is known to be notoriously hard [133]. Discretizing the accuracy space might be a workable alternative, though it may still be unnecessarily complex since many accuracy bins may result in the same policy.

In response, we adopt a *two-class*[2] worker model, assuming that a worker is randomly drawn from one of the two classes: skillful and unskillful. Both classes of workers have their own (initially unknown) accuracy distributions with mean accuracy of the skillful class higher than that of the unskillful class. This abstraction is beneficial for three reasons. First, for each individual worker the information gathering is limited to ascertaining which class they belong to, and no estimation of individual accuracy or accuracy bin is needed. Second, the worker population mix is abstracted into class means, which can be estimated using an initial exploration phase; these parameters are global, and once estimated apply to the whole population and need not be re-estimated per worker. Third, we can estimate these latent variables in closed form without the need for computationally-expensive approximate inference subroutines within our reinforcement-learning algorithm.

Our model also includes a parameter for workers leaving the task at will. We treat this as a global parameter for the whole worker population. Finally, we also realize that a key reason for repeated testing is that workers can become complacent and their performance may lapse over time. We model this by adding a variable (D) in the state, which, when to set to 1, represents that the worker is diligent (not complacent), and answering to the best of their ability. We now describe the controller in detail.

4.2.1 The POMDP Model

Defining a POMDP requires specifying (or learning) a state space, action space, transition and observation probabilities, and a reward function. We first describe these aspects of our controller POMDP.

- The state space $\mathcal{S}$ can be factored as $\langle C, D \rangle$, where C is a variable indicating the worker class (skillful or unskillful) and D is a Boolean variable indicating

[2]We performed preliminary experiments with larger state spaces and found the gains to be insignificant. Moreover, these more expressive models require significantly more data to learn, which reduces their empirical effectiveness.

whether the worker is diligent ($D = 1$). The state variable D captures behavior where workers may lapse and start answering without focusing on the problem (with a consequent drop in accuracy); it also models the possibility of spammers faking high-quality work until they think a requester has stopped testing. Additionally, we define a special terminal state denoting the end of process.

- The set of actions $\mathcal{A}$ available to our agent in any non-terminal state consists of *test* (administer a test using a gold question), *work* (ask a question with an unknown answer), and *boot* (terminate employment and hire a replacement worker). No action is available in the terminal state.

- The transition function $P(s' \mid s, a)$, depicted pictorially in Figure 4.1, specifies a probability distribution over new states s' given that the agent takes action a from state s. We assume that workers are diligent to start, so any new worker starts from a state with $D = 1$ and C unknown. The POMDP's belief is a probability distribution over C which is initialized with a prior distribution over classes. This prior is estimated by a single class mix parameter that learns what fraction of workers are a priori skillful and what fraction unskillful.

  When the controller agent takes *test* or *work* actions, the worker may decide to leave the task with (unknown) probability p_{leave} and transition to the terminal state. Moreover, if the worker is in a state $\langle C, D = 1\rangle$, she may transition to a spamming state $\langle C, D = 0\rangle$ with probability $p_{\text{lapse}}(1 - p_{\text{leave}})$ or remain in the same state with probability $(1 - p_{\text{lapse}})(1 - p_{\text{leave}})$. If the worker is already in spamming state $\langle C, D = 0\rangle$, she remains there with probability $(1 - p_{\text{leave}})$. The *boot* action fires this worker and hires a replacement worker with initial belief as described above.

- Observation function: The agent can only directly observe responses to *test* actions. If the worker is in a state with $D = 1$, we assume she answers correctly

according to her unknown class accuracy μ_c. Workers with state variable $D = 0$ spam with accuracy 0.5, i.e., they resort to random guessing of a binary-valued question. Additionally, the agent directly observes when a worker leaves.

- Reward function: The agent incurs a cost c for each *test* and *work* action it takes (assuming the worker provides a response). Additionally for *work* actions, our reward model incurs a penalty PN for each incorrect answer and a reward RW for each correct answer it receives. Since *work* actions ask questions with unknown answers, the POMDP needs to compute an expected reward, as follows:

$$R = \mathrm{E}_{Y,X}\left[f(y, x)\right],$$

where

$$f(y, x) = \begin{cases} \text{RW, if } y = x \\ \text{PN, otherwise.} \end{cases}$$

Here Y and X are binary random variables for the latent true answer to the question and the worker response, respectively. This expectation can be computed using joint probabilities $P(Y, X) = P(Y)P(X \mid Y)$, where $P(X = y \mid Y = y)$ is the worker accuracy. This accuracy depends on the current state in the POMDP. We assume that each worker class c has its own latent mean accuracy, μ_c, but that when $D = 0$, workers generate random answers.

So that the requester may specify this reward function in an intuitive way, our experiments focus on the setting where RW = 1 and the value of PN will vary depending on the requester's desired accuracy. If the requester would like to gather answers from workers with accuracy greater than a^*, she may specify PN $= a^*/(a^* - 1)$, which will induce positive reward[3] only for workers with accuracy greater than a^*.

As defined, the POMDP, in its pursuit to maximize its long-term expected reward, should learn to test and boot unskillful workers (if their class accuracy is less than a^*)

[3]This formula comes from setting the expected reward $(a\text{RW} + (1 - a)\text{PN})$ to 0 when RW = 1.

in order to obtain a positive reward. It should also periodically test skillful workers in order to verify that their performance hasn't begun to degrade. The exact parameters of how often and how much to test depend on the unknown mix of the worker classes, target accuracy, model parameters p_{lapse} and p_{leave}, as well as the belief on the current worker at hand.

Our POMDP need only reason about distributions over five world-states (based on different assignments of C and D, and the terminal state), and three actions; thus, it can easily be solved using most modern POMDP algorithms.

4.2.2 Reinforcement learning

When our system is deployed in a new crowdsourcing environment, it must also learn the POMDP model parameters. This necessitates a reinforcement learning solution with an exploration-exploitation tradeoff.

Five parameters need to be learned: p_{leave}, p_{lapse}, the class mix in the worker population, and the mean accuracies μ_1 and μ_2 for the two classes. Learning these parameters accurately could be data intensive, which means that a typical controller starting from scratch may waste significant budget in learning the model. It may, as part of exploration, boot skillful workers, leading to worker dissatisfaction and subsequent bad requester rating by workers. Most requesters would not be able to afford this.

To alleviate this concern, we allow the requesters to specify a base policy, say, something similar to CrowdFlower's recommended best practice policy. Our reinforcement learner can start with this base policy instead of a random policy. It will gradually transition to following the POMDP policy once it has observed enough workers to estimate parameters. Common base policies that insert a fraction of test questions are desirable because (1) they are widely adopted and easy to implement and (2) the inserted test questions enable our agent to estimate parameters accurately.

To implement this mixed off/on-policy learning strategy, when the controller agent hires a new worker, it follows the base policy with probability $q(b, B)$ and the POMDP policy with probability $(1 - q(b, B))$, where b is the budget spent so far and B is the total budget. When the agent decides to follow the POMDP policy, it reestimates model parameters and replans prior to hiring a new worker. We define our particular choice of function q in the next section.

To estimate parameters, we treat the sequential data as an input-output hidden Markov model [11] and use the Baum-Welch (EM) algorithm initialized with parameters from the previous episode and one random restart. We use default uninformed priors of Beta(1, 1) on parameters, but use a prior of Beta(5, 2) for class accuracy and a prior of Beta(2, 20) for p_{lapse}. Beta(5, 2) encodes the fact that the accuracy should be at least 0.5 for Boolean questions, and Beta(2, 20) is an optimistic prior that workers don't tend to become spammers very frequently.[4] We assume that each class has its own accuracy but use parameter tying to estimate values for p_{leave} and p_{lapse} that are shared between classes.

We also experimented with an estimation process that is identical, except that it does not estimate the class accuracies. Instead, it defines two worker accuracy bins, one on $[a^*, 1.0]$ for skillful workers and one on $[0.5, a^*)$ for unskillful workers and fixes $\mu_1 = (a^* + 1.0)/2$, $\mu_2 = (0.5 + a^*)/2$. While these accuracy means may differ from the maximum likelihood estimates, they still allow the agent to distinguish between workers expected to give work of sufficient quality (positive reward) and workers the agent should boot. This variant has two primary benefits. First, estimating only the class ratios requires much less data than estimating both the class ratios and the accuracy means. Second, it is possible that the maximum likelihood estimates for accuracy means will produce means $\mu_1, \mu_2 < a^*$. Fixing an accuracy bin above a^*

[4] In preliminary experiments, we found an optimistic prior was important to enable our system to distinguish between skillful workers who may drop in accuracy and unskillful workers who had low accuracy from the start.

ensures that the agent will be able to identify high-quality workers even if the mean accuracy μ_1 for skillful workers is below the desired accuracy a^*.

4.3 Experiments

In experiments, we first test our agent with simulated workers under a variety of settings; the goal is to assess the robustness of our method to differing task settings and its ability to adapt to attacks by different populations of spammers. Later, we demonstrate performance on three NLP datasets using real workers from Amazon Mechanical Turk.

We implement a reinforcement learning agent (POMDP-RL) that improves upon a base policy of inserting 20% gold questions and firing workers if their accuracy is less than the desired accuracy (Test-and-boot). We compare the RL agent with this base policy on its own, as well as a baseline policy that inserts no gold questions (Work-only). The Test-and-boot base policy inserts a batch of 4 test questions in every set of 20 questions. In simulation experiments, we are also able to compare performance with an agent that has access to the true model parameters (POMDP-oracle).

POMDP-RL learns the class ratio, class mean accuracies (μ_1, μ_2), p_{leave}, and p_{lapse} using the priors specified in the previous section. We use the Test-and-boot policy in each experiment as our base exploration policy. We used the ZMDP POMDP package,[5] which implements the Focused Real Time Dynamic Programming (FRTDP) algorithm for solving POMDPs [152]. We ran the solver with default configuration settings, maximum solve time of 1 minute,[6] and discount factor of 0.99. In order to speed up experiment run-times, the agents recomputed the POMDP policy at most once every 10 workers and at most 10 times total.

---

[5] `https://github.com/trey0/zmdp`

[6] Solver ran on 2.3 GHz Intel Xeon E7-8850-v2 processors.

4.3.1 Agent Robustness

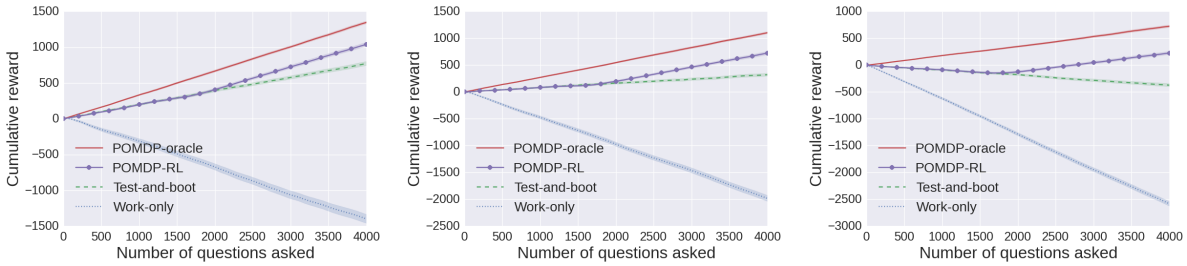
In our simulation experiments, we considered two classes of workers, one of high accuracy ($\mu_1 = 0.9$) and one of low accuracy ($\mu_2 = 0.6$). Worker accuracy in each class varies according to a truncated normal distribution parameterized by the class mean and $\sigma = 0.1$ and bounded on $[0.5, 1]$. Unless otherwise noted, we consider a 50:50 mixture of these two classes, where workers degrade with probability $p_{\text{lapse}} = 0.01$. Only the simulator and POMDP-oracle had access to the true worker parameters; POMDP-RL estimated parameters based only on observations.

Our experiments are from the point of view of a requester who wants to ensure data quality above $a^* = 0.75$. Our penalty function gives us PN $= -3$, which provides positive rewards for data above this accuracy. The default budget size is $B = 4000$ questions. In this set of experiments, we used the sigmoid exploration function $q(b, B) = 1 - 1/(1 + \exp(40(b/B - 0.4))$, which approximately changes from 1 to 0 in the range of 25% to 50% of the budget.

RQ1 Is our agent robust to the ratio of skillful to unskillful workers in the labor pool?

In these experiments, we varied the mixture of skillful workers to unskillful workers, while keeping the worker distribution properties fixed. Figure 4.2 shows that as this ratio decreases from 80:20 to 20:80, policies achieve lower reward, as we would expect with just a few skillful workers. The relative benefit of the POMDP agents, however, remains significant. In all settings, POMDP-RL learns a policy that gains roughly as much reward per question as the POMDP-oracle. This can be seen by the fact that in the last half of the budget, POMDP-oracle and POMDP-RL have similar reward slopes.

Both POMDP-oracle and POMDP-RL earn significantly higher total cumulative reward after spending the budget than the best baseline policy (Test-and-boot), ac-



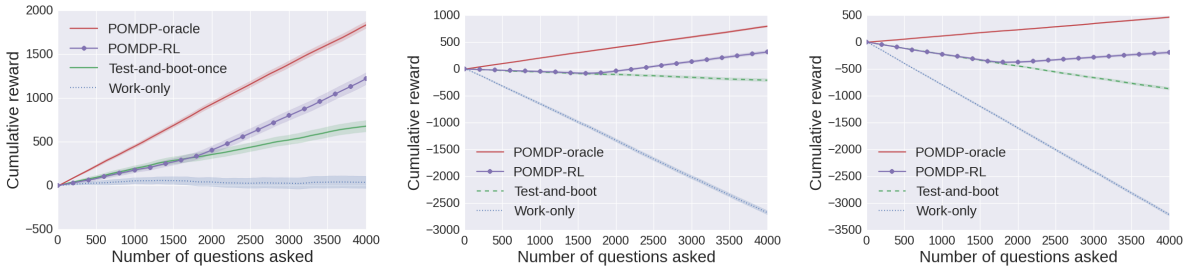
(a) 80:20 class ratio. (b) 50:50 class ratio. (c) 20:80 class ratio.

Figure 4.2: Our controller agent achieves significantly higher rewards than baseline policies under a variety of ratios of skillful to unskillful workers. As can be seen by the slopes of the curves, after learning, the RL policies start performing about as well as the optimal POMDP policy, which was given true parameters. These plots (and all subsequent reward plots) show mean performance over 200 runs with shaded 95% confidence intervals.

cording to two-tailed T tests ($p \ll 0.001$ for all; $t = 32.5$, $t = 13.2$ for 80:20; $t = 30.9$, $t = 13.8$ for 50:50; $t = 55.3$, $t = 27.7$ for 20:80). The differences tend to become more significant for settings with fewer skillful workers, since careful testing becomes more important there.

RQ2 Is our agent robust to the fraction of workers who stop being diligent (eventually answering randomly)?

In order to test how our agent responds to workers who may begin to spam with probabilities other than 0.01, we experimented with $p_{\text{lapse}} \in \{0, 0.2, 0.4\}$, as shown in Figure 4.3. Note that for $p_{\text{lapse}} = 0$ (Figure 4.3a), POMDP-RL uses the base policy of Test-and-boot-once, which tests 4 times *only at the start* (and fires the worker if they answer more than one question incorrectly). For this setting, we don't need a policy that tests at intervals; it is optimal to do all testing at the beginning if



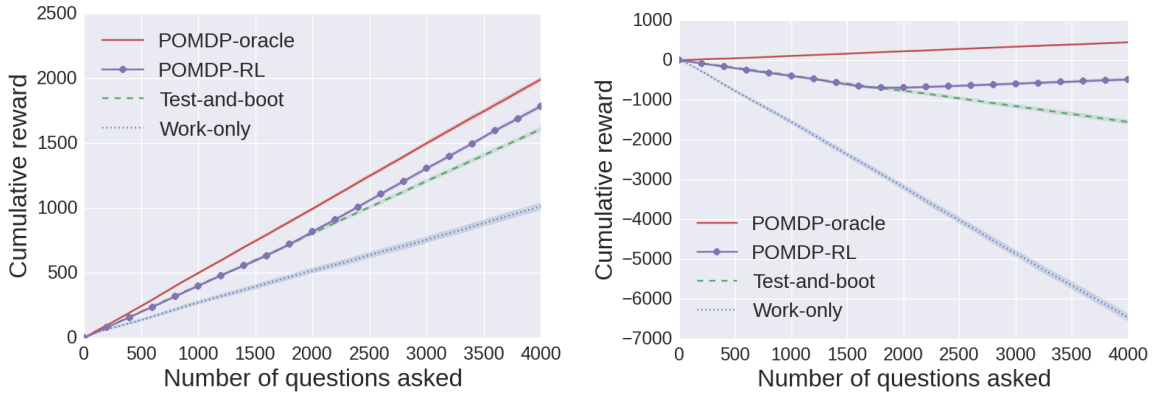
(a) $p_{\text{lapse}} = 0$. (b) $p_{\text{lapse}} = 0.02$. (c) $p_{\text{lapse}} = 0.04$.

Figure 4.3: Our RL controller is also robust to the number of deceptive spam workers, beating the baseline exploration policy, and eventually matching the performance of the POMDP-oracle model (which has knowledge of the true parameters). When $p_{\text{lapse}} = 0.04$, 4% of previously diligent workers start answering randomly after each question.

worker performance does not drop. This set of experiments used synthetic data from the default equally-sized worker classes. In all experiments, the POMDP-oracle and POMDP-RL agents again obtain significantly higher reward than the baselines.

RQ3 Is the RL controller robust to changes in the requester's utility function (i.e., desired accuracy)?

The previous experiments used PN $= -3$ (desired minimum accuracy of 0.75). In this set of experiments, we varied the utility function by setting PN $= -1$ and PN $= -6$, corresponding to desired accuracies of 0.5 and approximately 0.86, respectively. As shown in Figure 4.4, the relative gains of adaptive testing increase as the desired accuracy (and corresponding magnitude of penalty) increase. Note that for PN $= -6$ (Figure 4.4b), only the POMDP-oracle agent has final positive cumulative reward. After having spent some budget, POMDP-RL also learns a policy that improves cumulative reward (neither baseline policy is able to improve reward). In both



(a) Penalty -1 (desired accuracy 0.5).

(b) Penalty -6 (desired accuracy 0.86).

Figure 4.4: Our agents perform robustly when supplied different utility functions and produce higher relative rewards for utility functions corresponding to higher desired accuracies.

experiments, POMDP-oracle and POMDP-RL produce significantly higher reward than the best baseline policy (Test-and-boot).

RQ4 How good is the learned RL policy compared to the POMDP with known parameters?

Figures 4.2 through 4.4 show that POMDP-RL is able to learn policies with reward slopes similar to those of POMDP-oracle. This research question evaluates the policy learned by POMDP-RL isolated in a pure exploitation phase on the last 10% of an experiment with $B = 2000$ and the default worker configuration (50:50 class ratio, $p_{\text{lapse}} = 0.01$). Figure 4.5 shows that the reward obtained by POMDP-RL is on par with POMDP-oracle (there is no statistically significant difference in cumulative reward at the end of the budget), suggesting that the combination of base exploration policy and exploration-exploitation tradeoff is learning effectively.

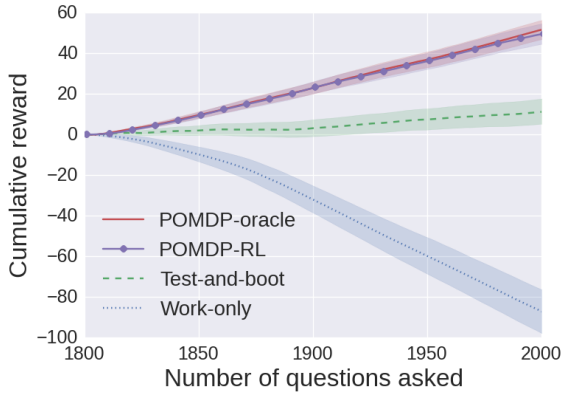


Figure 4.5: Zooming into the agent's behavior during the last 10% of the budget (i.e., after most learning), there is no statistically significant different between our RL controller and the agent that is given the true model of workers by an oracle. This figure uses the same worker distribution as Figure 4.2b (50:50 class mixture and $p_{\text{lapse}} = 0.01$).

Examination of worker traces shows that POMDP-RL and POMDP-oracle take similar actions. Both have an initial testing phase to establish worker quality, and then periodically insert a test question in every batch of approximately 6 questions. If a worker answers the single test question incorrectly, the agent will administer additional test questions adaptively.

4.3.2 Testing on Real Workers & Tasks

To answer the question of how well our agent performs on real datasets, we conducted experiments using three datasets gathered on Amazon Mechanical Turk. The worker completes an Entity Linking task in which a sentence and a mention (a portion of the sentence) is shown, and the worker is asked to match the mention to the correct Wikipedia entry. Two of our datasets, *LinWiki* and *LinTag*, were supplied by Lin et al. [100], who had Mechanical Turk workers answer questions using two different

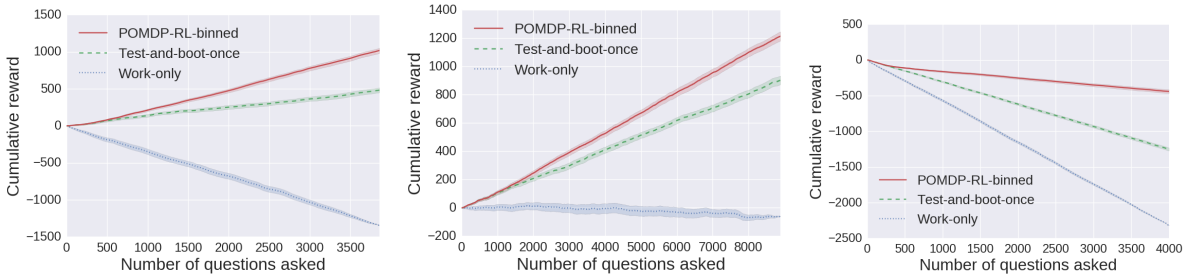
styles of questions, which they called WikiFlow and TagFlow. These datasets consist of 110 questions. 135 workers supplied 3,857 answers in LinWiki, and 149 workers supplied 3,999 answers in LinTag.

Our third dataset, *Rajpal*, consists of 150 questions from the same task. Rajpal et al. [136] recruited workers with Mechanical Turk Masters qualifications (35 workers, 3,015 answers) as well as without those qualifications (108 workers and 5,870 answers). We combined these two sets of responses into a single dataset (143 workers, 8,885 answers) for this experiment.

When performing experiments, we set the budget B equal to the total number of answers. The Work-only policy used every answer from every worker exactly once upon completing this budget. Since the other policies may boot workers and therefore require more workers than exist in the original dataset while consuming the budget, we recycle workers as needed for those policies. Our simulator randomizes the order of workers and the order of worker answers because the datasets do not contain metadata (e.g., timestamps) that would let us determine the order in which answers were received.

Since answers from a worker arrive in random order, expected worker quality should not change over time; we fix $p_{\text{lapse}} = 0$. Thus, these experiments use the Test-and-boot-once variant of the base policy, which performs one block of testing at the start only (and boots if the accuracy in that block is below the desired accuracy).

We set the desired accuracy to 0.85, a value close to the upper bound of what is reasonable, since only a small fraction of answers in the LinTag dataset come from workers above this accuracy. To give the base policy enough granularity when determining worker accuracy, the base policy tests 7 times (and boots if more than one answer is incorrect). We fix class mean accuracies μ_1, μ_2 for POMDP-RL using the binning method to ensure that our method can identify workers above the desired accuracy even if the maximum likelihood class means are below that value. Since the



(a) LinWiki dataset. (b) Rajpal dataset. (c) LinTag dataset.

Figure 4.6: Our RL controller (POMDP-RL) significantly outperforms baseline policies on all three datasets with real workers. Figures show mean performance over 200 runs (with shaded 95% confidence intervals).

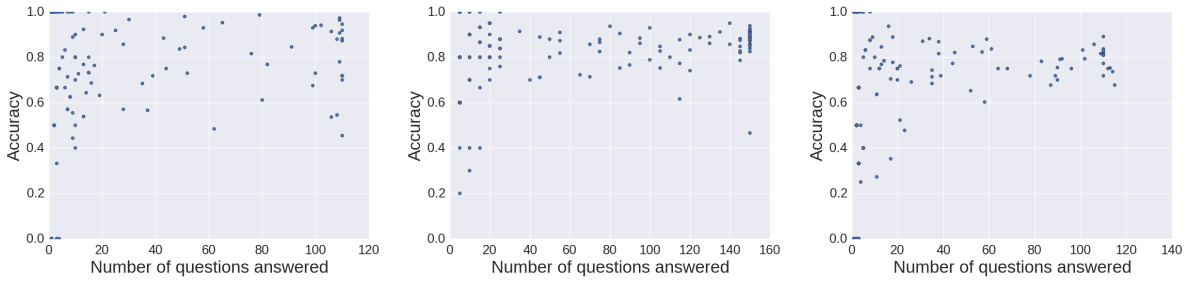
agent does not need to estimate these parameters, we explore only for the first 20 workers.

Note that workers cannot distinguish a gold question from a non-gold question; they have exactly the same effect on a worker. Since we know the correct answers, we can (post-hoc) treat any question as gold, and evaluate any possible policy.

Performance on these three datasets is summarized in Figure 4.6 and Table 4.1. Note that we are only able to run the POMDP-RL agent (not POMDP-oracle), since we do not know the true worker parameters. After consuming the budget, POMDP-RL generated 111% more cumulative reward than the best baseline for the LinWiki dataset (1018.2 vs. 481.7) and 35% more reward for the Rajpal dataset (1214.5 vs. 902.5). On the LinTag dataset, all methods produced negative reward, but POMDP-RL produced only 35% as much negative reward as the best baseline. The best baseline in each case is Test-and-boot-once. Running a two-tailed T test on these rewards determines that the differences are significant for all datasets ($t = 21.5$, 13.6, and 46.7 for the LinWiki, Rajpal, and LinTag datasets, respectively; $p \ll 0.001$).

Dataset	Policy	Reward	Labels	Accuracy
LinWiki	POMDP-RL	***1018.2**	2656	**90.8**
LinWiki	Test-and-boot-once	481.7	2895	87.5
LinWiki	Work-only	-1346.6	3857	79.8
Rajpal	POMDP-RL	***1214.5**	6867	**87.7**
Rajpal	Test-and-boot-once	902.5	7629	86.8
Rajpal	Work-only	-62.0	8885	84.9
LinTag	POMDP-RL	***-439.5**	1253	**79.6**
LinTag	Test-and-boot-once	-1251.3	2842	78.4
LinTag	Work-only	-2320.2	3999	76.3

Table 4.1: Our reinforcement learning agent captures higher rewards than the baseline policies on all three live datasets. Asterisks indicate significantly higher rewards. Our agent produces datasets of higher accuracy at the expense of gathering fewer labels.



(a) LinWiki dataset. (b) Rajpal dataset. (c) LinTag dataset.

Figure 4.7: Scatter plots of worker accuracy vs. number of questions answered show that low accuracy workers leave on their own accord after a small number of questions in the Rajpal and LinTag datasets — hence all forms of worker testing have limited benefit in these scenarios. Adaptive testing has the most potential benefit in the LinWiki dataset.

Inspecting the distribution of worker accuracies and number of questions answered by each worker helps to explain these results. As shown in Figure 4.7, some low-accuracy workers answer a large number of questions in the LinWiki dataset; POMDP-RL produces large gains by adaptively testing and filtering these workers. In contrast, the lower quality workers in the Rajpal and LinTag datasets tend to leave on their own accord after a small number of questions, reducing the possible benefit of testing them in order to fire poor performers. The LinTag dataset has a small fraction of workers above the desired accuracy, and none of the methods were able to produce a dataset with the desired accuracy.

Examining the action traces and overall statistics on the number of times the POMDP-RL agent took test, work, and boot actions gives some insight into how the agent is able to improve on the static Test-and-boot baseline. As the POMDP-RL agent transitions from the base (static) exploration policy to the learned adaptive policy, the mean number of test actions per worker stays constant or decreases slightly,

but the mean number of work actions per worker decreases and the agent boots more frequently. This suggests that the agent becomes more conservative with its work actions, thus increasing accuracy.

4.4 Extensions to Teaching

I have extended our model to handle instruction in addition to testing. This extended model chooses between (1) teaching a worker to improve proficiency, (2) testing to measure worker accuracy, and (3) getting work done, all before the worker quits the system.

4.4.1 The Teaching Model

We assume that the requester has an unlimited number of gold questions, $\mathcal{Q}_{\text{gold}}$, with known answers, as well as questions with unknown answers, $\mathcal{Q}$. Intuitively, our objective is to answer the questions in $\mathcal{Q}$ with maximal accuracy, given a limited labor budget.

The task is defined by a set $\mathcal{R}$ of guidelines (rules) specified by the requester. Each question q requires that a worker know a subset of rules, $\mathcal{R}_q$, to answer it correctly. In general, we don't know a priori which questions require which rules, but we do know the subset of rules required for each gold question $q \in \mathcal{Q}_{\text{gold}}$.

We assume we know the prior probability, $P(r)$, of a rule r being required for an unknown question in $\mathcal{Q}$. This quantity can be estimated by computing the fraction of questions in $\mathcal{Q}_{\text{gold}}$ which require the rule (assuming $\mathcal{Q}_{\text{gold}}$ represents the overall distribution), or by sampling a small set of questions from $\mathcal{Q}$.

Following the *knowledge tracing* literature [31], we assume that worker w will answer a question correctly with probability $(1 - P_{\text{slip}})$ if s/he possesses the required skills and with probability P_{guess} otherwise. We represent worker w using a latent boolean vector of skills γ_w, where $\gamma_{w,r} = 1$ iff the worker knows how to apply rule r to questions. Assuming that each rule $r \in \mathcal{R}$ occurs with independent probability $P(r)$,

the probability worker w answers an unknown question correctly is

$$P(\text{correct} \mid w) = P_{\text{guess}} \cdot (1 - z) + (1 - P_{\text{slip}}) \cdot z, \tag{4.1}$$

where $z = \prod_{r \in \mathcal{R}} 1 - P(r) \cdot (1 - \gamma_{w,r})$ is the probability that the worker possesses all the required skills. The probability a worker answers a gold question correctly follows the same equation, only we know $P(r) = 1, \forall r \in \mathcal{R}_q$, and 0 otherwise. Over time, we learn a prior distribution over γ_w for unseen workers. At any time step, a worker may leave the system with probability P_{leave}.

4.4.2 The Decision Problem

At each time step, the system needs to decide whether to *teach* or *test* the worker (and if so which rule, r, to use), or whether to *ask* the worker to answer an unknown question $q \in \mathcal{Q}$.

We model the problem of making this choice again as a POMDP:

- The state space $\mathcal{S}$ can be factored as $< \Gamma_w, S_a >$, where Γ_w is a vector of boolean variables corresponding to the latent skill vector γ_w and S_a is a variable of size $|\mathcal{R}| + 1$ specifying the last rule tested (or none).[7] Additionally, $\mathcal{S}$ contains a special "submit" state.

- The set of actions, $\mathcal{A}$, consists of four actions: ask the worker an unknown question, select a rule to test (by asking a gold question that requires that rule), teach the worker a rule (we assume one or more pieces of instructional text associated with each rule),[8] and reject a worker (if the system deems him/her not worth employing). The current system may only teach immediately following a

[7] An additional variable specifying the last action is needed since our observation function does not include the previous state.

[8] A simple instructional action consisting of restating the rule along with the gold answer can be generated automatically.

test action. The sequence of a test action followed a teach action corresponds to the *elicit* teaching action used in intelligent tutoring systems [26].[9] Finally, we assume that an external process selects questions from $\mathcal{Q}$ or $\mathcal{Q}_{\text{gold}}$.

- The transition function $P(s' \mid s, a)$ specifies a probability distribution over outcome states supposing that the system executes $a \in \mathcal{A}$ while in $s \in \mathcal{S}$.
  - Reject worker: Enter the submit state with probability 1.
  - Select a rule to test: Worker quits with probability P_{leave}. Otherwise, the worker answers and loses knowledge of each rule independently with probability P_{lose}. The state variable S_a is set to the rule tested.
  - Teach the rule that was just tested: The worker gains knowledge of the rule taught with probability P_{gain}. The state variable S_a is reset.
  - Ask an unknown question: Worker quits with probability P_{leave}. Otherwise, the worker answers and loses knowledge of each rule independently with probability P_{lose}.

- Observation function: The system can only directly observe the worker's answers to ask and test questions. The accuracy of the worker's answer is a function of which rules s/he knows and which are likely to be required by the question, as given in Equation 4.1. Additionally, we observe when workers leave.

- Reward function: The agent incurs a cost of c for each ask or test action (indistinguishable to the worker) and c_t for each teaching action.[10] The reward for asking question $q \in \mathcal{Q}$ is $\lambda f(q)$, where

$$f(q) = \mathrm{E}\left[\max_{z \in Z_q} P(Z_q = z \mid A_{q,w})\right] - \max_{z \in Z_q} P(Z_q = z)$$

[9] An alternative teaching action is to *tell* the worker the answer immediately, but this provides less information about worker knowledge.

[10] These labor costs can be computed as part of reinforcement learning, by estimating the average observed time to complete each action and multiplying by a fair wage.

is the expected accuracy gain produced by incorporating the new information provided by the worker into the posterior predictions and λ is a parameter specifying the utility of information. In the above equation, Z_q is the random variable for the latent answer to question q, and $A_{q,w}$ is the random variable for worker w's response to q. The expectation is taken over possible worker responses, where $P(A_{q,w} = z \mid Z_q = z)$ is determined by Equation 4.1.

We chose not to include latent answers to questions in the state space in the current work so that the state space does not grow with the number of questions. Moreover, including answers in the state space is unnecessary since we focus on maximizing the benefit provided by each worker separately. In expectation, the model we define is equivalent to a model with a state space that includes the latent answers, and which has a $2^{|\mathcal{Q}|}$ possible submit actions corresponding to submitting an answer for each question. Prior work on decision theoretic crowdsourcing [100, 101] has used such a model when $|\mathcal{Q}| = 1$; our setting is more challenging since we must consider the answers to many tasks simultaneously.

4.4.3 Experiments

We conducted experiments using synthetic data under a range of hand-estimated parameter configurations to demonstrate the potential benefits of our approach, which are twofold. First, learning a policy using our POMDP approach saves costly A-B testing to find the best fixed policy. Second, in some settings, the learned adaptive policy outperforms reasonable baseline policies, meaning that even extensive A-B testing will not yield a policy that is as good.

We assume in these experiments that worker skills are independent and thus the initial belief state probabilities are $P(\Gamma_w, S_a) = \prod_{r \in \mathcal{R}} P(\Gamma_{w,r})$ for states where S_a indicates that no rule has just been tested, and 0 otherwise. Simulations sample workers from this prior. Additionally, we assume binary multiple choice questions

and unskewed data ($P(Z_q) = 0.5, \forall q \in \mathcal{Q}$), and set $c = c_t = 0.1$ and $\lambda = 1$. We used the ZMDP POMDP solver[11] with default configuration settings, maximum solve time of 1 minute[12], and discount factor of 0.99.

Our preliminary experiments compare to baseline policies that teach each rule k times during an initial tutorial phase. Our experiments assume that $P(r)$ and $P(\gamma_{w,r})$ are the same $\forall r \in \mathcal{R}$, since these configurations are most favorable to the baseline policies.

Planning

Our first set of experiments assumes that the model parameters are known.

We experimentally obtain basic policies under a range of parameter settings. For instance, if workers already know the rules with high probability ($P(\gamma_w)$ is high) or the rules are infrequently needed to answer unknown questions ($P(r)$ is low), our system learns to begin asking unknown questions immediately (no teaching). The system also limits teaching if workers tend to leave quickly (P_{leave} is high). On the other hand, if workers need instruction but fail to respond to teaching ($P(\gamma_w)$ and P_{learn} are low), our system rejects workers immediately.

In addition to finding these policies, our system automatically tunes the amount of instruction. The POMDP policy earns at least as much reward as the best baseline policies in all the configurations tested, unsurprising since the parameter values are known in these tests. Additionally, the adaptive policies typically require significantly fewer teaching actions than baselines since they can better estimate when a worker knows a rule. Using fewer gold questions reduces the burden of creating gold questions for each rule as well as the risk that shared answers to gold questions will enable cheating.

---

[11] `https://github.com/trey0/zmdp`

[12] Experiments were run on 6-core 2.4 GhZ processors.

Reinforcement Learning

When our system is first deployed in a new crowdsourcing environment, it must also learn the POMDP model parameters. This necessitates an exploration-exploitation tradeoff. Preliminary experiments show that our system is able to learn the model using a simple epsilon-greedy strategy.[13] For episode e (the eth worker hired), we select actions randomly with probability $1/e$. We reestimate model parameters and replan prior to each episode. To estimate parameters, we treat the POMDP as an input output hidden Markov model [11] and use the Baum-Welch (EM) algorithm initialized with parameters from the previous episode. We use a default prior of Beta(1.1, 1.1)[14] on parameters, but introduce a weak bias toward 0 (Beta(2, 5)) to match our intuition that P_{slip}, P_{lose}, and $P(\gamma_w)$ tend to be smaller than 0.5.

Figure 4.8 shows the learning performance given two rules and a set of plausible parameters ($P_{\text{leave}} = 0.01$, $P_{\text{learn}} = 0.4$, $P_{\text{lose}} = 0.05$, $P_{\text{slip}} = 0.1$, $P_{\text{guess}} = 0.5$, $P(\gamma_w) = 0.2$, $P(r) = 0.5$). "POMDP (known)," the model given true parameters, obtains 4.6 times as much reward as the best baseline policy in our space of baselines (in this case, teach each rule twice). Figure 4.9 shows a sample execution trace of this model. After having seen 100 workers, the reinforcement learning agent accumulates 2.8 times as much reward as the best baseline policy. While one could make the space of baseline policies arbitrarily more complex (e.g., by introducing an additional parameter controlling intervals between teaching sessions), the POMDP approach is simple to specify, does not require extensive A-B testing, and can perform significantly better even than the best candidate baseline.

[13] We also tried a Thompson sampling-based strategy, which performed worse due to high variance in the models and policies.

[14] This prior gives initial parameter estimates of 0.5.

Figure 4.8: Cumulative rewards for a simple (two rule) problem, averaged over 1000 simulations and shown with 95% confidence bands. "POMDP (known)" uses the true parameters, "POMDP (learned)" reestimates parameters each episode, and the best fixed policy is the baseline policy that teaches each rule twice. Each episode corresponds to hiring a new worker.

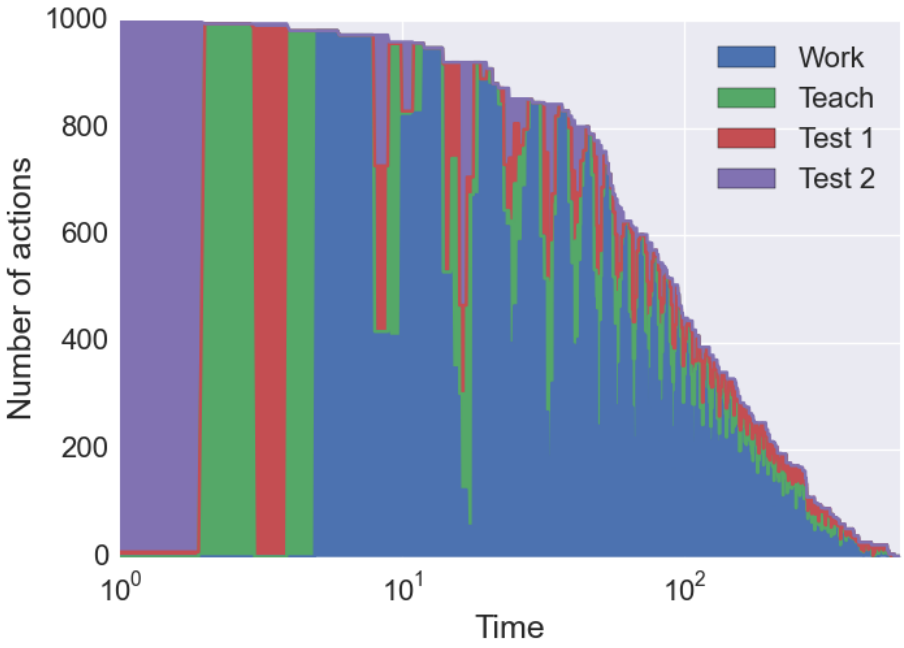


Figure 4.9: Sample execution trace for the "POMDP (known)" model in Figure 4.8. The number of actions (Y axis) is summed over 1000 workers and drops over time as workers abandon the task. Note that the system executes different (adaptive) teaching sequences depending on test results. Work corresponds to asking an unknown question and Test 1 and Test 2 correspond to testing rules 1 and 2, respectively.

4.5 Discussion and Future Work

In order to distinguish between high-quality and error-prone workers, requesters on crowdsourcing platforms, such as Amazon Mechanical Turk, routinely reserve 10–30% of their workload for "gold" test questions, whose answers are known, and then dismiss workers who fail a disproportionate percentage. This type of static policy is arbitrary and wastes valuable budget. The exact percentage of test questions and dismissal rate is often chosen with little experimentation, and, more importantly, it does not adapt to individual workers, the current mixture of skillful vs. unskillful workers, nor the number of tasks workers perform before quitting. Intuitively, the percentage of test questions should be high if a large percentage of the workers are spammers. Furthermore, once one is very certain that a worker is diligent, one can likely decrease the testing frequency.

To develop a principled solution to the problem of balancing between (1) testing workers to determine their accuracy, and (2) actually getting work performed by good workers, we formulate it as a partially-observable Markov decision process (POMDP). Our worker model captures the possibility that worker performance may degrade over time (whether due to fatigue, boredom, or deceit) and workers may leave our system after any question. Our model also takes as input a minimum desired accuracy of the final output, and a base testing policy. We apply reinforcement learning over the POMDP to dynamically improve the given base policy with experience. We comprehensively test our model using simulation experiments and show that our algorithm is robust to variations in desired output accuracy, worker mix, and other parameters. Additional experiments on three live data sets (collected using Amazon Mechanical Turk) show that our agent performs up to 111% better than common policies found in the literature. We also formulate a version of our agent that can *teach* workers. Importantly, our software is fully automated, easy to apply, runs mostly out of the box, and is made available for further use by the research community.

We note that our method has several limitations. For example, gold test questions are only applicable for crowd work with objective answers. Neither our testing approach nor methods based on expectation maximization will work for subjective questions or descriptive tasks where correct answers can be highly variable. However, we think our POMDP model could be extended to schedule *separate* validation jobs from an independent set of crowd workers.

Another possible direction for future work is to adopt more sophisticated models of worker learning and engagement. In order to facilitate learning parameters from a small amount of data, we assume that a worker learns or forgets any rule with the same probability. However, in some settings, rules may require different amounts of instruction. Additionally, modeling individual learning rates [99] or individual probabilities of leaving [117] may further improve performance. Behavioral traces may also be useful for predicting disengaged workers [145], signaling the need for interventions with tests or warnings.

Managing workers is a critical function performed by a self-improving crowdsourcing agent. The methods in this chapter enable such an agent to improve without any explicit guidance from the requester (other than by providing an optional initial base policy) or meta-workers; the agent learns from interacting with workers performing the task. This work is a first step toward our vision of an agent with a comprehensive set of worker management actions that can maximize the benefit of individual worker contributions and overall answer quality.

Chapter 5
ROUTING TASKS TO ALL AVAILABLE WORKERS

The previous two chapters described methods for recruiting workers (Chapter 3) and managing individual workers by balancing the amount of testing and training they receive (Chapter 4). Once they are in the system, determining which specific questions workers answer—for example, by matching workers with skill-appropriate tasks—can also have a major impact on crowdsourcing outcomes; it is critical that a self-improving crowdsourcing agent be able to perform this function well.

This chapter presents algorithms for routing tasks to all available workers *in parallel*, a problem that has not been addressed by prior work. Requesters could attempt to craft heuristic rules for matching workers and tasks, but this chapter demonstrates that algorithms outperform ad-hoc approaches (and may require less effort from the requester). Due to the combinatorial nature of the action space, we formulate routing as a tractable submodular optimization problem and provide further extensions that improve scalability for large numbers of workers and questions. While we do not explicitly investigate the extent to which these algorithms can improve themselves over time in this chapter, our agent can self-improve by making use of methods to compute more accurate worker accuracy estimates [159] as the task proceeds.

5.1 Introduction

While there are millions of workers present and thousands of tasks available on crowdsourcing platforms today, the problem of effectively matching the two, known as *task routing*, remains a critical open question. Law & von Ahn [97] describe two modes of solving this problem – the pull and the push modes. In the pull mode, such

as on Amazon Mechanical Turk (AMT), workers themselves select tasks based on price, keywords, etc. In contrast, a push-oriented labor market, popular on volunteer crowdsourcing platforms (*e.g.*, Zooniverse [108]), directly allocates appropriate tasks to workers as they arrive. Increasing amounts of historical data about tasks and workers create the potential to greatly improve this assignment process. For example, a crowdsourcing system might give easy tasks to novice workers and route difficult problems to experts.

Unfortunately, this potential is hard to realize. Existing task routing algorithms (*e.g.*, [23, 44, 67, 161]) make too restrictive assumptions to be applied to realistic crowdsourcing platforms. For example, they typically assume at least one of the following simplifications: (1) tasks can be allocated purely sequentially, (2) workers are willing to wait patiently for a task to be assigned, or (3) the quality of a worker's output can be evaluated instantaneously. In contrast, an ideal practical task router should be completely unsupervised, since labeling gold data is expensive. Furthermore, it must assign tasks *in parallel to all available workers*, since making engaged workers wait for assignments leads to an inefficient platform and frustrated workers. Finally, the task router should operate in real-time so that workers need not wait. The two latter aspects are especially important in citizen science and other forms of volunteer (non-economically motivated) crowdsourcing.

In this chapter we investigate algorithms for task routing that satisfy these desiderata. We study a setting, called JoCr (**Jo**int **Cr**owdsourcing [90]), in which the tasks are questions, possibly with varying difficulties. We assume that the router has access to a (possibly changing) set of workers with different levels of skill. The system keeps each available worker busy as long as the worker stays online by assigning him or her questions from the pool. It observes responses from the workers that complete their tasks and can aggregate their votes using majority vote or more sophisticated Bayesian methods in order to estimate its confidence in answers to the assigned ques-

tions. The system's objective is to answer all questions from the pool as accurately as possible after a fixed number of allocations.

Even within this setting, various versions of the problem are possible depending upon the amount of information available to the system (Figure 5.1). In this chapter we focus on the case where task difficulties and worker abilities are known a priori. Not only does this setting lay a foundation for the other cases, but it is of immediate practical use. Recently developed community-based aggregation can learn very accurate estimates of worker accuracy from limited interactions [159]. Also, for a wide range of problems, there are domain-specific features that allow a system to roughly predict the difficulty of a task.

We develop algorithms for both offline and adaptive JoCr task routing problems. For the offline setting, we first prove that the problem is NP hard. Our approximation algorithm, $\text{JUGGLER}_{\text{OFF}}$, is based on the realization that a natural objective function for this problem, expected information gain, is submodular. This allows $\text{JUGGLER}_{\text{OFF}}$ to efficiently obtain a near-optimal allocation. We use similar ideas to devise an algorithm called $\text{JUGGLER}_{\text{AD}}$ for the adaptive problem, which achieves even better empirical performance. $\text{JUGGLER}_{\text{BIN}}$ is an approximate version of $\text{JUGGLER}_{\text{AD}}$, which is able to scale to large numbers of workers and tasks in an efficient manner. Finally, we present $\text{JUGGLER}_{\text{RT}}$, which handles workers who take different amounts of time to complete tasks.

We test our algorithms both in simulation and using real data. Experiments with live oDesk workers on a natural language named entity linking task show that $\text{JUGGLER}_{\text{AD}}$'s allocation approach achieves the same accuracy as the commonly used round-robin strategy, using only 48% of the labor. Additional experiments on simulated data show that (1) our adaptive method significantly outperforms offline routing, (2) the adaptivity's savings grow when worker skills and task difficulties are more varied, (3) our binned (approximate) adaptive algorithm provides scalability

without compromising performance, and (4) our approach yields significant savings even when worker response times vary.

5.2 Problem Definition

In our JoCr model, we posit a set of workers $\mathcal{W}$ and a set of questions $\mathcal{Q}$. Each question $q \in \mathcal{Q}$ has a (possibly latent) *difficulty* $d_q \in [0, 1]$, and each worker $w \in \mathcal{W}$ has a (possibly latent) *skill* parameter $\gamma_w \in (0, \infty)$. While more complex models are possible, ours has the advantage that it is learnable even with small amounts of data. We assume that the probability of a worker w providing a correct answer to a question q is a monotonically increasing function $P(d_q, \gamma_w)$ of worker skill and a monotonically decreasing function of question difficulty. For example, one possible such function, adapted from [39], is

$$P(d_q, \gamma_w) = \frac{1}{2}(1 + (1 - d_q)^{\frac{1}{\gamma_w}}). \tag{5.1}$$

We assume that JoCr algorithms will have access to a pool of workers, and will be asking workers from the pool to provide answers to the set of questions submitted by a requester. This pool is a subset of $\mathcal{W}$ consisting of workers available at a given time. The pool need not be static — workers may come and go as they please. To formalize the problem, however, we assume that workers disappear or reappear at regularly spaced points in time, separated by a duration equal to the time it takes a worker to answer a question. We call each such a decision point a *round*. At the beginning of every round t, our algorithms will assign questions to the pool $P_t \subseteq \mathcal{W}$ of workers available for that round, and collect their responses at the round's end.

The algorithms will attempt to assign questions to workers so as to maximize some utility over a fixed number of rounds, a time *horizon*, denoted as T. Since we focus on volunteer crowdsourcing platforms, in our model asking a worker a question incurs no monetary cost. To keep workers engaged and get the most questions answered, our techniques assign a question to every worker in pool P_t in every round $t = 1, \ldots, T$.

An alternative model, useful on a *shared* citizen-science platform would allow the routing algorithm a fixed budget of n worker requests over an arbitrary horizon.

5.2.1 Optimization Criteria

The JoCr framework allows the use of various utility functions $U(S)$ to optimize for, where $S \in 2^{\mathcal{Q}\times\mathcal{W}}$ denotes an assignment of questions to workers. One natural utility function choice is the expected *information gain* from observing a set of worker responses. Specifically, let $\mathcal{A} = \{A_1, A_2, \ldots, A_{|\mathcal{Q}|}\}$ denote the set of random variables over correct answers to questions, and let $\mathcal{X} = \{X_{q,w} \mid q \in \mathcal{Q} \wedge w \in \mathcal{W}\}$ be the set of random variables corresponding to possible worker responses. Further, let $\mathcal{X}_S$ denote the subset of $\mathcal{X}$ corresponding to assignment S. We can quantify the uncertainty in our predictions for $\mathcal{A}$ using the joint entropy

$$H(\mathcal{A}) = -\sum_{\mathbf{a}\in \mathrm{dom}\mathcal{A}} P(\mathbf{a}) \log P(\mathbf{a}),$$

where the domain of $\mathcal{A}$ consists of all possible assignments to the variables in $\mathcal{A}$, and $\mathbf{a}$ denotes a vector representing one assignment. The conditional entropy

$$H(\mathcal{A} \mid \mathcal{X}_S) = -\sum_{\substack{\mathbf{a}\,\in\,\mathrm{dom}\mathcal{A},\\ \mathbf{x}\,\in\,\mathrm{dom}\mathcal{X}_S}} P(\mathbf{a},\mathbf{x}) \log P(\mathbf{a} \mid \mathbf{x})$$

represents the expected uncertainty in the predictions after observing worker votes corresponding to the variables in $\mathcal{X}_S$. Following earlier work [164], we assume that questions are independent (assuming known parameters) and that worker votes are independent given the true answer to a question, *i.e.*,

$$P(\mathbf{a},\mathbf{x}) = \prod_{q\in\mathcal{Q}} P(a_q) \prod_{w\in\mathcal{W}\text{ who answered }q} P(x_{q,w} \mid a_q),$$

where $P(x_{q,w} \mid a_q)$ depends on worker skill and question difficulty as in Equation 5.1. We can now define the value of a task-to-worker assignment as the expected reduction in entropy

$$U_{IG}(S) = H(\mathcal{A}) - H(\mathcal{A} \mid \mathcal{X}_S). \tag{5.2}$$

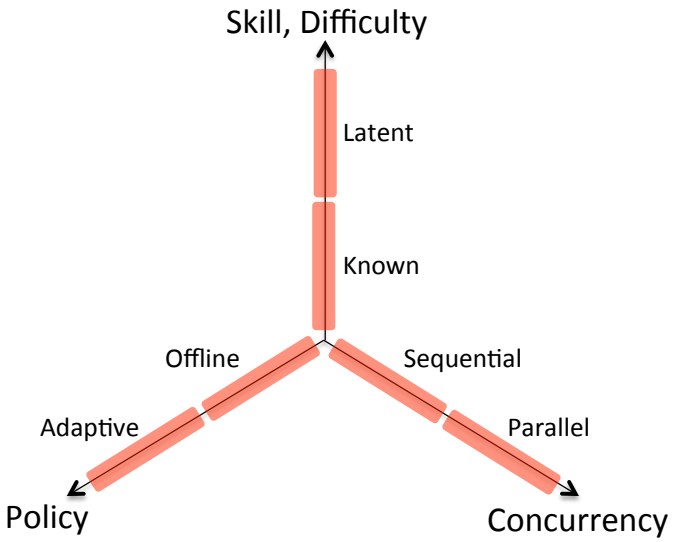


Figure 5.1: The space of allocation problems.

5.2.2 Problem Dimensions

Given the information gain maximization objective, we consider the problem of finding such an assignment under combinations of assumptions lying along three dimensions (Figure 5.1):

- **Offline vs. online (adaptive) assignment construction.** In the offline (or *static*) mode, a complete assignment of T questions to each worker is chosen once, before the beginning of the first round, and is not changed as workers provide their responses. Offline assignment construction is the only available option when workers' responses cannot be collected in real time. While we provide an algorithm for the offline case, JUGGLER$_{\mathrm{OFF}}$, it is not our main focus. Indeed, worker responses typically *are* available immediately after workers provide them, and taking them into account allows the router to resolve worker disagreements on the fly. Our experimental results confirm this, with adaptive JUGGLER$_{\mathrm{AD}}$ outperforming the offline algorithm as well as our baselines.

- **Sequential vs. parallel hiring of workers.** A central question in online allocation is the number of workers to hire in a given round. If one could only hire n workers in total, an optimal strategy would ask workers sequentially, one worker per round, because this allows the maximum use of information when building the rest of the assignment. However, this strategy is impractical in most contexts, especially in citizen science — using just a single worker wastes the time of the majority and demotivates them. Therefore, in this chapter we consider the harder case of parallel allocation of questions to a *set* of workers. In our case, this set comprises all workers available at the moment of allocation.

- **Known vs. latent worker skill and question difficulty.** Worker skill and question difficulty are critical for matching tasks with workers. Intuitively, the optimal way to use skillful workers is to give them the most difficult tasks, leaving easier ones to the less proficient members of the pool. In this chapter, we assume these quantities are known — a reasonable assumption in practice. Since workers and requesters often develop a long-term relationships, EM-based learning [163, 164] can estimate worker accuracy from a surprisingly short record using community-based aggregation [159].

  Similarly, domain-specific features often allow a system to estimate a task's difficulty. For example, the linguistic problem of determining whether two nouns corefer is typically easy in the case of apposition, but harder when a pronoun is distant from the target [135, 155]. In the citizen science domain, detecting planet transits (the focus of the Zooniverse project Planet Hunters) is more difficult for planets that are fast-moving, small, or in front of large stars [116].

  If skill or difficulty is *not* known, the router faces a challenging exploration-exploitation tradeoff. For example, it gains information about a worker's skill by asking a question for which it is confident of the answer, but of course this precludes asking a question whose answer it does not know.

5.3 Offline Allocation

We first address the simplest form of the allocation problem, since its solution (and complexity) forms a basis for subsequent, more powerful algorithms. Suppose that an agent is able to assign tasks to a pool of workers, but unable to observe worker responses until they have all been received and hence has no way to reallocate questions based on worker disagreement. We assume that the agent has at its disposal an estimate of question difficulties and worker skills, as discussed above.

Theorem 2. *Finding an assignment of workers to questions that maximizes an arbitrary utility function $U(S)$ is NP-hard.*

Proof sketch. We prove the result via a reduction from the Partition Problem to offline JoCr. An instance of this problem is specified by a set X containing n elements and a function $s : X \to \mathbb{Z}^+$. The Partition Problem asks whether we can divide X into two subsets X_1 and X_2 s.t. $\sum_{x \in X_1} s(x) = \sum_{x \in X_2} s(x)$.

We construct an instance of offline JoCr to solve an instance of the Partition Problem as follows. For each element $x_i \in X$, we define a corresponding worker $w_i \in W$ with skill $\gamma_i = s(x_i)/\max_{x_i \in X} s(x_i)$, and let the horizon equal 1. We also define two questions, q_1 and q_2, with the same difficulty d. Let $S_{q,w}$ be an indicator variable that takes a value of 1 iff worker w has been assigned question q. Let $f(S, q) = \log\left[\sum_w S_{q,w}\gamma_w + 1\right]$ and define the utility function as $U(S) = \sum_q f(S, q)$.

The solution to a Partition Problem instance is *true* iff $f(S, q_1) = f(S, q_2)$ for an optimal solution S to the corresponding JoCr problem constructed above.

$\Rightarrow$: Suppose that there exist subsets of X, X_1 and X_2, such that the sum of elements in the two sets are equal. Further, suppose for contradiction that the optimal solution S from JoCr assigns sets of workers with *different* sums of skills to work on the two questions. Then we can improve $U(S)$ by making sum of skills equal (by monotonicity and submodularity of logarithm), implying that S was suboptimal, leading to a contradiction.

$\Leftarrow$: Suppose that S is an optimal solution to the JoCr problem s.t. $f(S, q_1) = f(S, q_2)$. Then sums of worker skills (multiplied by the maximum worker skill) for each question are equal and the solution to the Partition Problem is *true*. □

Given the intractability of this problem, we next devise approximation methods. Before proceeding further, we review the combinatorial concept of submodularity. A function $f : 2^{\mathcal{N}} \rightarrow R$ is *submodular* if for every $A \subseteq B \subseteq \mathcal{N}$ and $e \in \mathcal{N}$: $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$. Further, f is *monotone* if for every $A \subseteq B \subseteq \mathcal{N}$: $f(A) \leq f(B)$. Intuitively, this is a diminishing returns property; a function is submodular if the marginal value of adding a particular element never increases as other elements are added to a set.

Theorem 3. *The utility function $U_{IG}(S)$ based on value of information defined in the previous section is monotone submodular in S, provided that worker skill and question difficulty are known.*

Proof sketch. Given worker skill, question difficulty, and the true answers, worker ballots are independent. We can use the methods of [92] to show that expected information gain $U_{IG}(S)$ is submodular and nondecreasing. □

Since the utility function $U_{IG}(S)$ is monotone submodular, we naturally turn to approximation algorithms. Our optimization problem is constrained by the horizon T, meaning that we may not assign more than T questions to any worker. We encode this constraint as a *partition matroid*. Matroids capture the notion of linear independence in vector spaces and are specified as $M = (\mathcal{N}, \mathcal{I})$, where $\mathcal{N}$ is the *ground set* and $\mathcal{I} \in 2^{\mathcal{N}}$ are the subsets of elements in $\mathcal{N}$ that are independent in the matroid. A partition matroid is specified with an additional constraint. If we partition $\mathcal{N} = \mathcal{Q} \times \mathcal{W}$ into disjoint sets $B_w = \{w\} \times \mathcal{Q}$ corresponding to the set of possible assignments for each worker w, we can construct the desired partition

Algorithm 2 The $\text{JUGGLER}_{\text{OFF}}$ algorithm

Input: Workers $\mathcal{W}$, prior $P(\mathbf{a})$ over answers, unobserved votes $\mathcal{X}_R$, horizon T, initial assignment S

Output: Final assignment $\mathcal{S}$ of questions to workers

for w **in** sorted($\mathcal{W}$, key $= \gamma_w$) **do**
  for $i = 1$ **to** T **do**
    $\mathcal{X}_w \leftarrow \{X_{q,w'} \in \mathcal{X}_R \mid w' = w\}$
    for $X_{q,w} \in \mathcal{X}_w$ **do**
      $\Delta X \leftarrow H(A_q \mid \mathbf{x}) - H(A_q \mid X_{q,w}, \mathcal{S}, \mathbf{x})$
    end for
    $X^* \leftarrow \arg\max\{\Delta X : X \in \mathcal{X}_w\}$
    Set $\mathcal{S} \leftarrow \mathcal{S} \cup \{X^*\}$ and $\mathcal{X}_R \leftarrow \mathcal{X}_R \setminus \{X^*\}$
  end for
end for

matroid by defining an independent set to include no more than T elements from each of these sets.

$\text{JUGGLER}_{\text{OFF}}$ uses the greedy selection process shown in Algorithm 2, which is guaranteed to provide a 1/2-approximation for monotone submodular optimization with a matroid constraint [52], yielding the following theorem:

Theorem 4. *Let S^* be a question-to-worker assignment with the highest information gain U_{IG}, and let S be the assignment found by maximizing U_{IG} greedily. Then $U_{IG}(S) \geq \frac{1}{2} U_{IG}(S^*)$.*

$\text{JUGGLER}_{\text{OFF}}$ improves efficiency by making assignments separately for each worker (exploiting the fact that Equation 5.1 is monotonically increasing in worker skill) and

by using lazy submodular evaluation [91]. Interestingly, while the greedy algorithm dictates selecting workers in order of descending skill in Algorithm 2, we observe drastic improvement in our empirical results from selecting workers in reverse order; our implementation sorts by increasing worker skill, which prioritizes assigning easier questions to the less-skilled workers. We have simplified the utility computation of ΔX by observing that entropy decomposes by question in our model. We note that more involved algorithms can improve the theoretical guarantee to $(1 - 1/e)$ [8, 51, 160], but given the large positive effect of simply reversing the order of workers, it is unlikely that these methods will provide significant benefit.

Since we are motivated by the citizen science scenario, we have not included cost in the agent's utility function. However, we note that our utility function remains submodular (but not longer monotone) with the addition of a linear cost term. Thus, one can formulate the optimization problem as non-monotone submodular optimization, for which algorithms exist.

5.4 Adaptive (Online) Allocation

$\text{JUGGLER}_{\text{OFF}}$ performs a complete static allocation and does not adapt based on worker response. However, in most crowdsourcing platforms, we have access to intermediate output after each worker completes her attempt on a question; it makes sense to consider all previous responses during each round, when assigning tasks. Intuitively, the router may choose to allocate more workers to a question, even if easy, that has generated disagreement. In this section we describe our algorithms for the adaptive setting. We believe that these are the first adaptive task allocation algorithms for crowdsourcing to engage all available workers at each time step.

Unlike in the offline setting, the objective of an optimal algorithm for the online setting is to compute *an adaptive way of constructing assignments*, not any fixed assignment per se. Formally, this problem can be seen as a Partially Observable Markov Decision Process (POMDP). The state space is $\mathcal{A}$, the set all possible answers

Algorithm 3 The $\text{JUGGLER}_{\text{AD}}$ algorithm

Input: Workers $\mathcal{W}$, prior $P(\mathbf{a})$ over answers, unobserved votes $\mathcal{X}_R$, horizon T

$\mathbf{x} \leftarrow \emptyset$

for $t = 1$ **to** T **do**

$S_t \leftarrow$ call $\text{JUGGLER}_{\text{OFF}}(T = 1,\, S = \emptyset)$

Observe $\mathbf{x}_{S_t}$ and set $\mathbf{x} \leftarrow \mathbf{x} \cup \mathbf{x}_{S_t}$

$\mathcal{X}_R \leftarrow \mathcal{X}_R \setminus S_t$

Update prior as $P(\mathbf{a} \mid \mathbf{x})$

end for

to the questions $\mathcal{Q}$. The state of the POMDP never changes; the goal is to estimate this (hidden) state, which is $\mathbf{a}^*$, the vector of true answers to all the questions in $\mathcal{Q}$. In each round t, the available actions correspond to possible allocations $S_t \in 2^{\mathcal{Q} \times P_t}$ of tasks to all workers in the pool $P_t \subseteq \mathcal{W}$ s.t. each worker gets assigned only one task, the observations are a vector of worker responses $\mathcal{X}_{S_t}$, and the total planning horizon is T. The optimal solution to this POMDP is a policy π^* that satisfies, for each round t and the set of observations $\cup_{i=1}^{t-1} \mathbf{x}_i$ received in all rounds up to t,

$$\pi^*(\bigcup_{i=1}^{t} \mathbf{x}_i) = \text{argmax}_\pi \mathbb{E}\left[U(\bigcup_{j=t}^{H} \{\mathcal{S}_j \sim \pi(\bigcup_{i=1}^{t-1} \mathbf{x}_i, \bigcup_{i=t}^{j-1} \mathcal{X}_i))\} \right]$$

Existing methods for solving POMDPs apply almost exclusively to linear additive utility functions U and are intractable even in those cases, because the number of actions (allocations) in each round is exponential.

Although we can prove theoretical guarantees for the offline problem (and therefore for the problem of making an assignment in each round), it is more difficult to provide guarantees for the T-horizon adaptive allocation. A natural analysis approach is to use *adaptive submodularity* [60], which generalizes performance guarantees of the greedy algorithm for submodular optimization to adaptive planning. Intuitively, a

function f is adaptive submodular with respect to probability distribution p if the conditional expected marginal benefit of any fixed item does not increase as more items are selected and their states are observed. Unfortunately, we have the following negative result:

Theorem 5. *The utility function $U_{IG}(S)$ based on the value of information is not adaptive submodular even when question difficulties and worker abilities are known.*

Proof. In order for $U(S)$ to be adaptive submodular, the conditional expected marginal benefit of a worker response should never decrease upon making more observations. As a counterexample, suppose we have one binary question with difficulty $d = 0.5$ and two workers with skills $\gamma_1 = 1$ and $\gamma_2 \to \infty$. If the prior probability of the answer is $P(A = True) = 0.5$, the expected information gain $H(A) - H(A \mid X_2)$ of asking for a vote from the second worker is initially one bit, since she will always give the correct answer. However, if we observe a vote from the first worker before asking for a vote from the second worker, the expected information gain for the second worker will be less than one bit since the posterior probability $P(A = True)$ has changed. □

While we are unable to prove theoretical bounds on the quality of the adaptive policy in the absence of adaptive submodularity, our experiments in the next section show that JUGGLER$_{\mathrm{AD}}$ uses dramatically less human labor than commonly-used baseline algorithms.

5.4.1 Scalability

Assigning at Most One Worker at a Time to a Task

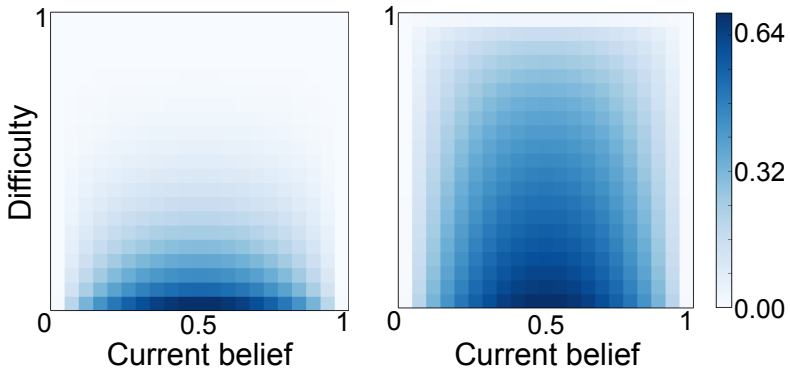
As an adaptive scheduler, JUGGLER$_{\mathrm{AD}}$ must perform allocations quickly for all available workers, so that workers

Binning Questions for Faster Performance

When scaling to large numbers of workers and questions, even a greedy strategy that never assigns more than one worker to a task at a time can be prohibitively expensive, as it requires making $O(|\mathcal{Q}|)$ utility computations per worker in each round. However, since the utility of an assignment is a function of the current belief about the true answer, as well as of the question difficulty and worker skill, we can reduce the complexity by partitioning assignments into bins.

$\text{JUGGLER}_{\text{BIN}}$ (Algorithm 4) is an approximate version of $\text{JUGGLER}_{\text{AD}}$, which makes uses of binning in order to reduce the cost of making an assignment to $O(|\mathcal{W}|^2 + |\mathcal{W}|C^2)$, where C is user-specified parameter representing the number of partitions. $\text{JUGGLER}_{\text{BIN}}$ uses C^2 bins representing the cross product of C question difficulty partitions and C belief partitions (evenly-spaced on the interval from 0 to 1). During initialization, the system sorts bins from highest to lowest utility for each worker (computed using the mean question difficulty and belief for the bin, assuming a uniform distribution of questions). For each worker, the system maintains a hash from each bin to a set of unanswered questions whose difficulty and current belief fall within the bin boundaries — and which are not currently being answered by another worker. The system makes an assignment by popping a question from the nonempty bin with highest utility. $\text{JUGGLER}_{\text{BIN}}$ approximates the performance of $\text{JUGGLER}_{\text{AD}}$ arbitrarily closely as the parameter C increases, and can scale independently of the total number of questions since it does not need to consider all questions in each round.

Note that the order in which $\text{JUGGLER}_{\text{BIN}}$ visits bins is not the same for all workers, and depends on a particular worker's skill. Figure 5.2 shows, for instance, that it may be more valuable to ask a high-skilled worker a difficult question we know little about (belief close to 0.5) than an easy question whose answer is quite certain (belief close to 0 or 1), but same statement might not hold for a low-skilled worker.



(a) Low-skilled worker (b) High-skilled worker

Figure 5.2: Visualization of the IG-based utility of asking questions of (a) a low-skilled worker ($\gamma = 0.71$) and (b) a high-skilled worker ($\gamma = 5.0$), as a function of question difficulty and current belief. Darker color indicates higher expected utility. (Bin-size corresponds to running $\text{JUGGLER}_{\text{BIN}}$ with $C = 21$.)

5.4.2 Variable Worker Response Times

Up to this point, we have assumed that each worker completes his assignment in each round. In practice, however, workers take different amounts of time to respond to questions, and a task router must either force some workers to wait until others finish or make new assignments to subsets of workers as they finish. $\text{JUGGLER}_{\text{RT}}$ described in Algorithm 5, performs the latter service by maintaining a record of outstanding assignments in order to route the most useful new tasks to available workers.

5.5 Experiments

We seek to answer the following questions: (1) How well does our adaptive approach perform in practice? We answer this question by asking oDesk workers to perform a challenging NLP task called *named entity linking* (NEL) [122]. (2) How much

benefit do we derive from an adaptive approach, compared to our offline allocation algorithm? We answer this question by looking at average performance over many experiments with simulated data. (3) How much does our adaptive approach benefit from variations in worker skill or problem difficulty? We answer this question through simulations using different distributions for skill and difficulty. (4) Finally, how well does our approach scale to many workers and tasks? We answer this question by first evaluating our strategy of limiting one worker per task per round, and then evaluating how speed gains from binning impact the quality of results.

5.5.1 Benchmarks

To evaluate the performance of our adaptive strategy, we compare its implementation in the information gain-based JUGGLER$_{\mathrm{AD}}$ to a series of increasingly powerful alternatives, some of which are also variants of JUGGLER:

- **Random (Ra).** The random strategy simply assigns each worker a random question in each round.

- **Round robin (RR).** The round robin strategy orders the questions by the number of votes they got so far, and in each round iteratively assigns each worker a random question with the fewest votes.

- **Uncertainty-based (Unc).** The uncertainty-based policy orders workers from the least to most skilled[1] and in each round iteratively matches the least skilled yet-unassigned worker to the unassigned question with the highest label uncertainty, measured by the entropy of the posterior distribution over the labels already received for that question. Note that this strategy differs from our implementation of the information gain-based JUGGLER$_{\mathrm{AD}}$ in just two ways: (a) it never assigns two workers to the same question in the same round and (b) it

[1]For consistency with other approaches (we did not observe a significant ordering effect).

ignores question difficulty when performing the assignment, thereby computing a difficulty-oblivious version of information gain. Thus, the uncertainty-based strategy can be viewed as a simpler version of JUGGLER$_{\text{AD}}$ and is expected to perform similarly to JUGGLER$_{\text{AD}}$ if all questions have roughly the same difficulty.

- **Accuracy gain-based (AG).** This policy is identical to the information gain-based JUGGLER$_{\text{AD}}$, but in every round greedily optimizes expected accuracy gain instead, i.e., seeks to find a single-round question-to-worker assignment S that maximizes

$$\mathbb{E}\left[\sum_{q\in\mathcal{Q}} \max_{a_q}(P(a_q \mid \mathcal{X}_S), 1 - P(a_q \mid \mathcal{X}_S))\right].$$

  Unlike information gain, accuracy gain is not submodular, so the worker allocation procedure that maximizes it greedily does not have a constant error bound even within a single allocation round. Nonetheless, intuitively it is a powerful heuristic that provides a reasonable alternative to information gain in the JUGGLER framework.

In the next subsections, we denote the information gain-based JUGGLER$_{\text{AD}}$ as IG for ease of comparison with the other benchmarks. While the random and round robin strategies a-priori look weaker than IG, the relative qualitative strength of the other two is not immediately clear and provides important insights into IG's practical operation.

In the experiments that follow, we initially force all benchmarks and IG itself to assign questions that have not yet been assigned, until each question has been asked once, before proceeding normally. This ensures that all predictions are founded on at least one observation and provides empirical benefits.

5.5.2 Comparison of Adaptive Strategies

In order to measure performance with real workers and tasks, we selected a binary named entity linking task [100, 107]. Since we wished to compare different policies on the same data, we controlled for variations in worker performance by coming up with a set of 198 questions and hiring 12 oDesk workers, each of whom completed the entire set.

In order to estimate worker skill and problem difficulty, we computed a maximum likelihood estimate of these quantities by running gradient descent using gold answers. Note that in general, these parameters can be estimated with little data by modeling worker communities and task features, as discussed previously.

Recall that in each round, each of our workers is assigned one of the 198 questions. After each round, for each policy we calculate the most likely answers to each question by running the same EM procedure on the worker responses. From the predictions, we can calculate the accuracy achieved by each policy after each round.

Figure 5.3 compares the performance of our adaptive policies, using the observed votes from the live experiment. Since simulating policy runs using all 12 workers would result in a deterministic procedure for IG and AG, we average the performance of many simulations that use 8 randomly chosen workers to produce a statistically significant result. One way to compare the performance of our policies is to compute the relative labor savings compared to the round robin policy. First, we compute a desired accuracy as a fraction of the maximum accuracy obtained by asking each worker to answer each question, and then compute the fraction of votes saved by running an adaptive policy compared to the round robin policy. Using this metric, all adaptive approaches achieve 95% of the total possible accuracy using fewer than 50% of the votes required by round robin to achieve that same accuracy.

In order to tease apart some of the relative differences between the adaptive policies, we also generated synthetic data for the 198 questions and 12 workers by ran-

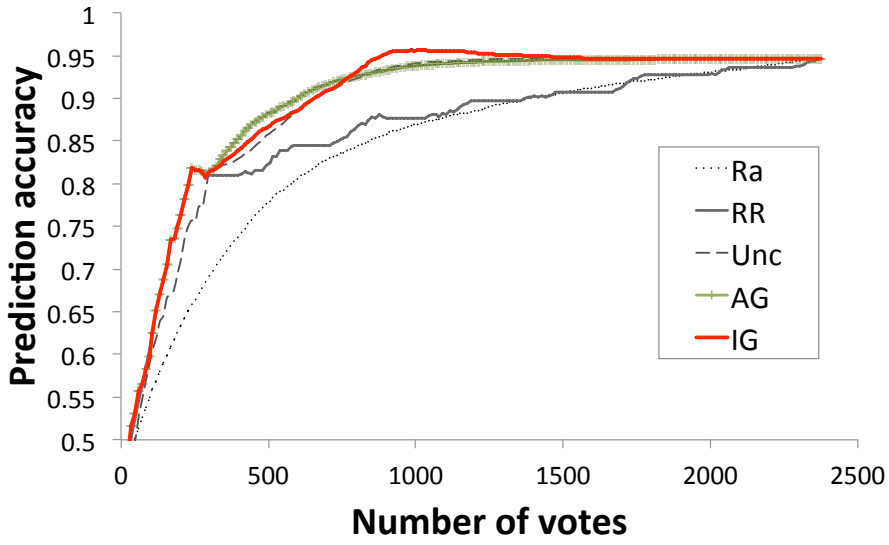


Figure 5.3: In live experiments, JUGGLER$_{\text{AD}}$ (IG) reaches 95% of the maximum attainable accuracy with only 48% of the labor employed by the commonly used round robin policy (RR).

domly sampling worker quality and question difficulty using the parameters we estimated from the live experiment. Figure 5.4 shows the results of this experiment. IG significantly outperforms AG, which in turn significantly outperforms Unc, in terms of fraction of votes saved compared to round robin in order to achieve 95% or 97% of the total possible accuracy ($p < 0.0001$ using two-tailed paired t-tests). These results demonstrate three important points:

- The largest savings are provided by JUGGLER$_{\text{AD}}$ (IG), followed by AG and Unc.

- All three adaptive algorithms significantly outperform the non-adaptive baselines.

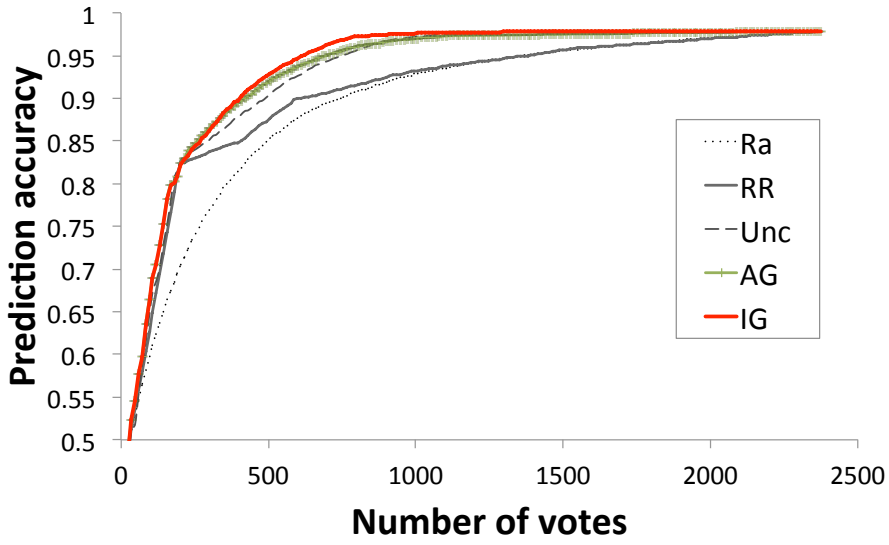


Figure 5.4: In experiments with simulated data using worker skills and question difficulties estimated from the live experiment, JUGGLER$_{\text{AD}}$ (IG) outperforms the accuracy gain (AG) and uncertainty-based (UNC) policies.

- Although the information gain-based JUGGLER$_{\text{AD}}$ "wins" overall, the other modifications of JUGGLER$_{\text{AD}}$ (AG and UNC) perform very well too, despite providing no theoretical guarantees.

We also compared these policies in the context of JUGGLER$_{\text{RT}}$, as shown in Figure 5.5. In order to simulate worker response times, we fit log-normal distributions to the observed response times and sampled from those distributions. Although the relative benefits are smaller, the ranking of policies remains the same as in the pure round-based setting, demonstrating the promise of our approach to generalize to the more realistic setting of variable worker response times.

Figure 5.6 summarizes the relative savings of our policies in each of these three scenarios outlined above—fixed votes, simulated votes, and simulated votes with variable response times. Since there is a clear advantage to IG across scenarios, in the follow-

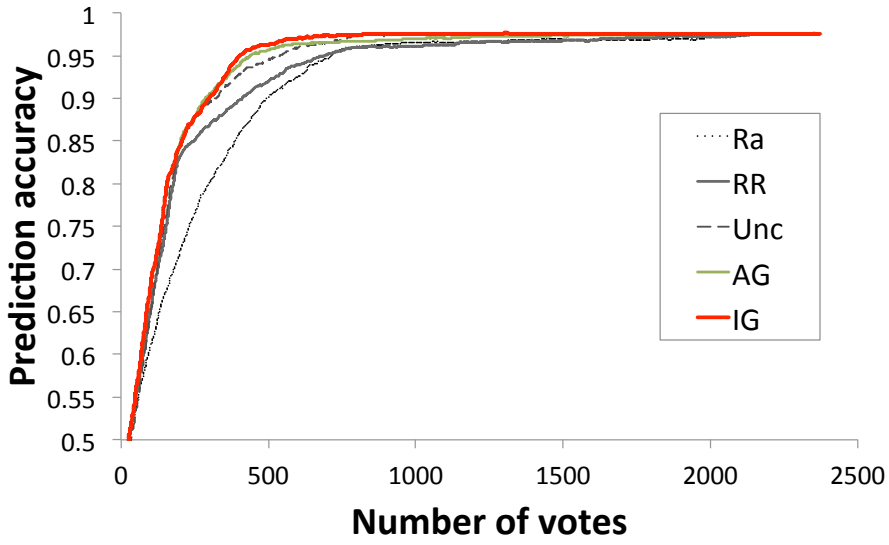


Figure 5.5: In live experiments with variable worker response times, JUGGLER$_{\mathrm{RT}}$ (IG) also outperforms the other policies, despite the need to make assignments in real time as workers complete tasks.

ing sections we conduct additional simulations to further characterize its performance and answer our remaining empirical research questions.

5.5.3 Benefit of Adaptivity

In order to determine the benefit of adaptivity, we generated synthetic data for 50 questions and 12 workers by randomly sampling worker quality and question difficulty.

Figure 5.7 shows the relative labor savings of JUGGLER$_{\mathrm{AD}}$ and JUGGLER$_{\mathrm{OFF}}$ compared to the round robin policy. The desired accuracy is again computed as a fraction of the maximum accuracy obtained by asking each worker to answer each question. Our simulation draws question difficulties uniformly and inverse worker skills $1/\gamma \sim \mathcal{N}(0.79, 0.29)$, a realistic distribution that fits estimates from the live

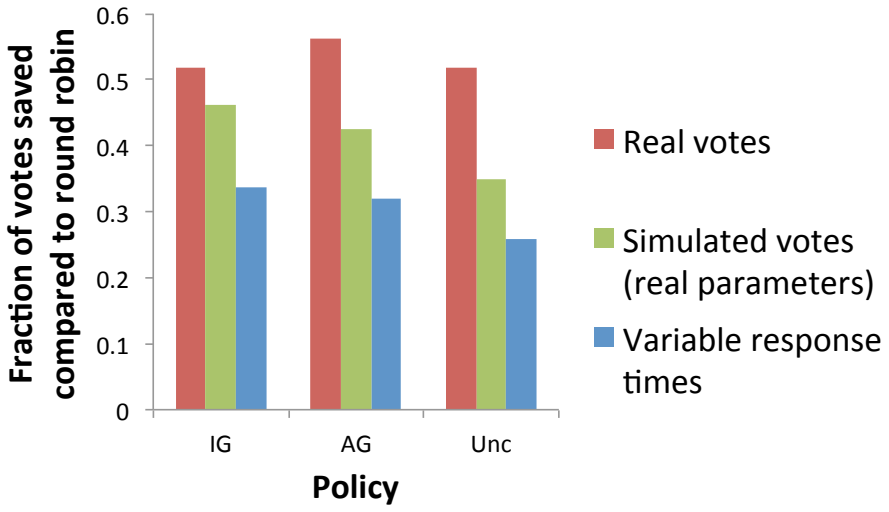


Figure 5.6: Summary of the fraction of savings of IG, AG, and UNC compared to the round robin policy for reaching an accuracy of 95% of the total achievable accuracy (from asking each worker each question).

experiment. JUGGLER$_{\text{AD}}$ saves significantly more labor than JUGGLER$_{\text{OFF}}$ underscoring the value of adaptive routing compared to offline.

5.5.4 Varying Worker Skills and Task Difficulties

We now investigate how various parameter distributions affect adaptive performance. Again, we generated synthetic data for 50 questions and 12 workers. Figure 5.8 shows that the relative savings of JUGGLER$_{\text{AD}}$ over round robin increases as the pool of workers becomes more diverse. We control the experiment by again sampling difficulties uniformly and using our earlier worker skill distribution, but varying the standard deviation from $\sigma = 0$ to $\sigma = 0.2$. Like before, accuracies are computed as a fraction of the total achievable accuracy given a set of workers. Our algorithms perform better for larger values of σ because they are able to exploit differences between workers to make the best assignments.

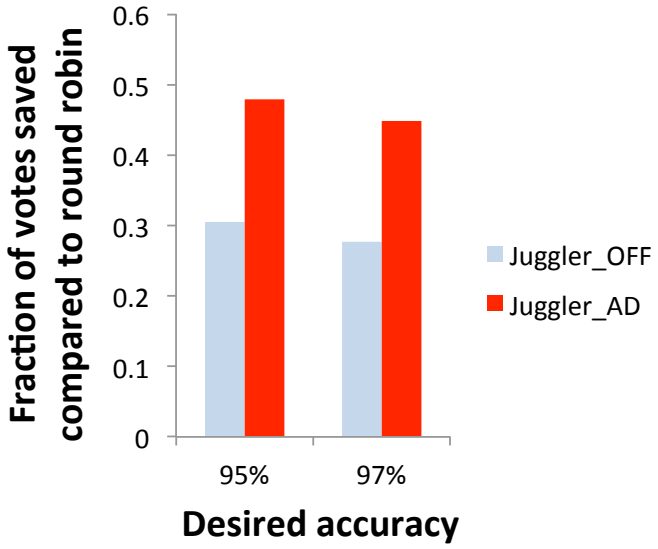


Figure 5.7: The "adaptivity gap." JUGGLER$_{\text{AD}}$ saves significantly more human labor than JUGGLER$_{\text{OFF}}$ by observing and responding to responses adaptively.

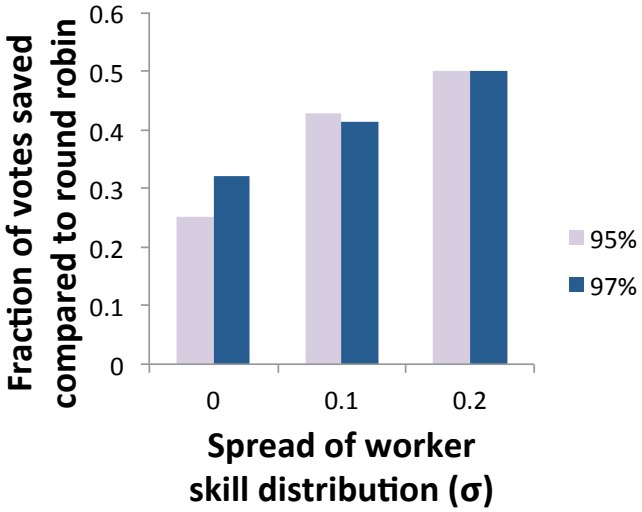


Figure 5.8: The benefit provided by JUGGLER$_{\text{AD}}$ increases as the pool of workers becomes more diverse.

Increasing the diversity of question difficulties while keeping the skill distribution fixed produces similar savings. For example, sampling difficulties from a Beta distribution with a single peak produces relative savings over round robin that are significantly less than sampling uniformly or from a bimodal distribution.

5.5.5 Scaling to Many Tasks and Workers

In order to empirically validate the speed and allocation effectiveness of our proposed scaling strategies, we experimented with data simulating a larger number of workers and questions (100 and 2000, respectively) using the same question difficulty and worker skill settings as in the earlier adaptivity experiment. Without imposing a limit on the number of simultaneously-operating workers per task, it is infeasible to run $\text{JUGGLER}_{\text{AD}}$ on problems of this size. We first checked to see if there was a drop in question-answering performance caused by limiting the number of workers assigned to a given task in each round. We found no statistically significant difference between limiting the number of workers per task to 1, 2, or 3 (at the 0.05-significance level using one-tailed paired t-tests to measure the fraction of votes saved to reach various thresholds). This result matches our experience running smaller experiments.

Restricting the number of workers per task to a small number reduces the scheduling time required by $\text{JUGGLER}_{\text{AD}}$ to 10–15 seconds[2] per round (scheduling 100 workers), but this is still significantly longer than the fraction-of-a-second scheduling times for our earlier experiments with 12 workers and 198 questions. $\text{JUGGLER}_{\text{BIN}}$, by contrast, is able to schedule workers in fewer than 3 seconds for any number of bins we tried ($C \in \{20, 40, 80, 160\}$). Indeed, $\text{JUGGLER}_{\text{BIN}}$ makes assignments using $O(|\mathcal{W}|^2)$ computations, which is drastically better than $O(|\mathcal{W}||\mathcal{Q}|)$ for the unbinned version when there are many questions. Although our runtime analysis for $\text{JUGGLER}_{\text{BIN}}$ includes an additional term on the order of $|\mathcal{W}|C^2$, in practice this

[2]All reported runtimes use an unoptimized implementation on a single machine with two 2.66 GHz processors and 32 GB of RAM.

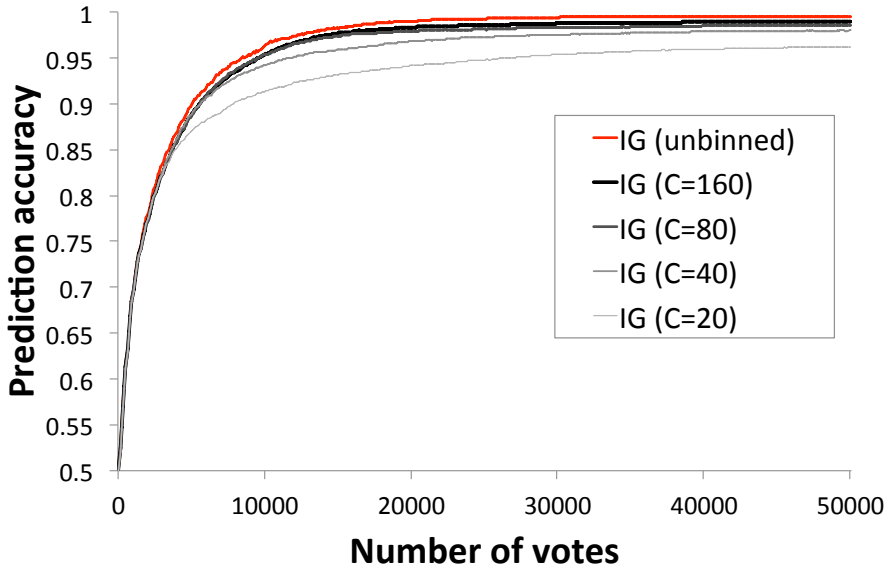


Figure 5.9: Increasing the parameter C governing the number of bins enables JUGGLER$_{\text{BIN}}$ to closely approximate JUGGLER$_{\text{OFF}}$.

cost is much smaller since we do not need to traverse all bins in order to find a valuable question to assign to a worker.

Finally, the performance of JUGGLER$_{\text{BIN}}$ approaches that of the unbinned algorithms as we increase the value of the parameter C, since smaller bins more closely approximate the true utility values. Figure 5.9 shows the results of this experiment. JUGGLER$_{\text{BIN}}$ asymptotes at a slightly lower accuracy than the unbinned approach, suggesting that more sophisticated approaches like adaptive binning may produce further gains.

5.6 Discussion and Future Work

In citizen science and other types of volunteer crowdsourcing, it is important to send hard tasks to experts while still utilizing novices as they become proficient. Since workers are impatient, a task router should allocate questions to *all available workers*

in parallel in order to keep workers engaged. Unfortunately, choosing the optimal set of tasks for workers is challenging, even if worker skill and problem difficulty are known.

This chapter introduces the JoCr framework, characterizing the space of task routing problems based on adaptivity, concurrency and amount of information known. We prove that even offline task routing is NP-hard, but submodularity of the objective function (maximizing expected value of information) enables reasonable approximations. We present $\text{JUGGLER}_{\text{OFF}}$ and $\text{JUGGLER}_{\text{AD}}$, two systems that perform parallel task allocation for the offline and adaptive settings respectively. Using live workers hired on oDesk, we show that our best routing algorithm uses just 48% of the labor required by the commonly used round-robin policy on a natural language named-entity linking task. Further, we present $\text{JUGGLER}_{\text{BIN}}$, a system that can scale to many workers and questions, and $\text{JUGGLER}_{\text{RT}}$, a system that handles variable worker response times and yields similar empirical savings.

These results demonstrate that a self-improving crowdsourcing agent can improve task routing outcomes compared to standard approaches. Further, such an agent can self-improve over time by making use existing methods for improving worker model estimates over time [159], though we do not conduct experiments to demonstrate this point.

Important future work remains to be done. While we have been motivated by volunteer labor from a citizen science scenario, we believe our methods extend naturally to paid crowdsourcing where the utility function includes a linear cost component. In the future we hope to relax the assumptions of perfect information about workers and tasks, using techniques from multi-armed bandits to learn these parameters during the routing process. We also wish to study the case where several different requesters are using the same platform and the routing algorithm needs to balance workers across problem types.

Algorithm 4 The JUGGLER$_{\text{BIN}}$ algorithm

Input: Workers $\mathcal{W}$, prior $P(\mathbf{a})$ over answers, unobserved votes $\mathcal{X}_R$, horizon T

function LOAD($\mathcal{Q}_{in}$)
  for w **in** $\mathcal{W}$ **do**
    for q **in** $\mathcal{Q}_{in}$ **do**
      if $\mathbf{x}_{q,w} \notin \mathbf{x}$ **then**
        $b \leftarrow$ TO_BIN($d_q, P(a_q \mid \mathbf{x})$)
        bins[w][b].add(q)
      end if
    end for
  end for
end function

 # Initialize bins
bins $\leftarrow$ {}
for w **in** $\mathcal{W}$ **do**
  bins[w] $\leftarrow$ {}
  for b **in** ENUMERATE_BINS() **do**
    bins[w][b] $\leftarrow \emptyset$
    $q' \leftarrow$ temp question with $\overline{d}, \overline{P(a)}$ for bin
    $\Delta\overline{X}_{b,w} \leftarrow H(A_{q'}) - H(A_{q'} \mid X_{q',w})$
  end for
end for
LOAD($\mathcal{Q}$)

 # Make assignments
$\mathbf{x} \leftarrow \emptyset$
for $t = 1$ **to** T **do**
    $S \leftarrow \emptyset$
    for w **in** sorted($\mathcal{W}$, key $= \gamma_w$) **do**
        for b **in** sorted(bins$[w]$, key $= \Delta\overline{X}_{b,w}$) **do**
            if bins$[w][b] \neq \emptyset$ **then**
                $q \leftarrow$ bins$[w][b]$.pop()
                $S \leftarrow S \cup \{X_{q,w}\}$
                for w' **in** $\mathcal{W} \setminus \{w\}$ **do**
                    bins$[w']$.remove(q)
                end for
                Break and move to next w
            end if
        end for
    end for
    Observe $\mathbf{x}_S$ and set $\mathbf{x} \leftarrow \mathbf{x} \cup \mathbf{x}_S$
    $\mathcal{X}_R \leftarrow \mathcal{X}_R \setminus S$
    Update prior as $P(\mathbf{a} \mid \mathbf{x})$
    LOAD($\{q$ **for** $x_{q,w}$ **in** $\mathbf{x}_S)\}$
end for

Algorithm 5 The $\text{JUGGLER}_{\text{RT}}$ algorithm

Input: Workers $\mathcal{W}$, prior $P(\mathbf{a})$ over answers, unobserved votes $\mathcal{X}_R$, horizon T

$\mathbf{x} \leftarrow \emptyset$

$S_{busy} \leftarrow \emptyset$

for $t = 1$ **to** T **do**

$S_t \leftarrow$ call $\text{JUGGLER}_{\text{OFF}}(T = 1,\, S = S_{busy})$

$S_{new} \leftarrow S_t \setminus S_{busy}$ and assign S_{new}

$S_{done} \leftarrow$ retrieve completed assignments

$S_{busy} \leftarrow (S_{busy} \cup S_{new}) \setminus S_{done}$

Observe $\mathbf{x}_{S_{done}}$ and set $\mathbf{x} \leftarrow \mathbf{x} \cup \mathbf{x}_{S_{done}}$

$\mathcal{X}_R \leftarrow \mathcal{X}_R \setminus S_{new}$

Update prior as $P(\mathbf{a} \mid \mathbf{x})$

end for

Chapter 6

PRIORITIZING TASKS FOR MULTI-LABEL CLASSIFICATION AND TAXONOMY CREATION

Chapter 5 described methods for routing tasks to workers. Those methods prioritize questions that would benefit most from receiving an additional worker response. However, for certain tasks, crowdsourcing systems (and a self-improving crowdsourcing agent) can achieve further efficiency improvements by reasoning about the value that the answer to one question provides *for related questions*. Multi-label classification, the problem of determining which *set* of labels apply, is one class of such tasks, where the presence or absence of a label can inform how likely other labels are to be applicable.

In this chapter, I present methods by which a self-improving crowdsourcing agent can dramatically improve the efficiency of solving crowdsourcing multi-label classification problems, motivated by the downstream application of taxonomy creation. Similar to the problems of recruiting workers (Chapter 3) and routing tasks (Chapter 5), deciding which questions (labels) to ask represents a large action space, which we solve by formulating a submodular optimization problem. We demonstrate that a self-improving crowdsourcing agent can learn model parameters for the environment, which enable it to improve its performance over time without involvement from the requester (or meta-workers).

6.1 Introduction

Large datasets enable many novel applications, for example, supervised machine learning and data mining. On the other hand, organizing these datasets for easy access

and better understanding requires significant human effort. One such organization practice involves constructing a taxonomy, a hierarchy of categories where each edge denotes an isA relationship (*e.g.*, president isA person). Instances of each category (*items*) are associated with the corresponding node in the taxonomy, for example, the entity Barack Obama is an instance of the president category. WordNet [121] and the Linnaean taxonomy are influential examples.

Taxonomizing large datasets and maintaining a taxonomy over time raise significant challenges, since they are a drain on the ontologist(s) responsible for these tasks. A promising answer to this challenge was recently proposed: a distributed crowdsourcing workflow, called Cascade [27]. Cascade provides a sequence of steps for generating a taxonomy from scratch and for taxonomizing a new item by posing simple questions to unskilled workers on a labor market, such as Amazon Mechanical Turk. Unfortunately, the Cascade workflow was not optimized. While the overall cost of a Cascade-produced taxonomy is comparable to one produced by an expert, Cascade requires about six times as much labor. This suggests that one might be able to refine the workflow, making taxonomy creation both inexpensive and low latency.

Toward this end, we propose Deluge, a decision-theoretic refinement of Cascade. Deluge adopts the high-level skeleton of Cascade's workflow, but optimizes its most expensive step: assignment of category labels to data items. This step is an example of a multi-label classification problem, an important class of problems which has not previously been optimized in a crowdsourcing setting. Where Cascade generates a large number of human tasks for each item-label pair, Deluge saves by ordering the tasks intelligently using a learned model of label and co-occurrence probabilities.

In summary, this chapter presents the following primary contributions:

- We present an efficient solution to the novel problem of crowdsourcing multi-label classification. We describe several alternative methods, culminating in

a decision-theoretic approach with two components: (1) a probabilistic model that estimates the true value of item-label relationships by allowing workers to be probabilistically accurate, and (2) a controller that chooses, for each item, which questions provide the maximum value of information toward a joint categorization.

- We provide theoretical guarantees for the optimality of our control strategy, as well as an efficient method for selecting batches of labels that makes our approach immediately usable in an online labor market environment.

- We conduct live experiments on Mechanical Turk showing that our best combination of policies requires less than 10% of the labor used by CASCADE for categorization.

Beyond reducing the cost of crowdsourcing multi-label classification and taxonomy creation, our work on DELUGE shows that artificial intelligence and decision-theoretic techniques can be applied to more complex workflows than the simple consensus task-based workflows [40, 80, 161] previously tackled.

6.2 Basic Taxonomy Algorithm

Both CASCADE and our refinement, DELUGE, take as input a set of *items* to be categorized, such as photographs or text snippets. Their output is a tree whose interior nodes are each labeled with a text string *label* (category); see Figure 6.5 for an example.

Our taxonomy creation algorithms use a mixture of algorithmic steps and three task schemata, which are submitted to human workers in the labor market. From a functional perspective, these tasks may be defined as follows:

- `Generate` (t items) $\rightarrow$ t labels:

  Displays t items and asks a worker to suggest a label for each item.

Figure 6.1: Sample interface for the `Categorize` worker task primitive used in our Mechanical Turk experiment.

- `SelectBest` (1 item, c labels) $\rightarrow$ 1 label:

  Presents a worker with a single item and c different labels and asks her to pick the best one.

- `Categorize` (1 item, s labels) $\rightarrow$ bit vector of size s:

  Shows a worker a single item and s labels and asks him to indicate which labels apply to the item. (See Figure 6.1 for our interface corresponding to an instance of this task.)

Since humans are the bottleneck in this approach to taxonomy creation, we seek to minimize the number of tasks requested from the labor market. At the highest level, both Cascade and Deluge start by using `Generate` tasks to brainstorm a set of candidate category labels. They then use `SelectBest` tasks to filter out poor labels. Afterwards, `Categorize` tasks identify appropriate labels for all items.

A final, purely algorithmic step, called *global structure inference*, builds a hierarchy from this data by inducing a parent-child relationship between two labels when most of

```
Procedure BuildTaxonomy (Items):
   ItemLabelMatrix := []
   While TaxonomyNeedsImproving?(ItemLabelMatrix) Do
      Labels := ElucidateLabels(the subset of Items with no label)
      For each Item in Items Do
         ItemLabelMatrix := Categorize(Item, Labels, ItemLabelMatrix)
      Taxonomy = GlobalStructureInference(ItemLabelMatrix)
   Return Taxonomy
```

Figure 6.2: The general taxonomy creation algorithm. CASCADE and DELUGE differ in their termination conditions and their implementations of `ElucidateLabels` and `Categorize`.

the items in one label are also in the other. Labels with too few items are eliminated, and labels with too great an overlap are merged. In this chapter, we make no changes to CASCADE's approach to this final step; see [27] for details.

Figure 6.2 summarizes this high-level algorithm, but does not specify exactly how the set of category labels should be elucidated, nor does it state how to categorize each item efficiently using a fixed set of labels. We discuss these issues in the next two subsections. As we shall see, CASCADE takes a relatively simple approach to these questions, but more sophisticated techniques can greatly decrease the amount of human labor required.

6.2.1 Elucidating Category Labels

Noting that there would likely be wasteful duplication if one asked humans (via a `Generate` task) to brainstorm candidate labels for *every* one of the items, CASCADE's implementation of `ElucidateLabels` starts by considering only the first few ($m = 32$)

items, termed the *initial item set*. CASCADE partitions this initial item set into groups of $t = 8$ and creates a `Generate` task for each, which is sent to $k = 5$ workers. After all $\lceil km/t \rceil$ tasks are completed, CASCADE is left with km candidate labels, not necessarily distinct.

CASCADE's next step is to prune the candidate labels. At this point, each of the m initial items will have up to k distinct suggested labels. For each item, CASCADE submits k `SelectBest` tasks requesting a human to choose which of the labels seems most appropriate. Any labels with two or more votes are retained; after this step, $p \leq 2m$ distinct labels remain (assuming $k = 5$).

In the next section, we use a combinatorial balls-and-urns model to describe an alternative, decision-theoretic method for controlling label elucidation.

6.2.2 Categorizing Items Once Labels are Known

Once labels have been elucidated, CASCADE enters its most costly phase, which incurs $O(np)$ worker tasks, where $n = |Items|$ and $p = |Labels|$. Intuitively, the idea is to iterate through the items and labels, asking k different workers whether a label applies to an item. Chilton et al. observed that workers sometimes lack the context to make these decisions, so they proposed categorizing in two sequential phases, which they term *adaptive context filtering*. The first phase iterates through items and labels as described above; every label which receives at least two (out of five) votes progresses to the next phase. In the second phase, workers are only shown labels which made the first round cut, and a label is considered to fit an item if at least four of the five workers deem it so. Thus, both phases together use between $\lceil knp/s \rceil$ and $2\lceil knp/s \rceil$ worker tasks.

In two sections, we present several improved algorithms for this categorization process, which we have noted is a multi-label classification problem. The first approach generates precisely the same labeling with strictly fewer worker tasks. The second uses substantially fewer workers, with little or no loss in classification accuracy.

The final approaches incrementally build probabilistic models of label occurrence and co-occurrence, which they use to optimize the order in which they pose questions to workers.

6.3 Pólya's Urn for Label Elucidation

CASCADE's label elucidation step asks workers to brainstorm relevant labels to be added to the taxonomy. CASCADE performs this step on a set of m items, where $m \ll n$, the total number of items to be categorized. The key insight behind elucidating labels for a small number of items is that labels generated for a random subset of items can be globally relevant, and that workers are likely to repeat labels across items. An important control question for optimizing this step involves the choice of m. CASCADE sets $m = 32$ in an ad hoc manner, but ideally we would like to estimate the quality of a set of category labels as it grows in order to determine when elucidating more labels would likely be wasteful.

DELUGE proposes modeling the brainstorming process using a Pólya urn model [75], also known as a Chinese Restaurant Process. A very general framework, Pólya urn models are particularly suited for modeling discrete, multi-label distributions where the number of labels is unknown a priori, as is the case for our category labels. The metaphor for this generative model is that of an urn containing colored balls, where colors correspond to labels. In each iteration, a ball is drawn uniformly from the urn and then placed back in the urn along with a new ball. If the drawn ball is black (a specially-designated color), the new ball is a previously unseen color; otherwise, the new ball is the same color as the drawn ball.

As balls are drawn from the urn, the number of colors in the urn increases but the probability of obtaining a new color decreases. Moreover, colors that are drawn frequently have a higher probability of being drawn than other colors. This behavior can be seen from the probabilities that govern draws from the urn. Suppose that there are N non-black balls, n_c balls of a specific (non-black) color c, and α black balls.

Then, the probability of drawing a ball of color c is $n_c/(N+\alpha)$ and the probability of drawing a previously unseen color is $\alpha/(N+\alpha)$. A Pólya urn model is parameterized by α; larger values of α imply higher probability of brainstorming new category labels.

A useful quantity to estimate for determining a stopping condition is the expected number of new labels that would be generated by a fixed number of future worker tasks.

Theorem 6. *Let our Pólya urn contain N colored balls and α black balls. Let the random variable X_d be the number of new colors present in the urn after d future draws. Then,*

$$E[X_d] = \sum_{i=0}^{d-1} \frac{\alpha}{N+\alpha+i}.$$

Recall that CASCADE asks k workers to brainstorm labels for each item. Thus, if we have generated labels for m items, with $n - m = r$ items remaining, we have $N = km$ and $d = kr$. Terminating the label elucidation phase at this point will result in an expected $\sum_{i=0}^{kr-1} \alpha/(km+\alpha+i)$ missed labels. The expected fractional increase in the total number of labels is this quantity divided by the number of distinct labels seen after the first m items.

Our model provides a principled stopping condition for this phase: terminate when the expected fractional increase in the number of labels is below a desired threshold. In order to operationalize this policy, we compute the maximum-likelihood estimate of α using gradient ascent on the log-likelihood of generating the observed data.

Note that this model assumes that all labels are independent and that workers are equally likely to generate new labels for any particular item. These assumptions are inaccurate due to the underlying label co-occurrence probabilities, as well as potential differences in the number of accessible labels for each item. However, the approximations are reasonable for the `Generate` phase, since we will likely not have enough data to learn parameters for a more complex model in any case. Our model

lets us estimate the approximate impact of stopping, which can be used to identify an appropriate termination point for this phase.

6.4 Improved Categorization Control Algorithms

CASCADE, like many crowdsourcing workflows, implements voting on binary outcomes by requesting a fixed number of votes k and setting a threshold number of votes T (majority voting is the special case where $T = k/2$). Once the requested number of votes is returned, this procedure returns a positive outcome if and only if the number of positive votes is at least T. The amount of work required by this procedure can be quite large, especially when attempting to scale workflows. In the Adaptive Context Filtering step of its workflow, CASCADE asks k workers to vote on each item and label combination. Supposing there are n items and p labels, this step requires asking for $O(knp)$ votes.

6.4.1 A Lossless Improvement to Threshold Voting

The first observation we make is that given a threshold number of votes T, asking for all k votes is often unnecessary. Once one has received T positive votes, or $(k-T+1)$ negative votes, one need not ask for further votes since the answer using k total votes is fully determined to be positive in the former case and negative in the latter case. We call this stopping condition *lossless stopping*; it can be seen as a generalization of the "Ask two people to vote and only ask a third if the first two disagree" policy in TurKit [109].

6.4.2 One-away Approximation for Threshold Voting

One can further reduce the number of votes required with a simple heuristic method, which we call the *one-away heuristic*, that we hypothesize will result in only a small amount of error compared to the original threshold voting method. (Note that lossless

stopping results in no error, compared to the original.) The one-away heuristic returns true early if we observe $\max\{T-1, 0\}$ positive votes and no negative votes, or returns false early if we observe $\max\{k-T, 1\}$ negative votes and no positive votes. The intuition behind this heuristic is that although the lossless stopping condition may not have been met, we have observed strong evidence for returning an outcome and no evidence in support of the alternative outcome.

6.4.3 A Simple Probabilistic Model

A more powerful way to approach the problem is from the Bayesian perspective. Suppose we have already labeled a large number of items, $I \in \mathcal{I}$, and hence know for each I if label L holds, denoted $\oplus(I, L) = 1$, or does not, denoted $\oplus(I, L) = 0$. Now, when given a new item I' we know nothing about, we can use the previously observed data to calculate the maximum likelihood prior probability of any label $P(\oplus(I', L)) = \sum_{I \in \mathcal{I}} \oplus(I, L)/|\mathcal{I}|$.

In order to update our posterior for $\oplus(I', L)$ after observing a worker's vote, we must model noisy workers. Our worker model uses two parameters to represent how accurately workers are able to detect true positives and true negatives. As in [139], we term these parameters worker *sensitivity* and *specificity*, respectively. We observe that worker specificity is much higher than worker sensitivity due to the sparsity of labels in our dataset, and that representing worker accuracy with two parameters instead of a single shared parameter greatly improves the discriminative ability of our probabilistic models.

If a worker with sensitivity p_{tp} and specificity p_{tn} answers that a label holds, we can update our posterior by simply multiplying the prior by the likelihood ratio $(p_{tp} + (1 - p_{tn})) \,/\, ((1 - p_{tp}) + p_{tn})$. In this model, the agent always knows the most probable value for $\oplus(I, L)$, and if a utility model associates different costs for false positive and false negative classifications, it can easily trade off between these errors.

We term this baseline probabilistic model the *independent* model, since it naively assumes that labels are independent, as shown in the graphical model in Figure 6.3a. If we denote the set of labels by $\mathcal{L}$, the independent model has a total of $(|\mathcal{L}| + 2)$ parameters: one for each label corresponding to the prior probability of that label, and two for a noisy worker model that we assume here is shared among all workers. The marginal label probabilities are

$$P(L \mid \mathbf{v}) \propto P(L)P(\mathbf{v}_L \mid L),$$

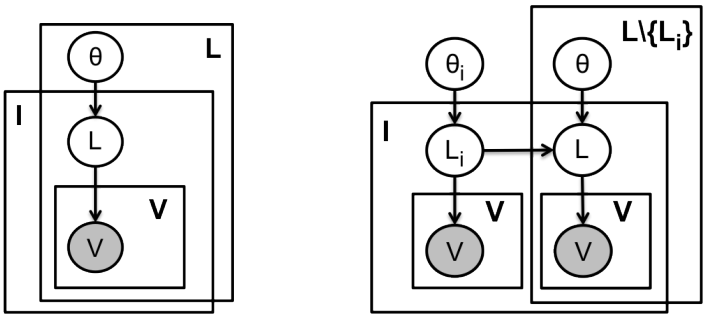
where $L \in \mathcal{L}$ is a Boolean random variable corresponding to an outcome $\oplus(I, L)$ for an item and $\mathbf{v}_L \subseteq \mathbf{v}$ is the vector of observed votes associated with that outcome.

One subtlety concerns the treatment of past data. Since the agent has no access to gold data, it does not know the true labels, $\oplus(I, L)$, even when it has seen the assessments of many workers. We use expectation maximization (EM) to estimate the values of these latent labels together with the parameters of our model. We perform a Bayesian estimate of our parameters to avoid problems when categorization is just starting, by assuming weak symmetric Beta priors and computing a maximum a posteriori estimate.

6.4.4 Modeling Label Co-occurrence

The assumption of label independence made by the previous model is a substantial approximation. For example, an item which has been categorized as "person" is more likely to be a member of the actor category than to be a member of the location category. This observation is especially pertinent to taxonomies with deep hierarchical structure, but is true for any set of overlapping labels.

It is natural, therefore, to learn a joint model of label probabilities. In this model, when a worker responds that an item is in a given category, the posterior for *all* other categories can be updated. This update will also affect the choice of which label has the highest value of information by the control strategy we will define.



(a) Independent (b) MLNB

Figure 6.3: Generative probabilistic models for multi-label classification. The I, L, and V plates correspond to items, labels, and votes, respectively. The MLNB model in (b) is the model for predicting label L_i; there are $|\mathcal{L}|$ such models.

There are many ways to represent a complex joint distribution. As a baseline approach, we explore a simple model, which we term the *multi-label naive Bayes* (MLNB) model. For each label in this model, we construct a star graph with directed edges from that label to all other labels; the graphical model in Figure 6.3b shows the star graph for label L_i. Using notation we defined for the independent model, the marginal label probabilities for the MLNB model are

$$P(L \mid \mathbf{v}) \propto P(L)P(\mathbf{v}_L \mid L) \prod_{L' \in \mathcal{L} \setminus \{L\}} \sum_{L'} P(L' \mid L)P(\mathbf{v}_{L'} \mid L').$$

Calculating marginal probabilities for all labels requires $O(|\mathcal{L}|^2)$ computations per item and involves summing out the latent label variables represented as child nodes in the graphical model. This approximation models pairwise co-occurrences between labels directly but ignores the higher order interactions.

In order to estimate parameters for the MLNB model in an efficient manner, we reuse parameters and label predictions obtained by running EM for the independent model. We approximate the additional $2(|\mathcal{L}^2| - |\mathcal{L}|)$ conditional label probabilities $P(L' \mid L)$ using these predictions as the expected fraction of items with labels L and

L' out of those with label L. Knowing that a small fraction of labels applies to any particular item, we use Laplace smoothing to bias these estimates against positive co-occurrence.

We have also explored a more sophisticated probabilistic graphical model that combines our generative independent model with a pairwise Markov network over the label variables for an item. We found that this model, which requires approximate inference methods, is too slow to be useful in a live crowdsourcing setting and does not produce significant gains over the MLNB model; we do not describe it further in this chapter.

6.4.5 Choosing Which Questions to Ask

One may also consider different control strategies for choosing the next question(s) to dispatch to a worker. CASCADE employed a simple *round-robin* strategy, but we advocate using a *greedy* search that asks about the label(s) where a worker's vote(s) would provide the greatest value of information.

More formally, each time DELUGE asks a worker for new votes, its goal is to select a set of votes that will result in the greatest expected decrease in the uncertainty of our label predictions. Information theory provides us with a useful criterion for measuring the amount of uncertainty in the distribution of label predictions, the joint entropy

$$H(\mathcal{L}) = -\sum_{\mathbf{l} \in \operatorname{dom} \mathcal{L}} P(\mathbf{l}) \log P(\mathbf{l}).$$

The domain of $\mathcal{L}$ consists of all possible assignments to the variables in $\mathcal{L}$, and $\mathbf{l}$ denotes one of those assignments. Let $\mathcal{A} \subset \mathcal{V}$, where $\mathcal{V}$ denotes the unbounded set of possible future votes. The expected uncertainty of the distribution of label predictions after receiving the votes in $\mathcal{A}$ is the conditional entropy

$$H(\mathcal{L} \mid \mathcal{A}) = -\sum_{\substack{\mathbf{l} \in \operatorname{dom} \mathcal{L} \\ \mathbf{a} \in \operatorname{dom} \mathcal{A}}} P(\mathbf{l}, \mathbf{a}) \log P(\mathbf{l} \mid \mathbf{a}).$$

We are interested in maximizing the difference of these two quantities, known as the expected information gain, or the mutual information $I(\mathcal{L}; \mathcal{A}) = H(\mathcal{L}) - H(\mathcal{L} \mid \mathcal{A})$.

Unfortunately, calculating the optimal set, $\mathcal{A}$, that maximizes information gain is intractable due to the combinatorial nature of the problem. However, we are able to select a near-optimal set by exploiting the combinatorial concept of *submodularity*. Nemhauser, Wolsey, and Fisher [125] show that the greedy algorithm for optimizing a submodular function provides a solution that is guaranteed to be within a constant factor of $(1 - 1/e) \approx 63\%$ of optimal. While information gain is not, in general, submodular, it does satisfy this property under our modeling assumption that workers' errors are not correlated, *i.e.*, that votes in $\mathcal{V}$ are independent given the true values of $\mathcal{L}$ [92]. Krause and Guestrin provide a greedy algorithm for selecting a near-optimal subset of variables under this assumption, and prove that one cannot achieve a tighter bound unless $\mathbf{P} = \mathbf{NP}$.

This greedy algorithm accumulates a set of future votes $\mathcal{A}$ by adding votes $V \in \mathcal{V}$ one at a time with a greedy heuristic. While we are interested in the set of votes that maximizes the information gain for $\mathcal{L}$, the greedy heuristic selects a vote V by ranking them according to the quantity $H(V \mid \mathcal{A}) - H(V \mid \mathcal{L})$. In general, these conditional entropies require that we represent the full joint distribution over $\mathcal{L}$, which is intractable for even a small number of labels. Fortunately, we can refine this heuristic using an additional conditional independence assumption of our models, which simplifies $H(V \mid \mathcal{L})$ to the local conditional entropy $H(V \mid L_V)$, where $L_V \in \mathcal{L}$ is the label corresponding to vote V.

Theorem 7. *Let each vote $V \in \mathcal{V}$ be independent of all other votes given the label L_V, and let $\mathcal{A}$ be the set of future votes accumulated by the greedy algorithm thus far. Also let V_L denote an arbitrary future vote for some label L. Then, the set $\mathcal{A}$ constructed by successively adding future vote V^* by the strategy*

$$V^* \in \operatorname*{argmax}_{L \in \mathcal{L}} H(V_L \mid \mathcal{A}) - H(V_L \mid L)$$

is within $(1 - 1/e)$ *of optimal.*

The proof follows from applying Krause and Guestrin's result to our model. Note that when the greedy algorithm selects the first vote, $\mathcal{A}$ is initially empty and thus $H(V \mid \mathcal{A})$ is simply $H(V)$.

This theorem yields a surprising result: selecting a near-optimal single question to ask a worker requires only the local entropies $H(V)$ and $H(V \mid L_V)$. DELUGE leverages this greedy strategy, along with the MLNB model of label co-occurrence, to optimize the categorization process.

6.5 Experiments

In our experiments, our goal is to compare the various strategies from the categorization control section. We first compare the simple improvements to threshold voting by analyzing the cost savings for each strategy (lossless, one-away) and threshold setting $T = \{2, 3, 4\}$, along with the quality of the taxonomy produced. Next, we evaluate the probabilistic models on their predictive performance and compare that against the original strategy from CASCADE.

6.5.1 Dataset

In order to better analyze the effect of different categorization algorithms, we controlled for variation in label elucidation policies and adopted a fixed set of candidate category labels from the literature. Specifically, we took a subset of the fine-grained entity tags described in [107] by eliminating low probability tags and all those for organizations, events, and facilities; this process yielded a manageable set of 33 labels. We then over-generated items for each of these labels, and constructed a random subset of 100 items.

Our worker vote collection process involved emulating a run of the `Categorize` procedure from CASCADE, called on these 100 items and 33 categories. We had a

Method	T	F-score	Votes	% savings
Lossless	2	0.83	130	21
One-away	2	0.82	96	42
Lossless	3	0.84	102	38
One-away	3	0.83	69	58
Lossless	4	0.75	72	56
One-away	4	0.70	38	77

Table 6.1: Comparison of threshold voting methods. The table shows mean F-score, number of votes per item, and % savings relative to CASCADE's method of always collecting five votes per label.

total of $k = 15$ workers from Mechanical Turk vote on batches of seven labels per Human Intelligence Task (HIT). The interface for our HITs, shown in Figure 6.1, uses form validation to ensure that a worker either selects at least one label, or deliberately indicates that none of the displayed labels apply to the item in question. Each HIT cost $0.04 and the total amount paid to workers was $300. The purpose of gathering this data was to allow us to compare different control strategies, controlling for worker error, since each control strategy would be seeing the same worker responses.

6.5.2 Threshold Approaches

In the first experiment, we compared the threshold voting modifications to the naive version of threshold voting implemented in CASCADE. Since CASCADE uses various threshold settings in the adaptive context filtering, we tested with thresholds of $T = \{2, 3, 4\}$ out of 5 total votes. Table 6.1 shows the number of votes per item used by lossless stopping and the one-away heuristic, and the fraction of votes saved compared to the original approach in CASCADE, which used $33 \times 5 = 165$ votes per item. Note

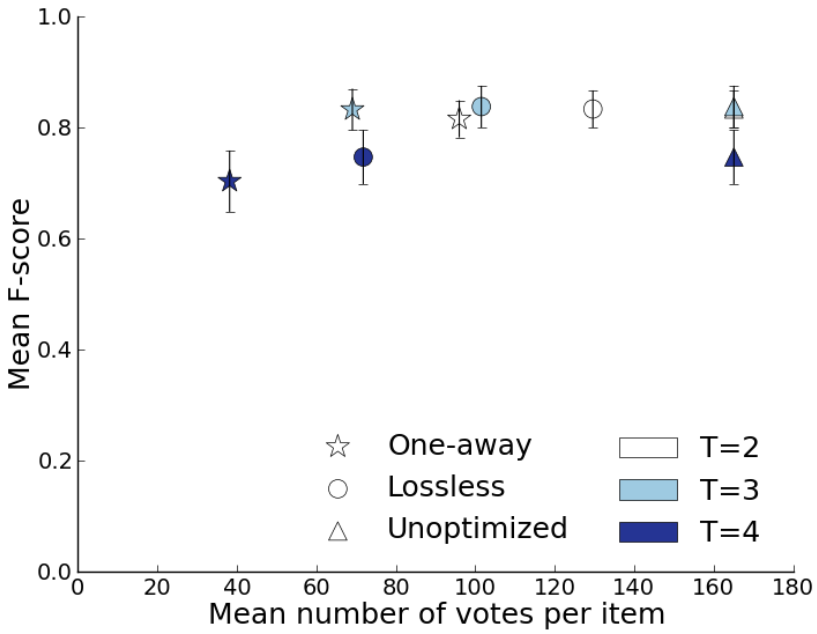


Figure 6.4: F-score vs. cost for threshold voting improvements.

that lossless stopping, which returns exactly the same predictions as the threshold voting procedure from CASCADE, is able to save up to 56% of the votes when $T = 4$.

In order to better understand the impact of the one-away heuristic on classification performance, in Figure 6.4 we plot F-score performance vs. the number of votes used for lossless stopping and the one-away heuristic. For threshold values $T = \{2, 3\}$, the one-away strategy significantly lowers the already reduced cost associated with lossless stopping without introducing a statistically significant decrease in F-score. The decrease in F-score for $T = 4$ is statistically significant ($p < 0.01$ using a two-tailed paired t-test), but can be attributed to poor recall. The one-away heuristic at this threshold setting returns false if the first vote is negative, which is suboptimal since worker sensitivity is significantly lower than worker specificity.

• **person (41)**	Jesus, Kenny G, The Pope, Mel ...
• **actor (9)**	Madonna, Meg Ryan, Eminem, Ha ...
• **director (2)**	Mel Gibson, Brad Pitt
• **religious leader (8)**	The Pope, rabbi, Pope John Pa ...
• **politician (8)**	Queen Elizabeth, Hillary Clin ...
• **artist (8)**	Eminem, Madonna, Monet, Georg ...
• **musician (5)**	Eminem, Kenny G, George Harri ...
• **football player (6)**	Peyton Manning, Joe Montana, ...
• **author (4)**	Martha Stewart, John Locke, I ...
• **organization founder (3)**	Martha Stewart, Steve Jobs, U ...
• **military person (3)**	Saddam Hussein, Frederick the ...
• **location (15)**	Fiji, Indonesia, Shanghai, Wa ...
• **country (7)**	Fiji, Greenland, Indonesia, F ...
• **island (6)**	Fiji, Whidbey Island, Greenla ...
• **city (4)**	Shanghai, Washington, Florida ...
• **river (3)**	The Charles River, The Potoma ...
• **vehicle (12)**	Honda Accord, Hummer, MiG-23, ...
• **car (6)**	Chevy Tahoe, Ford Taurus, Hum ...
• **truck (3)**	Chevy Silverado, tractor trai ...

Figure 6.5: The one-away policy with threshold $T = 3$ used only 42% of the labor required by CASCADE, yet produced an excellent taxonomy (excerpt shown).

In addition to classification performance, we are also interested in how our improvement methods impact the quality of the final output taxonomy. Visual inspection for errors in the output taxonomies did not reveal a decrease in quality when using the one-away heuristic. Figure 6.5 shows a high-quality taxonomy produced by the one-away heuristic with threshold $T = 3$.

6.5.3 Inference-based Approaches

We hypothesized that scaling multi-label classification and taxonomy creation to a large number of items requires a probabilistic approach. To empirically determine the

effectiveness of our approaches, we compared the performance of various inference and control strategies using the votes gathered from Mechanical Turk.

In our experiments, we tested three inference methods (MLNB, Independent, and Majority) and two control strategies (greedy and round-robin). MLNB and Independent inference methods were described in the previous section, and Majority performs simple majority vote estimation that defaults to a negative answer and breaks ties in favor of a positive answer (we found that these simple modifications improved results for our dataset). The greedy control strategy uses the heuristic from Theorem 7 to select labels that maximize information gain, while the round-robin strategy fills in votes layer by layer (*e.g.*, it asks once about each label before asking twice about any label). Majority with round-robin is our reconstruction of the original CASCADE approach.

In order to test how our models will perform when scaling in the number of items, we evaluate the performance of our models using leave-one-out cross-validation for the 100 items. We estimate model parameters using 99 items and five worker votes for each item-label pair in the training set.

Figure 6.6 shows the results of this experiment. MLNB and Independent show a clear improvement over the simple round-robin method, and MLNB in particular reaches high levels of performance very quickly. The improvement of MLNB over Independent is highly statistically significant at the 0.05 significance level (using a two-tailed paired t-test) for the first 47 votes, lending credence to our hypothesis that co-occurrence information aids classification. Furthermore, points where Independent crosses slightly above MLNB are not statistically significant. We note that the probabilistic models sometimes request votes in excess of the votes collected for an item-label pair, in which case we simply use the next-best label; this behavior does not happen frequently enough to impact our statistically significant results.

So, which control strategy is best? On this dataset, CASCADE's voting policy (accept a label if four out of five workers think it applies) required 165 worker tasks per

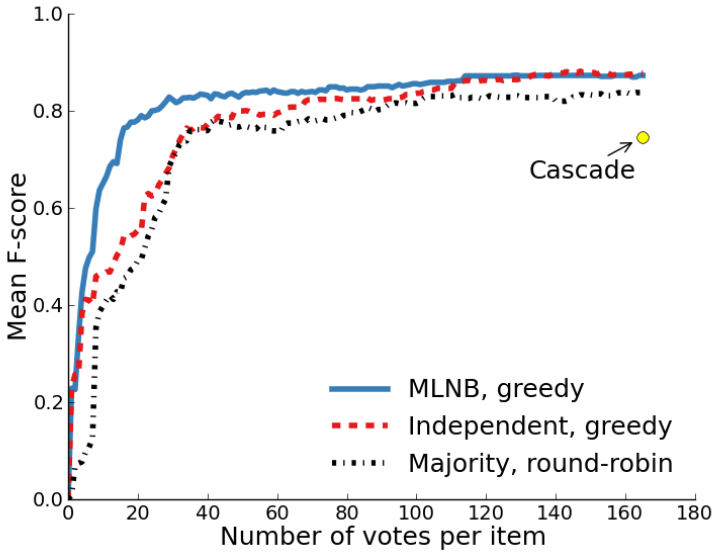


Figure 6.6: Performance vs. number of votes (Mechanical Turk data). CASCADE does not fall on the Majority line, as it uses a different threshold ($T = 4$).

item and yielded an F-score of 75% when compared to gold-truth data. In contrast, our one-away strategy with threshold $T = 3$ had an F-score of 83% and used only 42% as much labor. Our probabilistic approaches are anytime and can be stopped after any number of worker tasks. MLNB with a greedy control strategy produced an F-score around 76% after only 16 tasks per item, which is less than 10% as much labor as CASCADE required to achieve similar performance.

6.5.4 Batching Tasks

In order to be practically useful in a crowdsourcing setting, our control strategies need to be able to group tasks together so that a worker can answer multiple questions about an item at once; see Figure 6.1 for an example. Theorem 7 provides a method for choosing batches of labels, by accumulating a set of votes using the greedy heuristic.

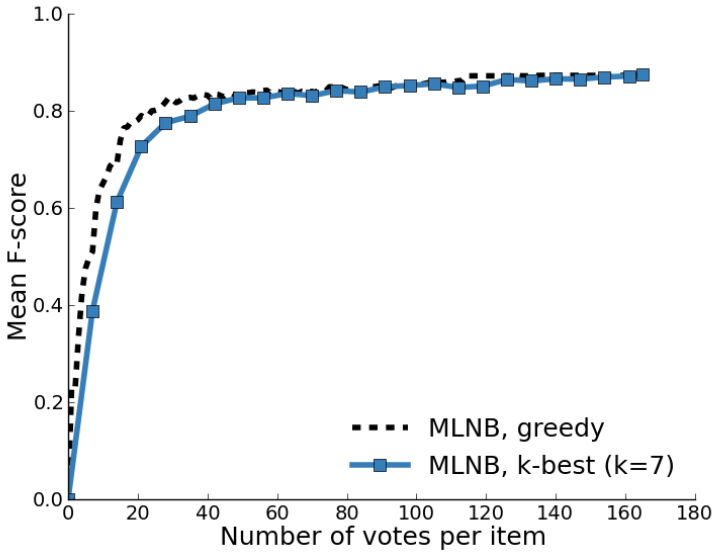


Figure 6.7: Performance vs. number of votes for selecting batches of $k = 7$ (Mechanical Turk data, MLNB with single label selection shown for comparison).

An alternative simple control strategy, which we term k-best, simply selects the top k labels ranked by the greedy heuristic before any votes have been accumulated.

In our experiments, we found that k-best offers the best trade-off between classification performance and computational complexity. Figure 6.7 shows that for $k = 7$ (the same number used by CASCADE and our own live experiment), MLNB with k-best control results in a small decrease in performance compared to MLNB with single label selection. This difference is statistically significant (at the 0.05 significance level using a two-tailed paired t-test) only until about 35 votes per item, and the batched version of MLNB still outperforms Independent with single label selection.

The accumulative greedy method failed to produce significant performance gains over k-best. Moreover, computing the greedy heuristic for $k > 1$ is computationally intensive, requiring approximation of conditional entropies for the MLNB model. One possible reason the accumulative method fails to improve performance is that labels

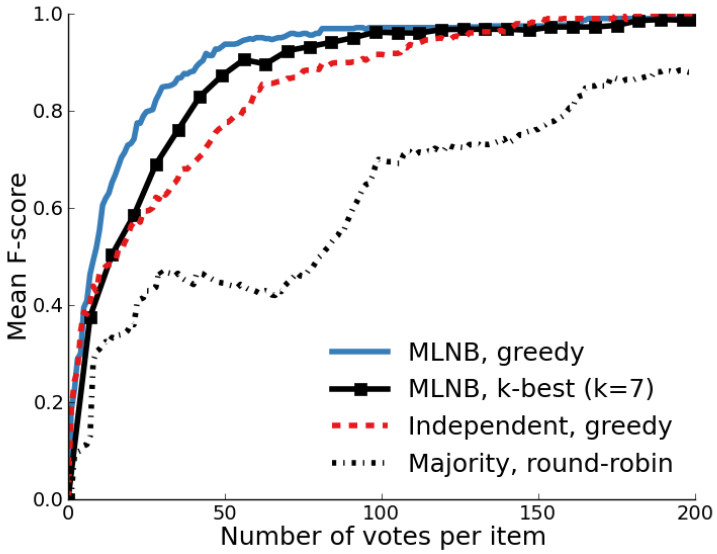


Figure 6.8: Performance vs. number of votes for a more difficult simulated task (sensitivity = 0.6, specificity = 0.8).

within a batch must be distinct in our setting (it is not beneficial to ask the same worker the same question more than once). Given this restriction, k-best is an effective heuristic that incurs no additional cost compared to selection of single labels.

6.5.5 Simulation Study

An intelligent control procedure must be robust to noise due to worker quality. In order to assess the behavior of our techniques on more complex classification problems where the workers may be more error-prone, we simulated workers with 60% sensitivity and 80% specificity. We perform this experiment using the gold-truth item-label answers in a purely simulation setting. The overall higher performance of our results in Figure 6.8 despite less accurate workers (average sensitivity and specificity for workers in our dataset was 76% and 98%, respectively) can be attributed to discrepancies between the gold-truth answers supplied by the authors and the collective

decisions made by workers on Mechanical Turk. Figure 6.8 shows the same statistically significant ordering of the models as we saw with real worker votes, suggesting that our results generalize to a wide array of multi-label classification tasks.

6.6 Discussion and Future Work

Machine learning and decision-theoretic techniques offer the potential for dramatically reducing the amount of human labor required in crowdsourced applications. However, to date, most work has focused on optimizing relatively simple workflows, such as consensus task and iterative improvement workflows. Taxonomy generation is an important task, which requires a complex workflow to create a globally consistent interpretation of a large dataset from workers who typically have only a narrow view of a small data subset. Since previous work on crowdsourcing taxonomy creation, Cascade, was both promising yet labor intensive, it is a natural target for decision-theoretic optimization.

This chapter presents Deluge, a refinement of the Cascade algorithm with novel approaches to the subproblems of label elucidation and multi-label classification. For the former, we introduce a combinatorial Pólya urn model that allows us to calculate the relative cost of stopping the label generation phase early. For the problem of classifying items with a fixed set of labels, we present four models: lossless, one-away, a simple probabilistic model, and the MLNB model of label co-occurrence. The latter two models support a greedy control strategy that chooses the most informative label to ask a human to evaluate, within a constant factor of the optimal next label. We also provide a batching strategy, making our approach to multi-label classification both highly general and practically useful.

Using a new dataset of fine-grained entities, we performed live experiments on Mechanical Turk to evaluate the relative effectiveness of the approaches to multi-label classification. While Cascade's voting policy required 165 worker tasks per item, our approaches achieve superior performance using much less labor. In par-

ticular, DELUGE uses MLNB with a greedy control strategy to exceed CASCADE's performance after only 16 tasks per item, or less than 10% as much labor.

These results suggest that a self-improving crowdsourcing agent can—and should—prioritize questions when the answer to one question can provide information about other questions. We showed that submodular optimization can be an effective technique that can help such an agent efficiently compute the next question to ask, which is critical when there are many candidate questions. Our experiments show that the agent can model its environment and improve its own performance without help from the requester (or meta-workers); in our domain, a small amount of training data, combined with a probabilistic model, was sufficient to produce a significantly better policy.

We envision extending this work in a number of ways. Our probabilistic models do not distinguish between individual workers, since we focus on comparing different representations of the underlying distribution on labels. However, learning individual noisy worker models would likely improve results for these models. Another line of inquiry involves exploration of the design implications of this work. For example, our anytime probabilistic approaches could be used to pose questions with a dynamic interface that updates as a worker provides responses. Finally, we hope that our work inspires other researchers to tackle the design and optimization of workflows for more complex problem domains.

Chapter 7

EFFICIENTLY SPECIFYING TASKS AND IMPROVING INSTRUCTION WORKFLOWS

The methods presented up to this point (Chapters 3– 6) have made a major assumption: the requester has provided the task instructions, as well as testing and training inputs. However, designing the task and creating these inputs are among the most difficult [3] and important [54, 72, 110] problems faced by requesters. The success of a crowdsourcing project often depends to a large extent on how well the requester designs the task.

Given the importance of task design and the challenge of performing it well, it is critical that a self-improving crowdsourcing agent support the requester in this activity. This chapter presents methods that combine the efforts of the requester, workers, and algorithms to reduce the effort of task design and improvement for the requester and improve task outcomes. Unlike the methods from Chapters 3– 6, where the agent improved its own performance by interacting with workers performing the task, in this chapter, I present a task improvement tool, Sprout, that engages meta-workers in more direct task improvement activities. Sprout then uses responses from these meta-workers to semi-automate task improvement and aid the requester in creating new instructions and testing questions that can be used for worker management (Chapter 4). By helping the requester create new actions (better instructions) that improve the agent's performance, this work fully closes the requester-in-the-loop task improvement loop (left side of Figure 1.1).

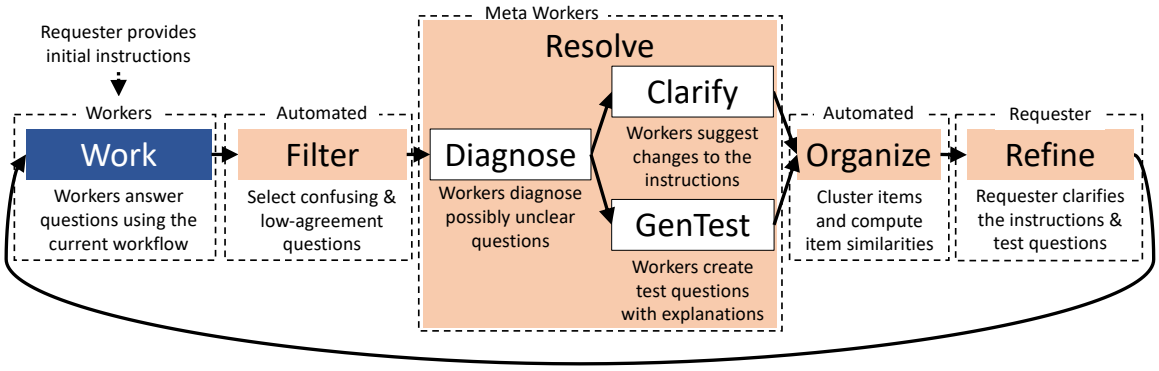


Figure 7.1: SPROUT is our implemented task-improvement meta-workflow (workflow for improving a task workflow), that interleaves steps where workers answer questions using the base workflow (blue box) and meta steps (orange boxes), where meta-workers diagnose problems and suggest fixes, while SPROUT guides the requester to iteratively improve the instructions, add clarifying examples, and insert test questions to ensure understanding.

7.1 Introduction

Ensuring high-quality work is considered one of the main roadblocks to having crowdsourcing achieve its full potential [162]. The lack of high quality work is often attributed to unskilled workers, though it can equally be attributed to inexperienced or time-constrained requesters posting imperfect task designs [72, 165]. Often, unclear instructions confuse sincere workers because they do not clearly state the task expectations [54]. In other cases, the task may be clear but complex; here, the lack of guided practice creates a mismatch between worker understanding and task needs [47, 110]. Finally, in many cases, the requesters themselves do not appreciate the nuances of their task, a priori, and need to refine their task definition [94].

Our hypothesis is that explicit or implicit feedback from workers can guide a requester towards a better task design. Unfortunately, existing tools for crowdsourcing

fall severely short in this regard. While they often include best practice recommendations to counter variance in worker quality [38] (e.g., gold standard question insertion for identifying under-performing workers, aggregation of redundant labels), they do not provide mechanisms for supporting requesters in effectively defining and designing the task itself, which can mitigate the need for these downstream interventions.

In response, we present a novel meta-workflow that interleaves tasks performed by both crowd workers and the requester (see Figure 7.1) for improving task designs. Sprout, our initial prototype, focuses on clarifying the task instructions and ensuring workers follow them, which are difficult [3] and important [54, 72, 110] aspects of task design. Sprout evaluates a preliminary task design and organizes confusing questions by clustering explanations and instruction edits suggested by crowd workers. Sprout's dashboard displays these organized confusions, allowing the requester to navigate their own dataset in a prioritized manner. The system goal is to support the requester in efficiently identifying sources of confusion, refining their task understanding, and improving the task design in response.

Sprout provides additional support for ensuring workers understand the instructions. It allows requesters to embed illustrative examples in the instructions and recommends potential test questions (questions with reference answers that test understanding of the instructions). Upon acceptance by the requester, the instructions and test questions are compiled into *gated instructions* [110], a workflow consisting of an interactive tutorial that reinforces instruction concepts and a screening phase that verifies worker comprehension before commencing work (see Figure 7.2). Overall, Sprout provides a comprehensive interface for requesters to iteratively create and improve gated task instructions using worker feedback.

We evaluate Sprout in a user study, comparing it against *structured labeling* [94], a previous method that is likely to aid requesters in creating instructions [22], while their understanding of the task may be evolving (unassisted by workers). Requesters who participated in our study created gated instructions for two different types of

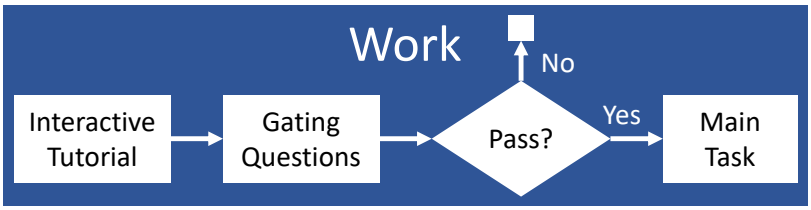


Figure 7.2: Sprout runs a gated instruction workflow [110] in the *Work* step of the meta-workflow (Figure 7.1), which ensures workers understand the instructions before starting the main task. Workers who do not pass gating do not continue with the task (indicated by the terminal square). The *Refine* step of the meta-workflow updates all parts of this workflow (before the first *Refine* step, only the main task is run since the system cannot yet construct tutorial or gating questions).

labeling tasks—the most common crowdsourcing task type [72]—strongly preferred and produced higher-rated instructions using Sprout.

In summary, this paper makes four main contributions:

- A novel meta-workflow—combining the efforts of both crowd workers and the requester—that helps the requester create high-quality crowdsourcing tasks more quickly and with substantially less effort than existing methods.
- Sprout, an open-source tool that implements this workflow for labeling tasks, the most common crowdsourcing task type [72]. Sprout first has workers suggest changes to the task instructions. It then clusters the suggestions and provides the requester with a comprehensive task-improvement interface that visualizes the clusters for fast task exploration and semi-automates the creation of a gated instruction (training and testing) workflow by suggesting test questions related to the instructions the requester has written.
- A user study with requesters with varying amounts of crowdsourcing experience comparing Sprout and structured labeling on two different types of labeling

tasks. The results demonstrate an overall preference for SPROUT over structured labeling and for the use of worker feedback during task design. Furthermore, requesters using SPROUT produced instructions that were rated higher by experts.

- A set of design principles for future task authoring and debugging tools, informed by our experience building SPROUT, and our observations and discussion with requesters during the user study.

We implement the SPROUT tool as a web application and release the source code for both it and for structured labeling in order to facilitate future research.[1]

7.2 Sprout: A Tool Supporting Task Design

In this section, we present the design of SPROUT, our system for efficiently creating gated task instructions for new tasks. The design decisions for SPROUT are based on previous work and the authors' extensive experience running crowdsourcing tasks, and were iteratively refined through pilot studies with workers and requesters (target users).

SPROUT embodies a feedback loop for task authoring and debugging. First, the requester writes a version of the instructions, which are shown to the crowd on an evaluation set (a small subset of the data) during a *Work* step of the meta-workflow (Figure 7.1). SPROUT identifies possibly confusing questions during an automated *Filter* step using signals such as low inter-annotator agreement. A different set of (meta)-workers then perform a *Diagnose* step (Figure 7.3b), where they decide if the question is ambiguous given the current instructions. Immediately after the *Diagnose* step, workers perform either a *Clarify* step (Figure 7.3c) where they edit the instructions based on their own definition of the task (if they diagnose the question to be ambiguous) or a *GenTest* step where they create a test question with an explanation

[1] `https://crowdlab.cs.washington.edu/task-design.html`

(if they believe the question has an unambiguous answer). The *Diagnose*, *Clarify*, and *GenTest* steps are implemented as a single, conditional Resolve HIT (Figure 7.3), in which workers take one of two possible paths (see Figure 7.1).

During a subsequent *Organize* step, SPROUT uses these edits and explanations to cluster various items and create item-item similarity scores. These clusters (and closely related items) are exposed in SPROUT's dashboard (Figure 7.4), which allows the requester to efficiently identify various ambiguities in the previous task design as part of a *Refine* step. The requester improves the instructions and test questions on the basis of this feedback, SPROUT compiles these into gated instructions, and the feedback loop repeats. When the current task design no longer results in worker confusion or the requester ends the process, the final task design is run on the whole dataset.

7.2.1 Finding and Characterizing Ambiguous Items

SPROUT's *Filter* step identifies possible points of confusion in a *Work* step (run on the requester's current instructions), using either indirect signals (e.g., questions with low inter-annotator agreement) or direct signals (e.g., via a feedback input on the task itself). Possibly confusing questions trigger Resolve HITs, where crowd workers *resolve* potential ambiguities and in the process generate useful metadata for organizing the dataset and creating gated instructions.

Resolve HIT Part 1: In the first part of the HIT (Figure 7.3b), a worker performs a *Diagnose* step by labeling whether the question (Figure 7.3a) could have multiple correct answers (is ambiguous) or has exactly one answer (is unambiguous). Depending on their response, the worker subsequently performs either a *Clarify* step or *GenTest* step, respectively, in the second part. These subsequent steps take about the same amount of work, so workers tend to perform *Diagnose* steps honestly.

The *Diagnose* step is designed to improve work quality. Our initial design omitted the *Diagnose* step, instead asking workers to perform a *Clarify* or *GenTest* step in

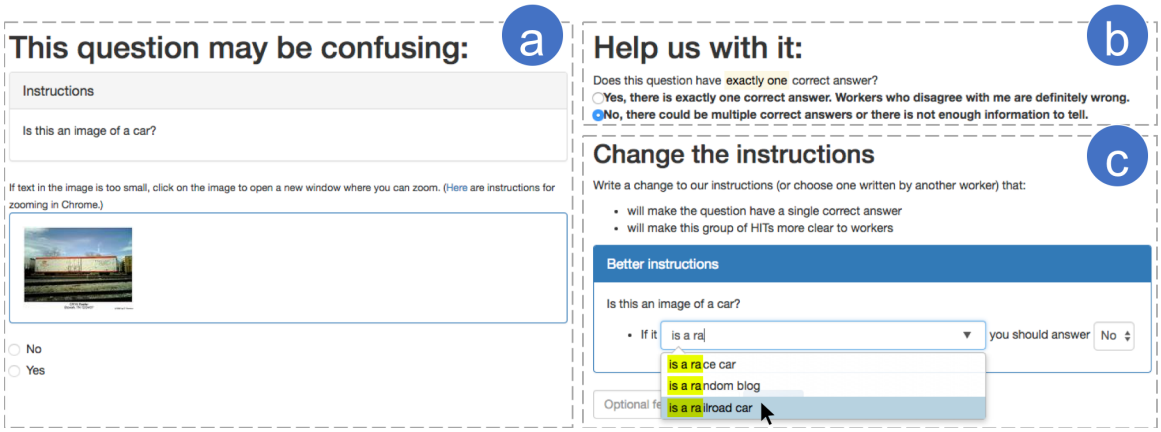


Figure 7.3: The Resolve meta-worker HIT primitive, which implements the *Diagnose*, *Clarify*, and *GenTest* steps of the meta-workflow (Figure 7.1). A worker (a) is shown a question from the base task (here, the *Cars* task) and (b) is asked to perform a *Diagnose* step. If she decides the question is ambiguous (has multiple correct answers) given the current instructions, she then (c) performs a *Clarify* step by adding a rule to the instructions (based on how she might define the task). Workers who decide the question is unambiguous instead perform a *GenTest* step (alternative version of (c), not pictured) by creating a test question. Demonstrations of both versions of (c) appear in the supplemental `sprout.mp4` video (Appendix B).

the appropriate location of a single form. However, some workers entered *GenTest* justifications in the intended *Clarify* location. Forcing workers to make an explicit initial judgement and dynamically adding a follow-up question helps to reduce these errors.

Resolve HIT Part 2, *Clarify* **Option**: If the worker decides the question is ambiguous, SPROUT elicits a category from the worker (via the text input box in Figure 7.3c) by having them perform a *Clarify* step in the second part of the HIT. This step consists of adding a clarification bullet to the instructions by describing the nature of ambiguity (category) and deciding how items in that category should be labeled if *they* were the requester (*yes* or *no*). SPROUT ultimately discards worker labeling decisions (since only the requester can make the final determination); their only purpose is to make the HIT feel more natural to workers. The category input field auto-completes to categories previously written by other workers to help workers reduce redundancy and arrive at a relatively small set of categories for future review by the requester.

SPROUT's method of eliciting ambiguous categories by having workers directly suggest edits to the instructions is designed to produce a rich set of categories. Workers in our experiments often entered non-standard categories that function as rich decision boundaries, useful for defining the task acceptance criteria, e.g., workers entered "has the car as the main subject" or "has windshields and seats and wheels" which could help define acceptable car images. Simply asking workers to categorize ambiguities did not produce these types of categories.

SPROUT's *Clarify* step also aims to produce focused text to improve similarity comparisons and clustering results in the next *Organize* step of the meta-workflow. Describing an ambiguity *in the context of instructions that other workers will see* helps keep the text succinct. For example, the first two workers performing *Clarify* steps for the same question entered "should only include photographs or realistic images of birds" and "is a toy bird," and a third worker also entered "is a toy bird" (via auto-

complete). These short phrases could all be included directly in the instructions. When asked to explain ambiguities without this context, workers often entered many words unrelated to the actual ambiguity.[2]

Resolve HIT Part 2, *GenTest* **Option**: Workers that decide the question is unambiguous instead perform a *GenTest* step in the second part of the HIT (alternative version of Figure 7.3c, not pictured), where they create a test question (for use in the gated instruction workflow) by marking the correct answer and providing an explanation. These questions are good candidates for testing workers because (1) a worker has a reason for why it is unambiguous and (2) it is likely to help filter workers who do not fully understand the instructions and initially disagreed with that worker, causing the question to be flagged.

For some questions, multiple workers indicated that an item is not confusing by performing *GenTest* steps, but submitted conflicting answers. We believe this is an important source of ambiguity, which likely happens due to differing interpretations of the same instructions. We include all such items in the set of ambiguous items and perform automatic clustering based on *GenTest* step explanations (since *Clarify* step categories are unavailable).

7.2.2 Clustering and Determining Related Items

Organize is the next meta-workflow step; here, Sprout uses all worker feedback to organize confusing questions for prioritized exploration by the requester and to determine question relatedness for context. It also maintains information for suggesting test questions to the requester in the *Refine* step. Toward this end, Sprout creates (1) a two level-hierarchy of ambiguous categories, (2) a priority order for top-level categories, and (3) similarity scores for each item pair.

---

[2]E.g., one worker wrote, "Although the image is of a bird made from legos, it is still an image of a bird. I would think that meets the criteria. However, the instructions are a bit ambiguous and don't say whether it needs to be an actual bird or one depicted in an image."

SPROUT adapts the taxonomization algorithm from Cascade [27] for creating its prioritized hierarchical clusters. Proceeding from the largest categories (auto-completed instruction edits from *Clarify* steps) with the most confusions to the least, SPROUT selects a category to include at the top-level and nests all smaller categories with overlapping items as "related" categories (see Figure 7.4a). This also creates a natural order for top-level categories, since the more confusing categories are prioritized higher. Note that this is a soft clustering, i.e., an item can appear in multiple categories, which is appropriate for our task, since one item could be confusing for multiple reasons—all such reasons are likely valuable to the requester.[3]

To compute item-item similarity, SPROUT first creates an item embedding. It uses all the text written by workers in the Resolve HITs and takes a TF-IDF-weighted linear combination of word embeddings in that text. Since the amount of text written by workers is relatively small, pre-trained word embeddings based on a Google News Corpus [120] suffice for this task—they capture similarities between semantically related words. SPROUT creates item-item similarity scores using the cosine distance between item embeddings.

7.2.3 Requester Dashboard for Improving Task Design

The SPROUT dashboard guides requesters performing a *Refine* step of the meta-workflow to efficiently identify important categories of confusion (Figure 7.4a), inspect individual items to gain deeper understanding (Figure 7.4b), and redesign the task (Figure 7.4c,d) in response.

Following visualization principles of "overview first and details on demand" [150], requesters can view all the top-level categories (largest, most confusing first) in the left column (Figure 7.4a), and expand categories to inspect individual items as needed. Category size is indicated next to the category name, along with a visualization of

[3]During pilot experiments, we also tried hierarchical clustering, but found the unprioritized, hard clustering to be less useful and coherent.

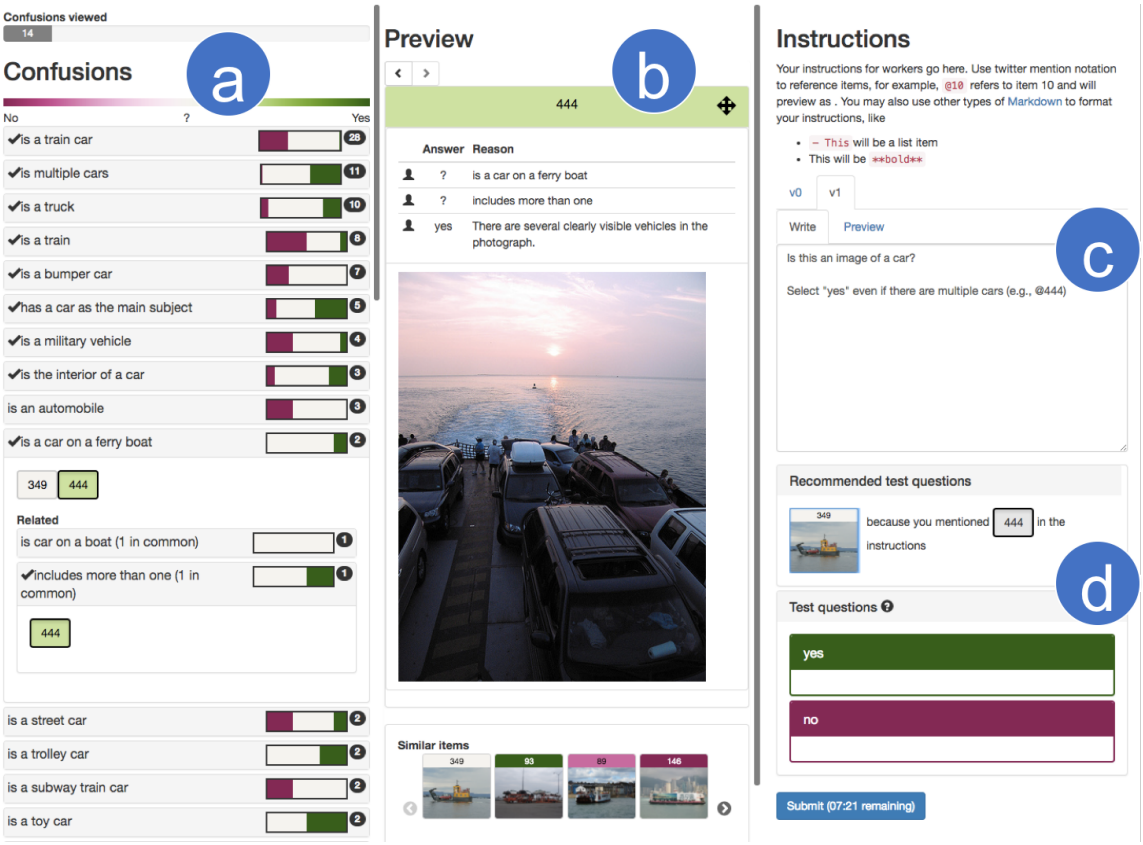


Figure 7.4: The SPROUT requester interface during a *Refine* step of the meta-workflow (Figure 7.1) for the *Cars* task. SPROUT enables requesters to (a) drill down into categories of ambiguous items, (b) view details of items (Item 444 shown), e.g., individual worker feedback (top) and similar items (bottom), (c) edit the instructions in response, and (d) create test questions, possibly from the set of recommended test questions (SPROUT recommended item 349 because it is similar to Item 444—an example the requester provided in the instructions—and thus a good candidate for testing worker understanding).

the distribution of workers who have answered *yes*, *no*, or *?*. Test question labels from *GenTest* steps are denoted as *yes* and *no*, while *Clarify* step labels are denoted as *?* (SPROUT discards labels from workers editing the instructions). Individual items within each category are represented by a button marked with the item id and colored with the mean worker answer. This compact item representation is inspired by Kulesza et al.'s [94] original structured-labeling implementation. The category and item visualizations use pink, white, and green to represent *no*, *?*, and *yes* worker answers, respectively.

Requesters can view additional details about individual items in the central preview column (Figure 7.4b), which is accessed by clicking on an item. The top of the preview shows worker responses from the Resolve HITs issued for the item. Below the preview, a panel shows thumbnails of similar items (sorted by descending similarity). These thumbnails are adapted from the original structured labeling implementation [94] for providing context about how to label an item.

The requester's instructions editor (Figure 7.4c) supports example creation and rich text editing. SPROUT lets requesters format their text using the Markdown markup language, and extends that language to support referencing items as examples using Twitter mention notation (e.g., `@12` will insert a reference to item 12). A preview tab lets the requester preview a formatted version of the text, with referenced items replaced by clickable item buttons.

To make an item into a test question, the requester simply drags it to the test questions panel (Figure 7.4d). Test questions are used for the gated instruction workflow that ensures that a new worker has understood the task (see next subsection). Clicking on the item in that panel opens a dialog box with a form that lets the requester edit the explanation. The form also provides guidance on best practices for writing test questions [38].

SPROUT suggests improvements to the task's training regimen in the form of test-question recommendations. Each time a requester references an example item using

the instructions editor, SPROUT recommends the most similar item (above a minimum similarity threshold) as a potential test question. These questions in general are good candidates for test questions because they are likely to reinforce or test understanding of the examples during the gated instruction workflow. In Figure 7.4d, the system has recommended an image of a boat carrying cars (item 349) since the requester had previously created an example of cars on a ferry (item 444).

As part of the overall workflow, requesters can quickly see which categories they have (and have not yet) inspected by the presence (or absence) of a checkmark to the left of the category name. SPROUT also provides a measure of the requester's overall progress toward viewing all confusing categories with a progress bar above the categories.

7.2.4 Compiling Gated Instructions

As the final part of the *Refine* step, SPROUT compiles the selected test questions into gated instructions [110] (see previous work and Figure 7.2 for details). SPROUT partitions the test questions for each label into two equal-sized sets for constructing the interactive tutorial and the gating questions (ensuring similar label distributions). These sets are subject to a maximum size, which is tunable and limits the duration of gated instruction. SPROUT uses any remaining test questions as gold standard questions to ensure workers remain diligent, using, e.g., a decision-theoretic gold question insertion strategy [16].

7.3 Exploratory User Study Design

We conducted a user study to validate our system design and inform the design of future tools for improving task design. Our evaluation is guided by three primary research questions:

RQ1: How useful do requesters find SPROUT's worker-powered interface for improving task design?

RQ2: How helpful are SPROUT's task improvement affordances (e.g., worker-powered structured navigation and test question suggestions)?

RQ3: How much improved are task designs produced with SPROUT?

Our research questions seek to validate our initial hypothesis that worker feedback helps task design by measuring requester attitudes and behaviors (RQ1, RQ2) and task design quality (RQ3). While it may seem obvious that feedback can help aspects of task design like wording of instructions, it is less clear that it will help other major task design bottlenecks like discovering and defining the nuances of the task. Thus, our evaluation compares SPROUT to structured labeling [94], a strong alternative to worker feedback that embodies best practices for exploring one's dataset and identifying important edge cases, which requesters can then add to the instructions [22].

To answer these research questions, we conducted a within-subjects laboratory study that allowed us to observe requesters using both SPROUT and structured labeling on two different task types, and to survey them on the relative benefits and weaknesses of the approaches. This study design let us get feedback from requesters with varying amount of crowdsourcing expertise, and control for interface condition and task type.

Our experiment design asks requesters to refine their task designs according to their *own* understanding of the concept, resulting in many different task definitions. We initially attempted to control for different requester concepts by fixing the concept up-front, but could find no way to communicate a fixed concept to requesters without interfering with the experiment. If one fully describes the concept to the requester up front, the requester could simply pass this description on to the workers, obviating much of the tool's benefit (finding ambiguities). We also tried having requesters interact during the experiment with an oracle that answers whether the requester has correctly labeled an item according to a fixed concept. However, in pilot studies, we found this interaction to be overly complex and unnatural. Ultimately, we decided to

let requesters specify their own concepts and account for this in our analysis. Previous studies of requester behavior [130] employed a similar design.

Following the user study, we evaluated the quality of the resulting instructions to help answer RQ3. SPROUT's objective and the truest measure of instruction quality is accuracy of the data generated by workers given those instructions. Unfortunately, we could not measure data accuracy because we do not have access to ground truth labels for comparison (each requester has their own, latent concept, which is only partially expressed through the instructions). In order to approximate data quality, we measured instruction quality directly by having two crowdsourcing experts[4] rank the instructions for each task, blind to tool condition, based on the number of ambiguous categories addressed in the instructions.[5] We asked the experts to base their rankings on this criterion, since reducing ambiguities is likely to improve data quality (assuming appropriate quality control measures are employed [110]),

In the rest of this section, we describe how we created the tasks and two requester interfaces, how we recruited participants for our study, and the details of our experimental procedure.

7.3.1 Labeling Tasks and Requester Interfaces

We selected two classification task types of different complexity: one image and one website (mix of text and images), inspired by prior work [22]. Since prior researchers did not release their tasks, we constructed two new equal-sized datasets:

- The *Cars* image dataset. We obtained this dataset by selecting images from all the ImageNet [74] synsets containing the word *car*, as described in prior work [22].

[4]The experts are researchers (one of whom is an author of [17]) who have each posted over 20 different crowdsourcing tasks and authored papers on the topic. Neither expert had previously seen the instructions.

[5]The experts determined categories independently by performing open coding [32] on the instructions.

- The *Travel* website dataset. In order to have a dataset of sufficient size for measuring instructions, we collected a new, expanded version of the dataset created by Kulesza et al. [94] from the DMOZ directory.[6] We found that sampling all pages from the *travel* category resulted in a sufficient number of ambiguous examples that it was not necessary to sample negative examples from other categories.

Our structured labeling implementation (Figure 7.5) operates completely independently of worker feedback and adapts key ideas from structured labeling. Requesters can label items by dragging them into *yes*, *no*, or *maybe* sections, organize items in groups within those sections, and name groups for easier recall. Figure 7.5 shows one requester's use of these sections during the user study. Our structured labeling implementation retains the instructions and test question editors of Sprout (Figure 7.4c,d), but provides space for requesters to organize items themselves (Figure 7.5) in place of the panel with categories created by workers (Figure 7.4a). We implemented structured labeling as closely as possible to the original paper, since the authors did not release their tool or source code.

We have released the tasks, a library for generating similar tasks, and the source code for Sprout and structured labeling for use by future researchers.[7] Our requester interface implementations are web applications built on the open-source React front-end library. We have provided demonstrations using both interfaces in the supplemental `sprout.mp4` video (Appendix B).

---

[6] We used an archived version of DMOZ [154], since the original DMOZ site is no longer active.

[7] `https://crowdlab.cs.washington.edu/task-design.html`

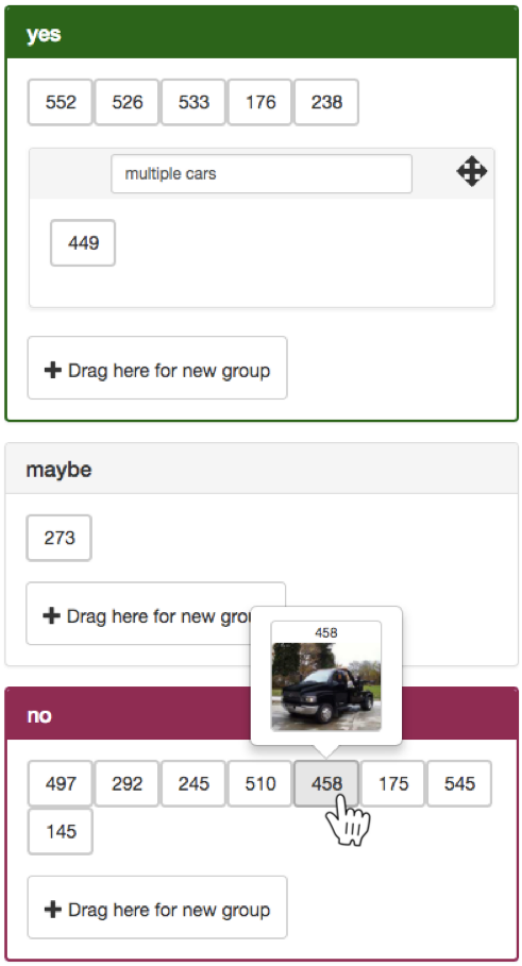


Figure 7.5: Our implementation of structured labeling [94], a method for labeling new tasks, which previous researchers have used to construct instructions [22]. Structured labeling supports concept evolution (one's changing definition of the task as one explores more data) by allowing the requester to defer labeling decisions with a *maybe* label, quickly change labels for groups of items (e.g, by dragging the "multiple cars" group to a different label), and name groups for fast recall. Here, a requester is hovering over an item (Item 458 here), which displays a thumbnail preview; a requester can also click on the item to view a larger preview (not pictured) or drag the item to a different label or group.

7.3.2 Participants

We recruited 11 participants[8] to use SPROUT and structured labeling as requesters, by emailing relevant mailing lists (with IRB approval). Our participants were graduate (10) and undergraduate (1) students at a major research university. Participants ranged in age from 21 to 45 and were balanced in terms of gender (5 male, 5 female, 1 other). Eight of the participants indicated prior experience as requesters, with four reporting having launched 1–3 different tasks and four reporting having launched 4–10 different tasks. Seven of the participants indicated some prior experience as crowd workers, but no participant indicated completing more than 10 HITs. Participants were paid $25 for approximately one hour of their time.

7.3.3 Procedure

We used a within-subjects experimental design. Each requester used SPROUT and structured labeling to improve the instructions for the two tasks, one per task. We assumed there was no learning effect across tasks, so we fixed the task order as *Cars* followed by *Travel*. We used a Latin square to randomize the order in which each requester encountered the interfaces. Before each task, requesters completed a tutorial and practice task (a dataset of confusing bird images) using the assigned interface in order to ensure they understood the goal and how to use the interface. After each task, requesters completed a brief survey about their experience, and after the last task, they also rated their preferred interface.

To ensure that the study completed within the allotted one hour (including tutorials and surveys), requesters were given 18 minutes per task to create an improved version of the instructions. We selected 300 items for each task as the evaluation set for the instructions, anticipating 300 to be sufficiently large such that (1) it would

[8]We excluded one participant who was not directly affiliated with the university mailing lists we advertised to, and who had difficulty understanding the objective and procedure of the experiment.

contain most of the common ambiguities in the task definition and (2) requesters would not be limited by the number of sample items during the experiment.[9]

Requesters were instructed to improve the initial instructions with the goal of having workers produce higher-accuracy answers that agree with the requester's concept. We used simple initial instructions for both datasets: "Is this an image of a car?" and "Is this a website about travel?" Accuracy depends on the task specification, and since each task is underspecified to start, each requester may arrive at a different target concept. Requesters who were unsure what their concept should be were prompted to pretend they were "launching a service for detecting cars" or "launching a website for travel tips." Requesters were also instructed to create at least three test questions that would ensure that workers understand their instructions.

In order to reduce the cost and variability of running the full Sprout workflow with every requester, we pre-collected worker (and meta-worker) answers for all questions in the evaluation sets by seeding Sprout with the initial versions of the instructions. These answers were replayed by Sprout during experiment sessions. We deployed the Resolve HIT to three workers for each of the 300 items in each task, paying $0.05 per image HIT and $0.07 per website HIT (based on a wage of $8 / hour calculated from pilot deployment timings). We limited participation to U.S. workers who had completed at least 100 HITs with an approval rate of at least 95%.

7.4 Results

7.4.1 RQ1: Usefulness of Worker Feedback

Figure 7.6a shows that requesters overall preferred Sprout over structured labeling. P1 summarized the high-level benefits of worker feedback with Sprout: "[c]ategorizing the inputs, showing me the cases where there was confusion, etc., made it SUPER easy to identify cases that needed clarification." In contrast, structured

[9]Experimentation during pilot studies showed that 18 minutes was a reasonable upper bound and 300 items was sufficiently large.

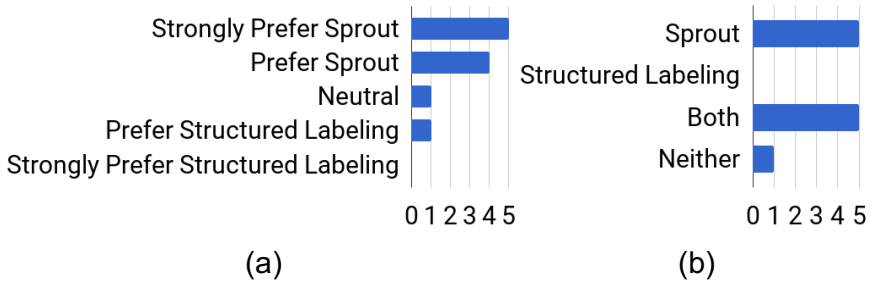


Figure 7.6: Requester responses to the questions (a) "Which interface did you prefer for creating instructions?" and (b) "Which interface would you use to create instructions in the future?". Requesters in our study overall preferred SPROUT with worker feedback rather than in structured labeling, but about half still saw uses for both interfaces.

labeling didn't provide "a sense of what the data looked like as a whole." Another requester (P4) expressed dismay after switching from SPROUT to structured labeling: "It sucks that you have to start from a completely blank slate. [SPROUT] gave you some more support."

While most requesters preferred SPROUT, about half indicated that they would use both interfaces in the future (Figure 7.6b), indicating that worker feedback is not always desired. P3 wrote "Before I had any crowd data, I liked [structured labeling] because it let me try doing the task myself...But after I had some preliminary labeled data, I would like to use [SPROUT] to see what kinds of things people were confused about." P2 wanted to use structured labeling to start to avoid being biased by workers when deciding on her task concept. P11 expressed mixed feelings about structured labeling; she liked that it "forced [her] to personally think through what was in the image and what [she] was looking for," but also said that "it was...pretty time consuming to create the different categories."

Requesters did not find the organizational aspects of structured labeling particularly useful and rarely created groups, likely because the instructions editor itself serves as an organizational tool that lets requesters describe categories with example items. Usage might change with more extended interaction; P1 noted "[groups] might be [useful] if [he] were iterating multiple times and wanted to come back...in the future."

The one requester who indicated he would use neither interface in the future (P1) did so because he prefers crafting examples by hand as a domain expert in natural language processing, explaining "[a]lthough SPROUT does help with quickly identifying the confusing cases...to make the instructions concise I typically have to come up with my own examples anyway (so I can reuse the same example sentence for a lot of examples)."

7.4.2 RQ2: Usefulness of SPROUT*'s Affordances*

Requesters found the recommended test questions particularly useful. One requester (P2), an NLP researcher, wrote "The most helpful features...is (sic) the automatically suggested test questions. The similarity metrics seems (sic) to be working great and the suggested items are great for testing the points I emphasized in the instructions." Overall, a significant fraction of test questions created by requesters were previously recommended by SPROUT (on average, 29% and 20% for the *Cars* and *Travel* tasks, respectively). Requesters reported them useful in surveys, and several created test questions almost exclusively based on recommendations (4/5 test questions by P1 on the *Cars* task and 4/4 test questions by P2 on the *Travel* task). Still, semantically similar items are not always the best test questions; P11 complained, "if I used a ferry with cars on it as an example, it'd just return a boat as a suggestion."

Requesters used the similar items panel infrequently, perhaps due to lack of familiarity with the interface or because SPROUT already recommends the most similar item to each example in the instructions as a test question.

While most requesters looked at individual items first, P9 scanned the text descriptions of the categories and began to edit the instruction text without inspecting items. This type of rapid instruction improvement was made possible by the organized presentation of worker feedback. We did not instruct requesters about such strategies, and other requesters may have learned to use Sprout more effectively with instruction.

7.4.3 RQ3: Impact on Task Design Quality

On average, requesters using Sprout wrote longer instructions, cited more examples, and were rated higher by experts.[10]

Requesters using Sprout on average wrote longer instructions ($\mu = 1672$ vs. 1110 characters) and cited nearly twice as many examples on the *Travel* task compared to structured labeling ($\mu = 4.6$ vs. 2.6 examples). These results are weakly significant based on a two-sided Welch's t-test ($t = 1.89$, $p = 0.10$; $t = 1.77$, $p = 0.11$). There was no significant difference on the *Cars* task ($t = -0.30$, $p = 0.78$; $t = -0.09$, $p = 0.93$).

Two crowdsourcing experts independently ranked the instructions for each task into five quality buckets (valued 1 to 5), assigning higher values to instructions that mentioned more categories of ambiguous items. Both experts ranked instructions produced using Sprout higher on the *Cars* task compared to structured labeling ($\mu = 4.0$ vs. 2.8; $\mu = 3.2$ vs. 2.0). These results are weakly significant based on a two-sided Welch's t-test ($t = 1.78$, $p = 0.11$; $t = 1.88$, $p = 0.11$). There was no significant difference on the *Travel* task for either expert rating ($t = 0.0$, $p = 1.0$; $t = 0.0$, $p = 1.0$).

[10]Larger samples are needed to establish statistical significance. We excluded P5 from analysis of the *Travel* task, since she was unable to complete the task due to difficulty finding a motivation for the task.

Together, these results suggest that Sprout helps create more comprehensive instructions. Detailed instructions can elicit higher quality data from workers when combined with proper training and screening methods like gated instructions [110].

7.5 Design Implications for Task Design

7.5.1 How to make exploration frictionless?

It is essential that future task design tools help requesters view items with minimal overhead. In structured labeling, even dragging items into *yes* / *no* / *maybe* sections was inefficient for some requesters, who found it faster to scroll through the carousel of item thumbnails. P1 complained that even this carousel had too much friction and the images were too small. A more efficient view might have been a vertical scroll (used by websites like Instagram) with an option to resize images, though factors such as scrolling direction and number of items per page can have subtle effects on performance [84, 85]. One requester (P8) seemed to experience similar friction with Sprout, repurposing the similar items (bottom of Figure 7.4b) for quickly retrieving new items (not just similar ones).

Sprout was designed with the idea that one must explore items before writing instructions. However, one can also view exploration in service of the ultimate goal of creating instructions. From this viewpoint, P10 felt that the instructions editor would be more natural on the left. Other requesters felt that Sprout could have benefited from more "visual hierarchy" (P8) and "linear process" (P3).

7.5.2 How much information to show?

Another design decision to consider is how much and when to show information about worker confusions to the requester. P5 felt overwhelmed by the number of categories displayed and began clicking through categories without regard to their prioritization. In contrast, P9 benefited from having all categories displayed initially,

as his strategy was to read the descriptions, edit the instructions, and only look at items periodically. Providing the right amount of support for a diverse set of users is a challenging problem for mixed-initiative systems. One possibility is to make the frictionless vertical scroll described above the default mode, and enable the requester to use additional features from Sprout and structured labeling on demand as they learn what tools most benefit their personal workflow.

7.5.3 How much initiative to take?

Another possibility is to create an adaptive version of Sprout that shows the requester a sequence of only the most important and diverse categories, taking into account the set of items the requester has viewed and the instructions she has written up to that point. More knowledge about the space of items considered by the requester so far could also enable smarter suggestions for improvements to the instructions.

While our tool supports test question creation by the requester, it is unclear whether the requester need be the one to do so, once she has written a clear set of instructions with examples. Indeed, our high-level vision (Figure 7.1) is that workers (or the system) should be able to determine when the requester's input is needed. This suggests that other tedious task components, such as gold questions or task advertisements, might be created automatically by meta-workers, saving requester time. And while P1 wanted Sprout to help him ensure the distribution of test question labels matched the overall distribution of labels—as recommended by CrowdFlower for gold questions [38]—such tools may not be necessary either.

7.5.4 How to balance self-organization and worker support?

While the instructions themselves can be used for organizational purposes, additional support for self-organization in Sprout could have been helpful. For instance, P9 had trouble recalling and finding a category of items he had previously read (but not

opened, so it did not have a check mark). P4 on the other hand wanted to incorporate some of the intelligence of Sprout into structured labeling by automatically narrowing the set of items classified as *maybe* using structured labeling (e.g., using item vector embeddings), when items in that section are covered by the instructions. Finding a middle ground and making transparent what the workers have done vs. what the requester has done is challenging but could pay dividends.

7.5.5 How much of the workflow to support?

Even if one starts out with a binary-classification problem, one can realize down the line that the task might benefit from decomposition or structured output of a different kind. Two of our requesters familiar with crowdsourcing for NLP (P1, P2) wanted to change the task interface, for instance by changing the answer labels, or to break the task into smaller subtasks "since it is easier to write instructions for small tasks." Supporting these aspects of task design are a worthy goal and could benefit from Sprout's feedback loop; we chose to focus on binary labeling tasks since (a) they are the most common crowdsourcing task type [72] and (b) we wanted to constrain the design space to avoid some of the pitfalls of previous work that tackled broader problems [95].

In addition to supporting more aspects of task design, we believe that future versions of Sprout could naturally support additional types of tasks. Labeling tasks with more than two answers are possible with minor interface changes, and more generally, we envision Sprout being useful for any task with many different instances (questions) that share a common design (so that improvement benefits many questions) and have a correct answer (so workers can be tested before beginning the task), for example information-seeking tasks [130].

7.5.6 How to scaffold the process of learning to design tasks?

Our requesters were largely unpracticed at writing high-quality crowdsourcing instructions. There is opportunity to incorporate tools for helping requesters—such as guiding them toward effective strategies—into Sprout. P4 mentioned it would helpful to have templates for common things to say to workers like "use your best judgement." P1 thought better support for task criteria labels would be useful for overall consistency. We agree that these would be great to add, but determining a single set of best practices is difficult. It may also be possible to train crowd workers instead to hone the presentation of the task.

7.6 Limitations

Several additional types of evaluation would strengthen our findings. Future studies should strive to measure both data accuracy and worker satisfaction resulting from task improvement. More studies are also needed to investigate what happens when requesters spend more time with the tools and instructions become very complex, and to demonstrate that our findings generalize to many types of requesters.

While our study design allowed for controlled observation of requester behavior, we also encountered several experimental challenges. Several requesters experienced difficulty deciding on a motivating concept to help them make labeling decisions, causing large delays (P5) or concept changes just to simplify the task (P11); providing tools to requesters solving their own problems may improve motivation. Future studies could also seek to better control for the amount of time requesters spend on each task (some requesters ended the task early), or the style of instructions (e.g., by providing more requester training). Finally, studying multiple task types was informative but decreased statistical power; future studies could try other experimental designs (e.g., with task type as a blocking factor).

7.7 Discussion and Future Work

To achieve high-quality output from crowdsourcing, one requires diligent workers working on well-designed, and clearly-explained tasks. While there are many papers on identifying diligent workers and substantial research on patterns for task decomposition, our work is perhaps the first tool that helps requesters design effective instructions. Instructions may be less glamorous than some other aspects of crowdourcing, but they have been shown to be deeply important [110].

Furthermore, Sprout uses a novel method to aid instruction design and debugging: having the crowd evaluate the current design on a sample of data, identifying confusing questions based on disagreement and worker diagnoses, clustering confusion categories based on worker instruction edits, and showing those in an organized and prioritized manner so that a requester can quickly learn the various nuances of their task and its current flaws. Sprout further aids a requester by providing a natural interface for improving instructions with embedded illustrative examples and recommending test questions for a gated instruction workflow that ensures worker understanding.

Nearly all the requesters in our user study (with varying amounts of crowdsourcing expertise) preferred to use Sprout (which has worker feedback) over structured labeling, a natural baseline that supports requesters learning about their task themselves rather than through worker feedback. Some requesters felt that structured labeling is a good interface for creating the first set of instructions, but overall they preferred the full power of Sprout, which makes effective task design more convenient. On average, instructions produced using Sprout were longer, cited more examples, and were rated higher by multiple crowdsourcing experts. This user study also led to our set of design implications for future task design, and we have released our source code and web-based implementations for further use by requesters and researchers.

These results demonstrate that a self-improving crowdsourcing agent can improve over existing practices by involving workers, and better supporting the requester. While true self-improvement might involve even less requester effort, it may not be feasible (or desired) to fully remove the requester from the loop, since only the requester may be able to make certain decisions about the task specification. Our experience with SPROUT suggests that engaging workers in work related to task improvement is fruitful, and that future researchers can further push the limits of combining algorithms and meta-workers with the goal of improving task outcomes.

In the future, we plan to use the crowd to improve other aspects of task and workflow design, such as task decomposition, and to support task design beyond labeling tasks. For example, we envision crowd workers retrieving task details from requesters as needed and collaboratively developing even better designs with minimal requester effort. We encourage other researchers to continue to explore new ways to leverage and develop worker task design skills, and to build systems that mediate worker-requester communications.

Chapter 8
CONCLUSIONS AND FUTURE WORK

8.1 Summary of Contributions

This dissertation examines methods of combining human and machine intelligence to improve work quality and lower cost. It provides evidence that high-quality, efficient crowdsourcing tasks *can* be created at low cost and with reduced requester effort. In order to achieve this goal, this dissertation proposes *self-improving crowdsourcing*, a new paradigm that reduces the burden of designing performant crowdsourcing tasks, by combining the efforts of algorithms, the requester, and crowdsourcing workers to achieve this goal.

In order to demonstrate that combining algorithms, the requester, and workers can significantly aid crowdsourcing task creation, I designed algorithms, built tools, and conducted experiments demonstrating the effectiveness of these approaches for optimizing many aspects of crowdsourcing task design. These methods use algorithms to optimize many parts of the crowdsourcing task design pipeline, from recruiting and managing workers (Chapters 3– 4) to routing and prioritizing tasks (Chapters 5– 6). In Chapter 7, I present a tool that closes the feedback loop and involves the requester and workers, too, in task self-improvement. As a whole, this body of work reduces the effort of creating and optimizing high-quality crowdsourcing tasks.

In pursuing this work, I have demonstrated various novel ways to combine algorithms, the requester, and workers performing crowdsourcing self-improvement. Chapter 7 describes extensive participation by the requester and workers, facilitated by algorithms that cluster items for requester review and semi-automate instruction improvement. This collaboration produces new *actions* (instructions and training /

test questions) that enable the agent (system) to collect answers that better match the requester's desired output. Similarly, Chapter 3 presents experiments with request designs that provide the agent with new actions to recruit workers more effectively; requesters (or even workers) could supply these actions as well. Chapter 3 presents feature engineering experiments that improve the agent's *model* of the environment and enable more efficient recruiting; the requester could similarly engineer domain-specific features to benefit any part of task pipeline. Finally, in Chapter 4, I demonstrate how an initial *policy* specified by the requester can provide a useful starting point for improvement by encouraging safe exploration with reinforcement learning.

Further, I have shown that myopic methods and compact model representations work well for optimizing many parts of the task design pipeline. I formulated the problems of recruiting workers, as well as routing and prioritizing tasks, as submodular optimization problems, which can be solved near-optimally using greedy algorithms that only perform a one-step lookahead. Worker management through testing and training benefits from a longer lookahead; Chapter 5 presents tractable methods for performing this lookahead by modeling a small number of worker classes and the most important action classes, enabling the problem to be solved using standard POMDP solution techniques.

8.2 Takeaways for Crowdsourcing System Designers

- Requesters and workers need not be limited to their traditional roles. For example, requesters can transition from a primary role as the designer to a supporting role for self-optimizing algorithms (e.g., by designing new recruiting actions for use by the algorithms). Similarly, workers can perform task-improvement roles (e.g., by suggesting edits to the instructions) in addition to answering standard task questions.

- *Combining* the efforts of requesters and workers for task creation can produce better outcomes with less requester effort. Requesters and workers have complementary abilities. Only the requester can make final determinations about what constitutes a "correct" answer. Workers, on the other hand, are more numerous and have first-hand experience about what confuses them about the task. This dissertation demonstrates that leveraging the diversity of worker opinions can help requesters specify and improve tasks.

- Task design and optimization are worthwhile investments that can have a major impact on results. For example, my algorithm for optimizing multi-label classification resulted in cost reductions of over 90%. Many crowdsourcing practitioners do not use optimization methods designed by researchers [118], but they should! Moreover, failure to invest sufficiently in task design can lead to poor results and even incorrect scientific conclusions [110]. Good task design is difficult; tools like my SPROUT meta-workflow can help.

- Careful modeling can help to overcome challenges to designing algorithms for crowd work, which often has large possible state and action spaces (e.g., all question-worker combinations) and no clear reward signal. My work on recruiting workers, and also routing and prioritizing tasks, overcomes problems with large search space by formulating submodular optimization objectives, which can be solved near-optimally using myopic algorithms. On the other hand, my work on managing workers uses model-based RL with constrained state and action spaces to make learning non-myopic policies tractable. To deal with the lack of a reward signal, that work also uses a proxy reward measure (test question performance) to adapt to workers.

8.3 Future Work

Systems that support collective human-machine collaboration represent an exciting opportunity to achieve more than is possible with either humans or machines alone. Research in this area is likely to result in additional contributions to machine learning, decision-making under uncertainty, and human-computer interaction (HCI). On the AI side, while I have found that myopic strategies and model-based RL work well for a variety of problems, more research is needed to improve learning with sparse rewards and in complex environments without access to a simulator, and to elicit preferences efficiently and scalably. Such methods are needed to further develop self-improving crowdsourcing methods, which must work even when most questions do not have known answers, humans cannot be simulated, and tasks are not fully specified. On the HCI side, future work will require discovering novel design patterns for human-human and human-machine workflows, as well as better understanding of the benefits and limitations of human-machine cooperation.

One vision for future work building on this dissertation is to build a system capable of taking a high-level goal specified in natural language, coordinating and supporting a network of heterogeneous human and machine agents working toward that goal [13, 14, 16, 48, 71, 124, 162], and communicating as needed to refine the goal (Chapter 7). Like the work in this dissertation, future work will benefit from an interdisciplinary approach that combines AI and HCI. HCI methods help to understand the needs of humans and invent mechanisms for interaction between humans and machines. At the same time, it is useful to consider how optimization fits into these goals, and these considerations often lead to advances in AI.

I will describe three problem areas that could bring us closer to being able realize this vision.

8.3.1 Expanded Worker Self-Improvement Roles

My SPROUT task improvement system focuses on supporting the task designer making decisions about how workers should respond to questions that are ambiguous given the initial set of instructions. It is possible that workers can take on more active roles in improving the task design. The requester is the only true source of ground truth, but workers, for instance, may be able to infer how to resolve certain ambiguities given previous information provided by the requester. In order to ensure that these inferences actually align with decisions the requester would have made, it may be useful to have contributors deliberate [48] so that all decisions are well-reasoned. Another important and open area of self-improvement that was not addressed in this thesis, which workers may be able to contribute to, is making interface improvements. Workers could also play more direct roles in training other workers, e.g., through apprenticeships [156].

8.3.2 Extensions to Subjective and Complex Tasks

This dissertation demonstrates the feasibility of self-improving crowdsourcing primarily in the domain of labeling tasks. In order to combat the devolution of crowdsourcing into minimum-wage piecework and to help realize more complex goals, it is essential to demonstrate that self-improving crowdsourcing is possible for more types of tasks, including subjective or creative tasks (e.g., writing an essay [86]) and tasks that require complex solution strategies (e.g., web search challenges [46]). Certain parts of this dissertation, e.g., recruiting methods (Chapter 3), are immediately useful for such tasks. However, methods presented for other problems exploit properties of labeling tasks, which enable automatically comparing a worker's answer to a reference answer (Chapter 4), aggregating worker answers (Chapters 5 and 6), and determining worker agreement (Chapter 7). Adapting these methods to other types of tasks may require

inventing new ways to perform these types of functions for tasks with less explicit structure.

8.3.3 Handing off Problems to AI Agents

As crowdsourcing for a task proceeds, it can be more efficient for machines to eventually perform all or part of the task in place of human workers. Managing this handoff is an important area of future work. Machines will require assessment before they can replace routine tasks performed by humans. Traditional assessment techniques, such as using held-out test sets, require gathering vast amounts of data to detect uncommon errors and risk experimental biases with data reuse. An alternative approach is to assist human debugging—more in line with software engineering best practices like test-driven development, developed over decades of deploying robust systems. Researchers can build tools to dramatically speed up human debugging of agents by encoding useful error-finding strategies. One source of inspiration for such tools is the automated transformation macros shared by the FoldIt community, which aid searching through large solution spaces [30]. Improved tools for finding these errors may also help to create challenge datasets that lead to better models [137, 138]. Current approaches seek to construct adversarial examples using fully automated approaches, but involving humans would help find more realistic errors.

BIBLIOGRAPHY

[1] Maneesh Agrawala and Chris Stolte. Rendering Effective Route Maps: Improving Usability Through Generalization. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 241–249, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi:10.1145/383259.383286.

[2] Harini Alagarai Sampath, Rajeev Rajeshuni, and Bipin Indurkhya. Cognitively Inspired Task Design to Improve User Performance on Crowdsourcing Platforms. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 3665–3674, Toronto, Ontario, Canada, 2014. ACM. ISBN 978-1-4503-2473-1. doi:10.1145/2556288.2557155.

[3] Omar Alonso and Stefano Mizzaro. Using Crowdsourcing for TREC Relevance Assessment. *Information Processing and Management*, 48(6):1053–1066, November 2012. ISSN 0306-4573. doi:10.1016/j.ipm.2012.01.004.

[4] Gabor Angeli, Julie Tibshirani, Jean Wu, and Christopher D. Manning. Combining Distant and Partial Supervision for Relation Extraction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP '14, pages 1556–1567. ACL, 2014. URL `http://aclweb.org/anthology/D/D14/D14-1164.pdf`.

[5] Josh Attenberg, Panagiotis G. Ipeirotis, and Foster J. Provost. Beat the Machine: Challenging Workers to Find the Unknown Unknowns. In *Human Computation, Papers from the 2011 AAAI Workshop, San Francisco, California,*

USA, August 8, 2011, volume WS-11-11 of *HCOMP '11*. AAAI, 2011. URL http://www.aaai.org/ocs/index.php/WS/AAAIW11/paper/view/3954.

[6] Joshua Attenberg, Panos Ipeirotis, and Foster Provost. Beat the Machine: Challenging Humans to Find a Predictive Model's "Unknown Unknowns". *J. Data and Information Quality*, 6(1):1:1–1:17, March 2015. ISSN 1936-1955. doi:10.1145/2700832.

[7] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group Formation in Large Social Networks: Membership, Growth, and Evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 44–54, Philadelphia, PA, USA, 2006. ACM. ISBN 1-59593-339-5. doi:10.1145/1150402.1150412.

[8] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast Algorithms for Maximizing Submodular Functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1497–1514, Portland, Oregon, 2014. Society for Industrial and Applied Mathematics. ISBN 978-1-61197-338-9. URL http://dl.acm.org/citation.cfm?id=2634074.2634184.

[9] Sumit Basu and Janara Christensen. Teaching Classification Boundaries to Humans. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*, AAAI '13. AAAI Press, 2013. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6359.

[10] Gerard Beenen, Kimberly Ling, Xiaoqing Wang, Klarissa Chang, Dan Frankowski, Paul Resnick, and Robert E. Kraut. Using Social Psychology to Motivate Contributions to Online Communities. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04,

pages 212–221, Chicago, Illinois, USA, 2004. ACM. ISBN 1-58113-810-5. doi:10.1145/1031607.1031642.

[11] Yoshua Bengio and Paolo Frasconi. An Input Output HMM Architecture. In *Advances in Neural Information Processing Systems 7*, NIPS '94, pages 427–434. MIT Press, 1994. URL http://papers.nips.cc/paper/964-an-input-output-hmm-architecture.

[12] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: A Word Processor with a Crowd Inside. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 313–322, New York, New York, USA, 2010. ACM. ISBN 978-1-4503-0271-5. doi:10.1145/1866029.1866078.

[13] Jonathan Bragg, Mausam, and Daniel S. Weld. Crowdsourcing Multi-Label Classification for Taxonomy Creation. In *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing*, HCOMP '13. AAAI, 2013. URL http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7560.

[14] Jonathan Bragg, Andrey Kolobov, Mausam, and Daniel S. Weld. Parallel Task Routing for Crowdsourcing. In *Proceedings of the Second AAAI Conference on Human Computation and Crowdsourcing*, HCOMP '14. AAAI, 2014. URL http://www.aaai.org/ocs/index.php/HCOMP/HCOMP14/paper/view/8981.

[15] Jonathan Bragg, Mausam, and Daniel S Weld. Learning on the Job: Optimal Instruction for Crowdsourcing. In *ICML '15 Workshop on Crowdsourcing and Machine Learning*, 2015. URL http://crowdml.cc/icml2015/papers/CrowdML-Paper17.pdf.

[16] Jonathan Bragg, Mausam, and Daniel S. Weld. Optimal Testing for Crowd Workers. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, AAMAS '16, pages 966–974, Singapore, Singapore, 2016. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-4239-1. URL `http://dl.acm.org/citation.cfm?id=2937029.2937066`.

[17] Jonathan Bragg, Mausam, and Daniel S. Weld. Sprout: Crowd-Powered Task Design for Crowdsourcing. In *Proceedings of the 31st ACM User Interface Software and Technology Symposium*, UIST '18, 2018. To appear.

[18] Moira Burke, Robert Kraut, and Elisabeth Joyce. Membership claims and requests: Conversation-level newcomer socialization strategies in online groups. *Small group research*, 2009.

[19] Moira Burke, Cameron Marlow, and Thomas Lento. Feed Me: Motivating Newcomer Contribution in Social Network Sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 945–954, Boston, MA, USA, 2009. ACM. ISBN 978-1-60558-246-7. doi:10.1145/1518701.1518847. Series Title: CHI '09.

[20] Brian S Butler. When is a group not a group: An empirical examination of metaphors for online social structure. *Social and Decision Sciences*, 1999.

[21] Chris Callison-Burch and Mark Dredze. Creating Speech and Language Data with Amazon's Mechanical Turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010.

[22] Joseph Chee Chang, Saleema Amershi, and Ece Kamar. Revolt: Collaborative Crowdsourcing for Labeling Machine Learning Datasets. In *Proceedings of*

the 2017 CHI Conference on Human Factors in Computing Systems, CHI '17, pages 2334–2346, Denver, Colorado, USA, 2017. ACM. ISBN 978-1-4503-4655-9. doi:10.1145/3025453.3026044.

[23] Xi Chen, Qihang Lin, and Dengyong Zhou. Optimistic Knowledge Gradient Policy for Optimal Budget Allocation in Crowdsourcing. In *ICML*, 2013.

[24] Justin Cheng, Jaime Teevan, and Michael S. Bernstein. Measuring Crowdsourcing Effort with Error-Time Curves. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, pages 1365–1374, 2015. ISBN 978-1-4503-3145-6. doi:10.1145/2702123.2702145.

[25] Ed H. Chi and Todd Mytkowicz. Understanding the Efficiency of Social Tagging Systems Using Information Theory. In *Proceedings of the Nineteenth ACM Conference on Hypertext and Hypermedia*, HT '08, pages 81–88, Pittsburgh, PA, USA, 2008. ACM. ISBN 978-1-59593-985-2. doi:10.1145/1379092.1379110.

[26] Min Chi, Pamela Jordan, Kurt VanLehn, and Moses Hall. Reinforcement Learning-based Feature Selection For Developing Pedagogically Effective Tutorial Dialogue Tactics. In *Educational Data Mining*, 2008.

[27] Lydia B. Chilton, Greg Little, Darren Edge, Daniel S. Weld, and James A. Landay. Cascade: Crowdsourcing Taxonomy Creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1999–2008, Paris, France, 2013. ACM. ISBN 978-1-4503-1899-0. doi:10.1145/2470654.2466265.

[28] Robert B Cialdini and Noah J Goldstein. Social influence: Compliance and conformity. *Annual review of psychology*, 2004.

[29] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beene, Andrew Leaver-Fay, David Baker, Zoran Popovic, and Foldit

players. Predicting protein structures with a multiplayer online game. *Nature*, 446:756–760, 2010.

[30] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popović, and Foldit Players. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010. ISSN 0028-0836. doi:10.1038/nature09304.

[31] Albert T Corbett and John R Anderson. Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. *User Modeling and User-Adapted Interaction*, 4:253–278, 1995.

[32] Juliet Corbin and Anselm Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, 2014. ISBN 9781483315683.

[33] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, and John Riedl. Is Seeing Believing?: How Recommender System Interfaces Affect Users' Opinions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 585–592, Ft. Lauderdale, Florida, USA, 2003. ACM. ISBN 1-58113-630-7. doi:10.1145/642611.642713.

[34] Dan Cosley, Dan Frankowski, Loren Terveen, and John Riedl. Using Intelligent Task Routing and Contribution Review to Help Communities Build Artifacts of Lasting Value. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 1037–1046. ACM, 2006. ISBN 1-59593-372-7. doi:10.1145/1124772.1124928.

[35] Dan Cosley, Dan Frankowski, Loren Terveen, and John Riedl. SuggestBot: Using Intelligent Task Routing to Help People Find Work in Wikipedia. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*,

IUI '07, pages 32–41, Honolulu, Hawaii, USA, 2007. ACM. ISBN 1-59593-481-2. doi:10.1145/1216295.1216309.

[36] CrowdFlower, Inc. Job Launch Checklist. URL https://success.crowdflower.com/hc/en-us/articles/202703195-Job-Launch-Checklist.

[37] Crowdflower, Inc. Ideal Jobs for Crowdsourcing, 2017. URL https://success.crowdflower.com/hc/en-us/articles/202703295-Ideal-Jobs-for-Crowdsourcing.

[38] CrowdFlower, Inc. Test Question Best Practices, 2017. URL https://success.crowdflower.com/hc/en-us/articles/213078963-Test-Question-Best-Practices.

[39] Peng Dai, Mausam, and Daniel S. Weld. Decision-Theoretic Control of Crowd-Sourced Workflows. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI '10. AAAI Press, 2010. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1873.

[40] Peng Dai, Christopher H. Lin, Mausam, and Daniel S. Weld. POMDP-Based Control of Workflows for Crowdsourcing. *Artificial Intelligence*, 202:52–85, 2013. doi:10.1016/j.artint.2013.06.002.

[41] Patrick M. de Boer and Abraham Bernstein. PPLib: Towards the Automated Generation of Crowd Computing Programs using Process Recombination and Auto-Experimentation. *ACM Transactions on Intelligent Systems and Technology*, 7(Special Issue: Crowd in Intelligent Systems), 2016. ISSN 21576912. doi:10.1145/2897367.

[42] Patrick M. de Boer and Abraham Bernstein. Efficiently Identifying a Well-Performing Crowd Process for a Given Problem. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work*

and Social Computing - CSCW '17, pages 1688–1699, 2017. ISBN 9781450343350. doi:10.1145/2998181.2998263. URL `http://dl.acm.org/citation.cfm?doid=2998181.2998263`.

[43] Chrysanthos Dellarocas and M Zhang. Using online ratings as a proxy of word-of-mouth in motion picture revenue forecasting. Technical report, 2005.

[44] Pinar Donmez, Jaime G Carbonell, and Jeff Schneider. Efficiently Learning the Accuracy of Labeling Sources for Selective Sampling. In *KDD*, 2009.

[45] Pinar Donmez, Jaime G Carbonell, and Jeff G Schneider. A Probabilistic Framework to Learn from Multiple Annotators with Time-Varying Accuracy. In *Proceedings of the SIAM International Conference on Data Mining (SDM '10)*, pages 826–837, 2010.

[46] Shayan Doroudi, Ece Kamar, Emma Brunskill, and Eric Horvitz. Toward a Learning Science for Complex Crowdsourcing Tasks. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 2623–2634, San Jose, California, USA, 2016. ACM. ISBN 978-1-4503-3362-7. doi:10.1145/2858036.2858268.

[47] Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. Shepherding the crowd yields better work. In *CSCW*, pages 1013–1022, New York, New York, USA, 2012. ACM Press. ISBN 978-1-4503-1086-4. doi:10.1145/2145204.2145355.

[48] Ryan Drapeau, Lydia B. Chilton, Jonathan Bragg, and Daniel S. Weld. MicroTalk: Using Argumentation to Improve Crowdsourcing Accuracy. In *Proceedings of the Fourth AAAI Conference on Human Computation and Crowdsourcing*, HCOMP '16, pages 32–41. AAAI Press, 2016. URL `http://aaai.org/ocs/index.php/HCOMP/HCOMP16/paper/view/14024`.

[49] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA, 2010. ISBN 978-0-521-19533-1.

[50] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *LREC '06*, 2006.

[51] Yuval Filmus and Justin Ward. A Tight Combinatorial Algorithm for Submodular Maximization Subject to a Matroid Constraint. In *53rd Annual IEEE Symposium on Foundations of Computer Science*, FOCS '12, pages 659–668. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.55.

[52] Marshall L Fisher, George L Nemhauser, and Laurence A Wolsey. An Analysis of Approximations for Maximizing Submodular Set Functions-II. *Mathematical Programming Study*, 8:73–87, 1978.

[53] Jonathan L Freedman and Scott C Fraser. Compliance without pressure: The foot-in-the-door technique. *Journal of personality and social psychology*, 1966.

[54] Ujwal Gadiraju, Jie Yang, and Alessandro Bozzon. Clarity is a Worthwhile Quality: On the Role of Task Clarity in Microtask Crowdsourcing. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media*, HT '17, pages 5–14, Prague, Czech Republic, 2017. ACM. ISBN 978-1-4503-4708-2. doi:10.1145/3078714.3078715.

[55] Snehalkumar (Neil) S. Gaikwad, Mark E. Whiting, Dilrukshi Gamage, Catherine A. Mullings, Dinesh Majeti, Shirish Goyal, Aaron Gilbee, Nalin Chhibber, Adam Ginzberg, Angela Richmond-Fuller, Sekandar Matin, Vibhor Sehgal, Tejas Seshadri Sarma, Ahmed Nasser, Alipta Ballav, Jeff Regino, Sharon Zhou, Kamila Mananova, Preethi Srinivas, Karolina Ziulkoski, Dinesh Dhakal, Alexander Stolzoff, Senadhipathige S. Niranga, Mohamed Hashim Salih, Ak-

shansh Sinha, Rajan Vaish, and Michael S. Bernstein. The Daemo Crowdsourcing Marketplace. In *Companion of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17 Companion, pages 1–4, Portland, Oregon, USA, 2017. ACM. ISBN 978-1-4503-4688-7. doi:10.1145/3022198.3023270.

[56] Krzysztof Gajos and Daniel S. Weld. SUPPLE: Automatically Generating User Interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, IUI '04, pages 93–100, Funchal, Madeira, Portugal, 2004. ACM. ISBN 1-58113-815-6. doi:10.1145/964442.964461.

[57] Yihan Gao and Aditya Parameswaran. Finish Them!: Pricing Algorithms for Human Computation. *Proc. VLDB Endow.*, 7(14):1965–1976, October 2014. ISSN 2150-8097. doi:10.14778/2733085.2733101.

[58] Elena L Glassman, Jeremy Scott, Rishabh Singh, Philip J. Guo, and Robert C. Miller. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale. *ACM Transactions on Computer-Human Interaction*, 22(2): 1–35, 2015. ISSN 10730516. doi:10.1145/2699751.

[59] Scott A Golder and Bernardo A Huberman. Usage Patterns of Collaborative Tagging Systems. *Journal of Information Science*, 32(2):198–208, 2006. doi:10.1177/0165551506062337.

[60] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, (2005):1–53, 2011.

[61] Matthew R Gormley, Adam Gerber, Mary Harper, and Mark Dredze. Non-expert correction of automatically generated relation annotations. In *NAACL*

Workshop on Creating Speech and Language Data With Amazon's Mechanical Turk, 2010.

[62] Nicolas Guéguen. Foot-in-the-door technique and computer-mediated communication. *Computers in Human Behavior*, 2002.

[63] Philipp Gutheim and Björn Hartmann. *Fantasktic: Improving Quality of Results for Novice Crowdsourcing Users*. Master's thesis, University of California, Berkeley, 2012. URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-112.html`.

[64] Daniel Haas, Jason Ansel, Lydia Gu, and Adam Marcus. Argonaut: Macrotask Crowdsourcing for Complex Data Processing. *Proceedings of the VLDB Endowmen*, 8(12):1642–1653, 2015. ISSN 21508097. doi:10.14778/2824032.2824062.

[65] F. Maxwell Harper, Dan Frankowski, Sara Drenner, Yuqing Ren, Sara Kiesler, Loren Terveen, Robert Kraut, and John Riedl. Talk Amongst Yourselves: Inviting Users to Participate in Online Conversations. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, IUI '07, pages 62–71, Honolulu, Hawaii, USA, 2007. ACM. ISBN 1-59593-481-2. doi:10.1145/1216295.1216313.

[66] David J. Hauser and Norbert Schwarz. Attentive Turkers: MTurk participants perform better on online attention checks than do subject pool participants. *Behavior Research Methods*, 48(1):400–407, 2016. ISSN 15543528. doi:10.3758/s13428-015-0578-z.

[67] Chien-Ju Ho and Jennifer Wortman Vaughan. Online Task Assignment in Crowdsourcing Markets. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI '12. AAAI Press, 2012. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5125`.

[68] Chien-Ju Ho, Shahin Jabbari, and Jennifer Wortman Vaughan. Adaptive Task Assignment for Crowdsourced Classification. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *ICML '13*, pages 534–542, 2013. URL `http://jmlr.org/proceedings/papers/v28/ho13.html`.

[69] Eric Horvitz, Paul Koch, and Johnson Apacible. BusyBody: Creating and Fielding Personalized Models of the Cost of Interruption. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, pages 507–510, Chicago, Illinois, USA, 2004. ACM. ISBN 1-58113-810-5. doi:10.1145/1031607.1031690.

[70] Meiqun Hu, Ee-Peng Lim, Aixin Sun, Hady Wirawan Lauw, and Ba-Quy Vuong. Measuring Article Quality in Wikipedia: Models and Evaluation. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 243–252, Lisbon, Portugal, 2007. ACM. ISBN 978-1-59593-803-9. doi:10.1145/1321440.1321476.

[71] Shih-Wen Huang, Jonathan Bragg, Isaac Cowhey, Oren Etzioni, and Daniel S. Weld. Toward Automatic Bootstrapping of Online Communities Using Decision-theoretic Optimization. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, CSCW '16, pages 582–594, San Francisco, California, USA, 2016. ACM. ISBN 978-1-4503-3592-8. doi:10.1145/2818048.2820024.

[72] Ayush Jain, Akash Das Sarma, Aditya Parameswaran, and Jennifer Widom. Understanding Workers, Developing Effective Tasks, and Enhancing Marketplace Dynamics: A Study of a Large Crowdsourcing Marketplace. In *43rd International Conference on Very Large Data Bases (VLDB)*, 2017. ISBN 21508097 (ISSN). doi:10.14778/2735471.2735474.

[73] Kevin Jamieson, Daniel Haas, and Ben Recht. The Power of Adaptivity in Identifying Statistical Alternatives. In *NIPS*, 2016. URL `http://arxiv.org/abs/1603.08037`.

[74] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. ISBN 978-1-4244-3992-8. doi:10.1109/CVPRW.2009.5206848.

[75] Norman Lloyd Johnson and Samuel Kotz. *Urn Models and Their Application: An Approach to Modern Discrete Probability Theory*. Wiley New York, 1977.

[76] Hyun Joon Jung, Yubin Park, and Matthew Lease. Predicting Next Label Quality: A Time-Series Model of Crowdwork. In *Proceedings of the Second AAAI Conference on Human Computation and Crowdsourcing*, HCOMP '14. AAAI, 2014. URL `http://www.aaai.org/ocs/index.php/HCOMP/HCOMP14/paper/view/8976`.

[77] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. ISSN 00043702. doi:10.1016/S0004-3702(98)00023-X.

[78] Sanjay Kairam and Jeffrey Heer. Parting Crowds: Characterizing Divergent Interpretations in Crowdsourced Annotation Tasks. In *CSCW*, pages 1637–1648, 2016. ISBN 978-1-4503-3592-8. doi:10.1145/2818048.2820016.

[79] Ece Kamar and Eric Horvitz. Planning for Crowdsourcing Hierarchical Tasks. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 1191–1199, Istanbul, Turkey, 2015. International Foundation for Autonomous Agents and Multiagent Sys-

tems. ISBN 978-1-4503-3413-6. URL http://dl.acm.org/citation.cfm?id=2772879.2773301.

[80] 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 - Volume 1*, AAMAS '12, pages 467–474, Valencia, Spain, 2012. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-9817381-1-7 978-0-9817381-1-6. URL http://dl.acm.org/citation.cfm?id=2343576.2343643.

[81] David R. Karger, Sewoong Oh, and Devavrat Shah. Iterative Learning for Reliable Crowdsourcing Systems. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a Meeting Held 12-14 December 2011, Granada, Spain.*, NIPS '11, pages 1953–1961, 2011. URL http://papers.nips.cc/paper/4396-iterative-learning-for-reliable-crowdsourcing-systems.

[82] David R. Karger, Sewoong Oh, and Devavrat Shah. Budget-optimal crowdsourcing using low-rank matrix approximations. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 284–291. Ieee, September 2011. ISBN 978-1-4577-1818-2. doi:10.1109/Allerton.2011.6120180.

[83] David R Karger, Sewoong Oh, and Devavrat Shah. Efficient Crowdsourcing for Multi-class Labeling. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, pages 81–92, 2013. doi:10.1145/2465529.2465761.

[84] Diane Kelly and Leif Azzopardi. How many results per page? A study of SERP size, search behavior and user experience. In *Proceedings of the 38th*

International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '15, pages 183–192, 2015. ISBN 978-1-4503-3621-5. doi:10.1145/2766462.2767732.

[85] Jaewon Kim, Paul Thomas, Ramesh Sankaranarayana, Tom Gedeon, and Hwan-Jin Yoon. Pagination versus Scrolling in Mobile Web Search. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16*, pages 751–760, 2016. ISBN 978-1-4503-4073-1. doi:10.1145/2983323.2983720.

[86] Joy Kim, Sarah Sterman, Allegra Argent Beal Cohen, and Michael S. Bernstein. Mechanical Novel: Crowdsourcing ComplexWork through Reflection and Revision. In *CSCW*, pages 233–245, 2017. ISBN 978-1-4503-4335-0. doi:10.1145/2998181.2998196.

[87] Aniket Kittur, Susheel Khamkar, Paul André, and R Kraut. CrowdWeaver: Visually managing complex crowd work. In *CSCW*, 2012. ISBN 978-1-4503-1086-4. URL `http://dl.acm.org/citation.cfm?id=2145204.2145357`.

[88] Ari Kobren, Chun How Tan, Panagiotis Ipeirotis, and Evgeniy Gabrilovich. Getting More for Less: Optimized Crowdsourcing with Dynamic Tasks and Goals. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 592–602, Florence, Italy, 2015. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-3469-3. doi:10.1145/2736277.2741681.

[89] Kenneth R. Koedinger, Emma Brunskill, Ryan Shaun Joazeiro de Baker, Elizabeth A. McLaughlin, and John C. Stamper. New Potentials for Data-Driven Intelligent Tutoring System Development and Optimization. *AI Magazine*, 34 (3):27–41, 2013. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/2484`.

[90] Andrey Kolobov, Mausam, and Daniel S. Weld. Joint Crowdsourcing of Multiple Tasks. In *Human Computation and Crowdsourcing: Works in Progress and Demonstration Abstracts, An Adjunct to the Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing*, volume WS-13-18 of *HCOMP '13*. AAAI, 2013. URL `http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7521`.

[91] Andreas Krause and Daniel Golovin. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, February 2014.

[92] Andreas Krause and Carlos Guestrin. Near-Optimal Nonmyopic Value of Information in Graphical Models. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, UAI '05, pages 324–331. AUAI Press, 2005. URL `https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1238&proceeding_id=21`.

[93] Robert E Kraut, Paul Resnick, Sara Kiesler, Moira Burke, Yan Chen, Niki Kittur, Joseph Konstan, Yuqing Ren, and John Riedl. *Building Successful Online Communities: Evidence-Based Social Design*. MIT Press, 2012.

[94] Todd Kulesza, Saleema Amershi, Rich Caruana, Danyel Fisher, and Denis Charles. Structured Labeling to Facilitate Concept Evolution in Machine Learning. In *CHI*, 2014. ISBN 978-1-4503-2473-1. doi:10.1145/2556288.2557238.

[95] Anand Kulkarni, Matthew Can, and Björn Bjorn Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In *CSCW*, New York, New York, USA, 2012. ACM Press. ISBN 978-1-4503-1086-4. doi:10.1145/2145204.2145354.

[96] Anand Kulkarni, Philipp Gutheim, Prayag Narula, Dave Rolnitzky, Tapan Parikh, and Bjorn Hartmann. MobileWorks: Designing for Quality in a Man-

aged Crowdsourcing Architecture. *IEEE Internet Computing*, pages 1–1, 2012. ISSN 1089-7801. doi:10.1109/MIC.2012.72. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6226337.

[97] Edith Law and Luis von Ahn. *Human Computation*. Morgan & Claypool Publishers, 2011. Series Title: Synthesis Lectures on Artificial Intelligence and Machine Learning Publication Title: Human Computation.

[98] Matthew Lease. On Quality Control and Machine Learning in Crowdsourcing. In *Human Computation, Papers from the 2011 AAAI Workshop, San Francisco, California, USA, August 8, 2011*, volume WS-11-11 of *HCOMP '11*. AAAI, 2011. URL http://www.aaai.org/ocs/index.php/WS/AAAIW11/paper/view/3906.

[99] Jung In Lee and Emma Brunskill. The Impact on Individualizing Student Models on Necessary Practice Opportunities. In *EDM*, 2012.

[100] Christopher H Lin, Mausam, and Daniel S Weld. Dynamically Switching between Synergistic Workflows for Crowdsourcing. In *AAAI*, 2012. URL http://www.aaai.org/ocs/index.php/WS/AAAIW12/paper/viewPaper/5334.

[101] Christopher H. Lin, Mausam, and Daniel S. Weld. Crowdsourcing Control: Moving Beyond Multiple Choice. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, UAI '12, pages 491–500. AUAI Press, 2012. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2309&proceeding_id=28.

[102] Christopher H Lin, Mausam, and Daniel S Weld. Towards a Language for Non-Expert Specification of POMDPs for Crowdsourcing. In *HCOMP Works in Progress*, pages 46–47, 2013. ISBN 978-1-57735-631-8. URL http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7524.

[103] Christopher H. Lin, Mausam, and Daniel S. Weld. To Re(label), or Not To Re(label). In *Proceedings of the Second AAAI Conference on Human Computation and Crowdsourcing*, HCOMP '14. AAAI, 2014. URL http://www.aaai.org/ocs/index.php/HCOMP/HCOMP14/paper/view/8978.

[104] Christopher H. Lin, Mausam, and Daniel S. Weld. Re-Active Learning: Active Learning with Relabeling. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI '16, pages 1845–1852. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12500.

[105] Christopher H. Lin, Mausam, and Daniel S. Weld. A Programming Language With a POMDP Inside. 2016. URL http://arxiv.org/abs/1608.08724.

[106] Christopher H Lin, Mausam, and Daniel S Weld. Active Learning with Unbalanced Classes & Example-Generation Queries. In *HCOMP*, 2018.

[107] Xiao Ling and Daniel S. Weld. Fine-Grained Entity Recognition. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI '12. AAAI Press, 2012. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5152.

[108] 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. doi:10.1111/j.1365-2966.2008.13689.x.

[109] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. TurKit: Tools for Iterative Tasks on Mechanical Turk. In *Proceedings of the ACM*

SIGKDD Workshop on Human Computation, HCOMP '09, pages 29–30, Paris, France, 2009. ACM. ISBN 978-1-60558-672-4. doi:10.1145/1600150.1600159.

[110] Angli Liu, Stephen Soderland, Jonathan Bragg, Christopher H. Lin, Xiao Ling, and Daniel S. Weld. Effective Crowd Annotation for Relation Extraction. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL '16, pages 897–906. The Association for Computational Linguistics, 2016. URL `http://aclweb.org/anthology/N/N16/N16-1104.pdf`.

[111] Yun-En Liu, Travis Mandel, Eric Butler, Erik Andersen, Eleanor O'Rourke, Emma Brunskill, and Zoran Popović. Predicting Player Moves in an Educational Game: A Hybrid Approach. *Proceedings of the 6th International Conference on Educational Data Mining*, pages 106–113, 2013.

[112] Claudia López and Peter Brusilovsky. Adapting Engagement E-mails to Users' Characteristics. CEUR Workshop Proceedings, 2011.

[113] James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, David Roberts, Matthew E. Taylor, and Michael L. Littman. Interactive Learning from Policy-Dependent Human Feedback. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. ISBN 9781510855144. URL `http://arxiv.org/abs/1701.06049`.

[114] V. K. Chaithanya Manam and Alexander J. Quinn. WingIt: Efficient Refinement of Unclear Task Instructions. In *Proceedings of the Sixth AAAI Conference on Human Computation and Crowdsourcing*, HCOMP '18, pages 108–116. AAAI Press, 2018. URL `https://aaai.org/ocs/index.php/HCOMP/HCOMP18/paper/view/17931`.

[115] Travis Mandel, Yun-En Liu, Sergey Levine, Emma Brunskill, and Zoran Popovic. Offline Policy Evaluation Across Representations with Applications to Educational Games. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '14, pages 1077–1084, Paris, France, 2014. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-2738-1. URL http://dl.acm.org/citation.cfm?id=2617388.2617417.

[116] Andrew Mao, Ece Kamar, Yiling Chen, Eric Horvitz, Megan E. Schwamb, Chris J. Lintott, and Arfon M. Smith. Volunteering Versus Work for Pay: Incentives and Tradeoffs in Crowdsourcing. In *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing*, HCOMP '13. AAAI, 2013. URL http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7497.

[117] Andrew Mao, Ece Kamar, and Eric Horvitz. Why Stop Now? Predicting Worker Engagement in Online Crowdsourcing. In *HCOMP*, 2013. URL http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7498/0.

[118] Adam Marcus and Aditya Parameswaran. *Crowdsourced Data Management: Industry and Academic Perspectives*, volume 6. 2015. ISBN 9781680830. doi:10.1561/1900000044. URL http://www.nowpublishers.com/article/Details/DBS-044.

[119] Brian McInnis, Dan Cosley, Chaebong Nam, and Gilly Leshed. Taking a HIT: Designing around Rejection, Mistrust, Risk, and Workers' Experiences in Amazon Mechanical Turk. In *CHI*, 2016. ISBN 978-1-4503-3362-7. doi:10.1145/2858036.2858539.

[120] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, pages 1–9, 2013. ISBN 2150-8097. doi:10.1162/jmlr.2003.3.4-5.951.

[121] George A Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 1995.

[122] Bonan Min and Ralph Grishman. Challenges in the Knowledge Base Population Slot Filling Task. In *LREC*, pages 1137–1142, 2012.

[123] Tanushree Mitra, C.J. Hutto, and Eric Gilbert. Comparing Person- and Process-centric Strategies for Obtaining Quality Data on Amazon Mechanical Turk. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1345–1354, Seoul, Republic of Korea, 2015. ACM. ISBN 978-1-4503-3145-6. doi:10.1145/2702123.2702553.

[124] Meredith Ringel Morris, Jeffrey P. Bigham, Robin Brewer, Jonathan Bragg, Anand Kulkarni, Jessie Li, and Saiph Savage. Subcontracting Microwork. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 1867–1876, Denver, Colorado, USA, 2017. ACM. ISBN 978-1-4503-4655-9. doi:10.1145/3025453.3025687.

[125] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions. *Mathematical Programming*, 14(1):265–294, 1978.

[126] Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z. Gajos. Platemate: Crowdsourcing Nutritional Analysis from Food Photographs. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 1–12, Santa Barbara, California, USA, 2011. ACM. ISBN 978-1-4503-0716-1. doi:10.1145/2047196.2047198.

[127] David Oleson, Alexander Sorokin, Greg P Laughlin, Vaughn Hester, John Le, and Lukas Biewald. Programmatic Gold: Targeted and Scalable Quality Assurance in Crowdsourcing. In *Human Computation Workshop*, page 11, 2011.

[128] Daniel M. Oppenheimer, Tom Meyvis, and Nicolas Davidenko. Instructional manipulation checks: Detecting satisficing to increase statistical power. *Journal of Experimental Social Psychology*, 45(4):867–872, 2009. ISSN 00221031. doi:10.1016/j.jesp.2009.03.009.

[129] Eleanor O'Rourke, Erik Andersen, Sumit Gulwani, and Zoran Popovic. A Framework for Automatically Generating Interactive Instructional Scaffolding. In *CHI*, 2015. ISBN 978-1-4503-3145-6. doi:10.1145/2702123.2702580.

[130] Alexandra Papoutsaki, Hua Guo, Danae Metaxa-Kakavouli, Connor Gramazio, Jeff Rasley, Wenting Xie, Guan Wang, and Jeff Huang. Crowdsourcing from Scratch: A Pragmatic Experiment in Data Collection by Novice Requesters. In *Proceedings of the Third AAAI Conference on Human Computation and Crowdsourcing*, HCOMP '15, pages 140–149. AAAI Press, 2015. URL `http://www.aaai.org/ocs/index.php/HCOMP/HCOMP15/paper/view/11582`.

[131] Aditya G Parameswaran, Anish Das Sarma, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Human-Assisted Graph Search: It's Okay to Ask Questions. *Proceedings of the VLDB Endowment*, 4(5):267–278, 2011. doi:10.14778/1952376.1952377.

[132] Aditya G. Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. CrowdScreen: Algorithms for Filtering Data with Humans. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 361–372, Scottsdale, Arizona, USA, 2012. ACM. ISBN 978-1-4503-1247-9. doi:10.1145/2213836.2213878.

[133] Josep M. Porta, Nikos Vlassis, Matthijs T.J. Spaan, and Pascal Poupart. Point-Based Value Iteration for Continuous POMDPs. *J. Mach. Learn. Res.*, 7:2329–2367, December 2006. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1248547.1248630.

[134] Anna N Rafferty, Emma Brunskill, Thomas L Griffiths, and Patrick Shafto. Faster Teaching by POMDP Planning. In *Proceedings of the 15th International Conference on Artificial Intelligence in Education (AIED 2011)*, pages 280–287, 2011.

[135] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. A Multi-pass Sieve for Coreference Resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 492–501, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1870658.1870706. Series Title: EMNLP '10.

[136] Shreya Rajpal, Karan Goel, and Mausam. POMDP-Based Worker Pool Selection for Crowdsourcing. In *ICML Workshop on Crowdsourcing and Machine Learning*, 2015.

[137] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *EMNLP*, 2016. ISBN 9781941643327. doi:10.18653/v1/D16-1264. URL http://arxiv.org/abs/1606.05250.

[138] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know What You Don't Know: Unanswerable Questions for SQuAD. In *ACL*, 2018. URL http://arxiv.org/abs/1806.03822.

[139] Vikas C Raykar, Shipeng Yu, Linda H Zhao, and Gerardo Valadez. Learning From Crowds. *Journal of Machine Learning Research*, 11:1297–1322, 2010.

[140] Paul J Resnick, Adrienne W Janney, Lorraine R Buis, and Caroline R Richardson. Adding an online community to an internet-mediated walking program. Part 2: Strategies for encouraging community participation. *Journal of medical Internet research*, 2010.

[141] Jean-Charles Rochet and Jean Tirole. Platform competition in two-sided markets. *Journal of the European Economic Association*, pages 990–1029, 2003.

[142] Matt Rosoff. Airbnb Farmed Craigslist To Grow Its Listings, Says Competitor. *Business Insider*, May 2011.

[143] S Russell and P Norvig. *Artificial Intelligence: A Modern Approach, 3rd Ed.* Prentice Hall, 2010.

[144] Jeffrey Rzeszotarski and Aniket Kittur. CrowdScape: Interactively Visualizing User Behavior and Output. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 55–62, Cambridge, Massachusetts, USA, 2012. ACM. ISBN 978-1-4503-1580-7. doi:10.1145/2380116.2380125.

[145] Jeffrey M Rzeszotarski and Aniket Kittur. Instrumenting the crowd: Using implicit behavioral measures to predict task performance. In *UIST*, pages 13–22, 2011. ISBN 978-1-4503-0716-1. URL `http://dl.acm.org/citation.cfm?id=2047199`.

[146] Avi Segal, Robert J Simpson, Victoria Homsy, Mark Hartswood, Kevin R Page, and Marina Jirotka. Improving productivity in citizen science through controlled intervention. In *WWW*, pages 331–337, 2015. ISBN 9781450334730. doi:10.1145/2740908.2743051.

[147] Avi Segal, Kobi Gal, Ece Kamar, Eric Horvitz, and Grant Miller. Optimizing Interventions via Offline Policy Evaluation: Studies in Citizen Science. In *AAAI*, 2018.

[148] Dafna Shahaf and Eric Horvitz. Generalized Task Markets for Human and Machine Computation. In *AAAI*, 2010. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/viewFile/1951/2132`.

[149] 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 Fourteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08)*, pages 614–622, 2008. ISBN 9781605581934.

[150] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343, 1996. ISBN 0-8186-7508-X. doi:10.1109/VL.1996.545307.

[151] Adish Singla, Ilija Bogunovic, Gábor Bartók, Amin Karbasi, and Andreas Krause. Near-Optimally Teaching the Crowd to Classify. In *ICML*, 2014.

[152] Trey Smith and Reid G. Simmons. Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, AAAI '06, pages 1227–1232. AAAI Press, 2006. URL `http://www.aaai.org/Library/AAAI/2006/aaai06-192.php`.

[153] Jacob Solomon and Rick Wash. Bootstrapping Wikis: Developing Critical Mass in a Fledgling Community by Seeding Content. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12,

pages 261–264, Seattle, Washington, USA, 2012. ACM. ISBN 978-1-4503-1086-4. doi:10.1145/2145204.2145247.

[154] Gaurav Sood. Parsed DMOZ data, 2016. URL `http://dx.doi.org/10.7910/DVN/OMV93V`.

[155] Veselin Stoyanov, Nathan Gilbert, Claire Cardie, and Ellen Riloff. Conundrums in noun phrase coreference resolution: Making sense of the state-of-the-art. In *ACL*, 2009. URL `http://dl.acm.org/citation.cfm?id=1690238`.

[156] Ryo Suzuki, Niloufar Salehi, Michelle S Lam, Juan C Marroquin, and Michael S Bernstein. Atelier: Repurposing Expert Crowdsourcing Tasks as Micro-internships. In *CHI*, 2016. ISBN 978-1-4503-3362-7. doi:10.1145/2858036.2858121.

[157] Michael Toomim, Travis Kriplean, Claus Pörtner, and James Landay. Utility of Human-computer Interactions: Toward a Science of Preference Measurement. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2275–2284, Vancouver, BC, Canada, 2011. ACM. ISBN 978-1-4503-0228-9. doi:10.1145/1978942.1979277.

[158] Long Tran-Thanh, Sebastian Stein, Alex Rogers, and Nicholas R Jennings. Efficient Crowdsourcing of Unknown Experts using Multi-Armed Bandits. In *ECAI*, pages 768–773, 2012.

[159] Matteo Venanzi, John Guiver, Gabriella Kazai, Pushmeet Kohli, and Milad Shokouhi. Community-based Bayesian Aggregation Models for Crowdsourcing. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 155–164, Seoul, Korea, 2014. ACM. ISBN 978-1-4503-2744-2. doi:10.1145/2566486.2567989.

[160] Jan Vondrak. Optimal Approximation for the Submodular Welfare Problem in the Value Oracle Model. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 67–74, Victoria, British Columbia, Canada, 2008. ACM. ISBN 978-1-60558-047-0. doi:10.1145/1374376.1374389.

[161] Fabian L. Wauthier and Michael I. Jordan. Bayesian Bias Mitigation for Crowdsourcing. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011*, NIPS '11, pages 1800–1808, 2011. URL `http://papers.nips.cc/paper/4311-bayesian-bias-mitigation-for-crowdsourcing`.

[162] Daniel S. Weld, Mausam, Christopher H. Lin, and Jonathan Bragg. Artificial Intelligence and Collective Intelligence. In *Handbook of Collective Intelligence*, pages 89–114. 2015.

[163] Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The Multidimensional Wisdom of Crowds. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1–9, 2010.

[164] Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier R Movellan. Whose Vote Should Count More: Optimal Integration of Labels from Labelers of Unknown Expertise. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[165] Meng-Han Wu and Alexander J Quinn. Confusing the Crowd : Task Instruction Quality on Amazon Mechanical Turk. In *HCOMP*, 2017. URL `https://aaai.org/ocs/index.php/HCOMP/HCOMP17/paper/view/15943`.

[166] Yan Yan, Romer Rosales, Glenn Fung, and Jennifer G Dy. Active Learning from Crowds. In *ICML*, 2011.

[167] Zhilin Yang, Saizheng Zhang, Jack Urbanek, Will Feng, Alexander H Miller, Arthur Szlam, Douwe Kiela, and Jason Weston. Mastering the Dungeon: Grounded Language Learning by Mechanical Turker Descent. In *ICLR*, pages 1–16, 2018.

[168] Ming Yin and Yiling Chen. Bonus or Not? Learn to Reward in Crowdsourcing. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '15, pages 201–208. AAAI Press, 2015. URL `http://ijcai.org/Abstract/15/035`.

[169] Ce Zhang, Feng Niu, Christopher Ré, and Jude Shavlik. Big Data Versus the Crowd: Looking for Relationships in All the Right Places. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 825–834, Jeju Island, Korea, 2012. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=2390524.2390640`.

Appendix A

CONTRIBUTION REQUEST EMAIL

Subject: Did you use [[Resource Name]]?

Hi,

We'd love to hear your opinion on [[Resource Name]]! Click one of the links below to tell us your opinion about this resource.

I would recommend this resource to other AI researchers

I would NOT recommend this resource to other AI researchers

I haven't used [[Resource Name]]

Your opinion will be publicly posted, along with your name, on Open AIR, an open source collaboration hub for AI researchers run by the Allen Institute for Artificial Intelligence (AI2). Your contribution will help make Open AIR a valuable resource for the entire AI community.

Your response will also help us improve our understanding of how to encourage contribution to an online community. This study is being conducted by researchers at the [[anonymized author information]] who, in collaboration with AI2, are studying ways of bootstrapping online communities, including an analysis of user behavior in response to different email campaigns. Click here for more details about our study. This study is completely anonymous, but if for any reason you do not wish to participate, please click here and no data will be recorded. If you have any questions or suggestions, please email [[anonymized author information]]

Appendix B

SPROUT INTERFACE DEMO

The supplementary file `sprout.mp4` contains demonstrations of the requester and worker task improvement interfaces described in Chapter 7.