

Chapter 7

Automated Task Design

As discussed in earlier chapters, a central challenge in designing human computation systems is understanding how to construct decision environments that effectively attract participants and coordinate the problem-solving process. At a high level, the design of a human computation system consists of two components. One component is the design of incentives—social rewards, game points, and money—that helps to attract a crowd and encourage high quality work. The other component is the organization of individuals—the selection of participants, assignment of tasks, and design of interfaces and workflows—that helps to usefully harness individual efforts to advance a system’s purpose. From the designer’s perspective, the goal is to maximize the rate and quality of output, while minimizing the amount of human effort required and the cost incurred.

In this chapter, we apply ideas from automated environment design to tackle a common computational environment design problem that requesters face on Amazon Mechanical Turk: how should a task be designed so as to induce good output from

workers? This question exemplifies both the incentive and organizational aspects of the design challenge. In posting a task, a requester decides how to break down the task into unit tasks (called HITs, for human intelligence tasks), how much to pay for each HIT, and how many workers to assign to each HIT. These design decisions shape the task environment, which may affect the rate at which workers view and complete unit tasks, as well as the quality of the resulting work.

There are a number of challenges involved in effectively designing a task for posting on Mechanical Turk. As we saw in the nutrition analysis example in Chapter 2, a notable problem is that the effect of design on the rate and quality of work is often imprecisely known *a priori*. Any design's effectiveness is likely dependent on the specifics of the task, and also the quality metric specified. While a designer may have some prior knowledge and be able to experiment with different designs, the design space is exponential in the number of design parameters while the number of experiments that can be performed is relatively small. Furthermore, Mechanical Turk is an inherently noisy and dynamic system, so any measurements obtained are affected in part by system conditions. Moreover, some statistics of interest, such as the number of active workers currently looking for tasks to perform, are unobservable by the requester.

Leveraging the active, indirect elicitation framework of automated environment design, we introduce a general approach for *automated task design*. In this approach, we construct models for predicting the rate and quality of work. These models are trained on worker outputs over a set of designs, and are then used to optimize a task's design. We demonstrate our approach on an image labeling task, for which we aim

to maximize the number of quality labels received, subject to budget constraints. We consider two measures of quality: one based on the number of distinct labels received, and another based on the number of distinct labels received that match an external gold standard.

In our experiments, we find that simple models can accurately predict the output per unit task for both quality metrics, and that the models generate different designs depending on the quality metric we care about. For predicting the rate of work, we observe that a task's completion time is correlated with the amount of work requested per dollar paid, and depends on the time of day when a task is posted. But despite these effects, we find that due to varying system conditions on Mechanical Turk, the task completion time is nevertheless difficult to predict accurately and can vary significantly even for the same design. Focusing on using the quality prediction models for design, we find that for the same budget and rate of pay, optimized designs generated by our models obtain significantly more quality tags on average than baseline designs for both quality metrics.

Section 7.1 reviews related work. Section 7.2 describes the Mechanical Turk marketplace and introduces a general approach for automated task design. Section 7.3 describes the image labeling task. Before exploring different designs for this task, Section 7.4 details an experiment to capture the amount of variability on Mechanical Turk, where we post the same task design multiple times under varying system conditions. Section 7.5 discusses our initial experiments and reports on the performance of models for predicting the rate and quality of work. We consider optimizing the task design based on trained models in Section 7.6, and compare the performance

of optimized designs to baseline designs that pay at the same rate. Section 7.7 discusses the implications of our experiments for automated task design and outlines the possibilities and challenges moving forward.

7.1 Related Work

Studies on the effect of monetary incentives on worker performance found that monetary incentives attracted Mechanical Turk workers (Turkers) to perform more HITs of a task [32, 66, 78, 11] but did not affect the quality of work [66, 78]. In our image labeling task, we also find that tasks are completed more quickly at higher rates of pay. While we find that we can accurately predict the quality of work without factoring in compensation, we do not study the effect of pay on work quality and focus instead on finding effective designs that elicit good output at a fixed rate of pay.

A number of studies have also considered the effect of non-monetary interventions on work quality. Dow et al. [23] showed that asking Turkers to self-assess their work against key performance criteria can improve work quality. Shaw et al. [85] showed that when coupled with monetary incentives, asking Turkers to think about their peers' responses can also improve work quality. Findings on the effect of intrinsic motivation on work quality are mixed; whereas Chandler and Kapelner [12] found that framing a task as being for a good cause did not induce Turkers to produce higher quality solutions, Rogstadius et al. [78] found in their experiments that doing so significantly improved solution quality.

Other studies have considered designing Turk tasks by organizing workers and aggregating output. Snow et al. [87] considered a number of different natural language

annotation tasks, and showed that annotations based on the majority output among a group of Turkers is comparable in quality to expert annotations, but is cheaper and faster to obtain. Su et al. [90] considered the effect of qualification tests on worker output and showed that workers with higher test scores achieve higher accuracy on the actual task. In an orthogonal direction, this chapter focuses on effectively distributing work across identical, parallel subtasks.

Human-powered database systems that recruit a crowd to perform operations such as filters, sorts, and joins are often concerned with efficiency and interested in optimizations that make better use of human effort. Marcus et al. [63, 62] introduced a declarative workflow engine called Qurk and proposed optimizations such as batching tasks and pre-filtering tables before joins. Parameswaran et al. [70] introduced a crowdsourced database system called Deco, and demonstrated that the choice of query execution plan can significantly affect performance. In these systems, having automated procedures that can learn and reason about the crowd’s performance on tasks can potentially provide a means for *query optimization*, that seeks to identify efficient, crowd-tailored query plans.

Several works have applied decision-theoretic planning techniques to control the request for additional work in human computation systems. Kamar et al. [43] demonstrated how predictive models can be used to control the request of additional votes for classifying celestial objects in Galaxy Zoo. Dai et al. [16, 17] introduced TurKontrol, a system for controlling the request of additional voting or improvement tasks based on costs and the inferred work quality. In this chapter, we focus on a complementary challenge of learning about workers to best design individual tasks.

7.2 Automated Task Design on Mechanical Turk

7.2.1 Mechanical Turk

We first review the design environment presented by Amazon Mechanical Turk (www.mturk.com). Mechanical Turk is a crowdsourcing marketplace for work that requires human intelligence. Since its launch in 2005, a wide variety of tasks have been posted and completed on Mechanical Turk. Example tasks include audio transcription, article summarization, and product categorization. Increasingly, Mechanical Turk is also attracting social scientists who are interested in performing laboratory-style experiments [33].

On Mechanical Turk, a *requester* posts jobs for hire that registered workers can complete for pay. A *job* is posted in the form of a group of *HITs* where each HIT represents an individual unit of work that a worker can accept. A requester can seek multiple *assignments* of the same HIT, where each assignment corresponds to a request for a unique worker to perform the HIT. The requester sets the lifetime during which the HITs will be available and the amount of time a worker has to complete a single HIT. The requester can also impose a qualification requirement for a worker to be eligible to perform the task.

When choosing a task to perform, a worker is presented with a sorted list of available jobs, where for each job the title, reward, expiration time, and number of HITs available are displayed. The list can be sorted by the number of HITs available (the default), the reward, creation time, or expiration time. Workers can see a brief task description by clicking the title, or choose to “view a HIT in this group” to see

a preview of a HIT. At this point the worker can choose to accept or skip the HIT. If the HIT is accepted, it is assigned to that worker until it expires or is submitted or abandoned. Workers are not provided with additional information on the difficulty of tasks by the system, although there is evidence of workers sharing information on tasks and requester reputation via browser extensions and on Turk-related forums.¹

Upon receiving completed assignments, the requester determines whether to approve or reject the work. If an assignment is rejected, the requester is not obligated to pay the worker. While tasks vary greatly in pay and the amount of work required, the *reward* per HIT is often between \$0.01 to \$0.10, and most individual HITs require between a few seconds to a few minutes to complete. There are thousands of job requests posted at any given time, which correspond to tens and hundreds of thousands of available HITs. For each HIT completed, Amazon charges the requester 10% of the reward amount or half a cent, whichever is more.

7.2.2 An Automated Approach to Task Design

An exciting aspect of Mechanical Turk as a human computation platform is that it allows a requester to post arbitrary tasks for a large population of workers to complete. A requester has the freedom to design his or her task as desired, with the aim of eliciting good effort from workers toward generating useful work. The task design allows a requester to optimize tradeoffs among the rate of work, the quality of work, and the cost of work. While some of the qualitative aspects of tradeoffs are well understood (e.g., paying more will increase the rate of work, both because more

¹See <http://turkopticon.differenceengines.com/> and <http://www.turkernation.com/>, respectively.

workers will want to accept HITs and that each worker will want to complete more HITs [32]), optimizing the design to achieve particular tradeoffs requires a quantitative understanding of the effect. The effect of non-monetary aspects of task design (e.g., the division of a task into HITs and assignments) on the quality and quantity of work is less well understood, even qualitatively. Such effects are likely to be specific to the task at hand, and depend on a particular requester’s goals and constraints.

We advance an automated approach to task design based on the active, indirect elicitation framework of automated environment design. For a given task, we first experiment with different designs and use the workers’ output and measurements of system conditions to learn a *task-specific model* of the effect of design on the rate and quality of work.² We then use learned models to optimize for good designs based on their predictions. From the automated environment design perspective, we are interested in whether a model learned from observing worker performance can effectively guide the search for better designs.

In the rest of the chapter, we consider as a case study the problem of automatically designing an image labeling task. We describe the task and its design space in the next section, and then apply the following steps to discover an effective design:

1. Estimate variances in target metrics with a baseline design (Section 7.4)
2. Explore the design space with experiments (Section 7.5)
3. Fit models to the experimental data (Sections 7.5.1 and 7.5.2)

²In the general active, indirect elicitation framework, learned information can be incorporated after each experiment and can inform which experiments to conduct thereafter. For simplicity, the elicitation strategy we consider in this setting simply picks a set of experiments to run in *batch*. The inference procedure then updates the model after all experiments are completed.

Provide tags for images

Requester: EnvDes **Reward:** \$0.01 per HIT **HITs Available:** 64 **Duration:** 30 minutes

Qualifications Required: HIT approval rate (%) is greater than 95

Tag 1 image.


Guidelines:

- For each image, you must provide 3 distinct and relevant tags.
- You should strive to provide relevant and non-obvious tags.
- Tags must describe the image, contents of the image, or some relevant context.
- Your submitted tags will be checked for appropriateness.

Payments:

We approve HITs and pay in batches. Your submission will be approved/rejected within 7 days.

Image 1



Tags:

Figure 7.1: A HIT of the image labeling task

4. Optimize the target metrics given the fitted models (Section 7.6.1)
5. Run experiments using the optimized task parameters to validate our approach (Section 7.6.2)

7.3 The Image Labeling Task

We consider an image labeling task in which workers are asked to provide relevant labels (or equivalently, tags) for a set of images. Each HIT contains a number of images, and for each image, requests a particular number of labels for that image.

Workers are informed of the number of images and number of labels required per image within the guidelines provided in the HIT, and are asked to provide “relevant and non-obvious tags.” Workers can provide tags containing multiple words, but this is not required nor specified in the instructions. See Figure 7.1 for a sample HIT that requests three labels for one image. Example labels for this image include “NASCAR,” “race cars,” “red,” “Dale Earnhardt Jr.,” “eight,” and “tires.”

We obtained a large dataset of images from the ESP game,³ which contains 100,000 images and labels collected through gameplay. From this dataset, we use images that contain at least ten labels, of which there are 57,745. Of these, we have used 11,461 images in our experiments. Any particular image we use appears in only one HIT.

We consider two metrics for judging the quality of labels received from workers. One metric counts the number of unique labels received, and is thus concerned with the number of labels collected. The other metric counts the number of labels received that also appear as labels in our gold standard (GS) from the ESP dataset. Since the gold standard labels are those most agreed upon in the ESP game, they are labels that are likely to capture the most noticeable features of an image.

To compute these metrics, we first preprocess labels to split any multi-word labels into multiple single-word labels and convert upper case letters to lower case. We then apply the standard Porter Stemming Algorithm [75] to normalize worker and gold standard labels. This ensures that labels such as “dog” and “dogs” are considered the same label, which is useful for our measure of uniqueness and for comparing received labels to the gold standard. Finally, we remove *stop words* such as “a” and

³<http://www.cs.cmu.edu/~biglou/resources/>

“the,” which account for 0.9% of gold standard labels and 4.6% of distinct labels collected.⁴

In designing the image labeling task, a designer can decide on the reward per HIT, the number of images and tags requested per image per HIT, the total number of HITs, the number of assignments per HIT, the time allotted per HIT, and the qualification requirements. The requester’s goal is to maximize the number of useful labels received as judged by the quality metric of interest, subject to any time and budget constraints. For example, a requester may have \$5 to spend, and aims to collect as many unique tags as possible within the next six hours. One can compare two different designs based on the amount of useful work completed within a certain time frame, or by examining the tradeoff between the work completed per dollar spent and the rate of work.

While each design variable may have an effect on output, we focus our efforts on designing the reward per HIT, the number of images per HIT, the number of labels requested per image, and the total number of HITs. For our experiments, we fix the time allotted per HIT at 30 minutes (the default), but do not expect workers to spend more than a few minutes per HIT. We fix the number of assignments per HIT at 5; this gives us multiple sets of labels per image and will enable a study of the marginal effects of recruiting an additional worker to a HIT on the quality of output in future research. We require all workers to have an approval rate of at least 95%, such that only workers with 95% or more of their previously completed HITs approved are allowed to work on our task.

⁴We used a short, conservative list of stop words from <http://www.textfixer.com/resources/>.

When posting tasks, we collect measurements of worker views and accepts over time, the amount of time a worker spends on a HIT, and the value of output as judged by our quality metrics. We also collect information on system conditions such as the time of day, the number of HITs available on Turk, the page position of our posting in different list orderings, and the number of completed HITs overall in Mechanical Turk. The last statistic is not available directly, and is estimated by tracking the change in the number of HITs available for tasks in the system at two minute intervals.

7.4 Measuring Output Variability

Before considering the effect of design on output, we first report on the amount of variability in the output from Mechanical Turk when using *a fixed task design*. This lets us know how much variance to expect from the system, and allows us to study the effect of system conditions on output.

By observing and following common practice on Mechanical Turk, we selected a design for which each HIT has a reward of \$0.01, contains one image, and requests three labels. We posted a group of 20 HITs at a time, and posted 24 groups of the same task design from 4/12/2010 to 4/20/2010. Each group of HITs was allowed to run for approximately eight hours, and groups of HITs were posted sequentially around the clock. All groups had at least 75% of the assignments completed, with 18 of the 24 groups finishing before the time expired.

Table 7.1 summarizes the mean and standard deviation of the rate and quality of output along a number of measurements.⁵ The task took 5 hours and 30 minutes

⁵We measure the completion time of an unfinished task as the time until the job expires (~8

Statistic	Mean	Standard Deviation
Time to 50% completion (min)	129.54	95.13 / 73%
Time to 100% completion (min)	330.44	124.93 / 38%
Total # of unique tags	264.56	18.06 / 7%
Total # of unique tags in GS	98.56	9.50 / 10%
# of unique workers	13.33	2.99 / 22%
Time to complete a HIT (s)	74.79	25.12 / 34%

Table 7.1: Statistics on a group of image labeling tasks with 20 HITs that was posted 24 times between 4/12/2010 and 4/20/2010. Each HIT pays \$0.01 and requests three labels for one image.

to complete on average, with the quickest run completing in just under 52 minutes and the longest run taking 8 hours and 37 minutes. Unlike task completion time, the number of unique labels received and the number of such labels that are in the gold standard vary much less, suggesting that the quality of output from workers remains relatively constant under different system conditions.

One possible explanation for the significant variation in completion time is that the activity level of workers on Mechanical Turk varies over time. While we do not know how many workers are active on Mechanical Turk at any given time, it is reasonable to think that activity level is correlated with time of day. That is, the system is likely more active during particular “work hours” than at other times. In Figure 7.2 we plot the relationship between the posting time and the time by which 50% or 100% of the tasks were completed. We observe that jobs posted between 6AM GMT and 3PM GMT were completed most quickly; this corresponds to posting between 2AM to 11AM EST in the United States and 11:30AM to 8:30PM IST in India, the two countries that provide 80% of workers on Mechanical Turk [39]. Given that these

hours), and only measure the number of tags and unique workers for completed tasks.

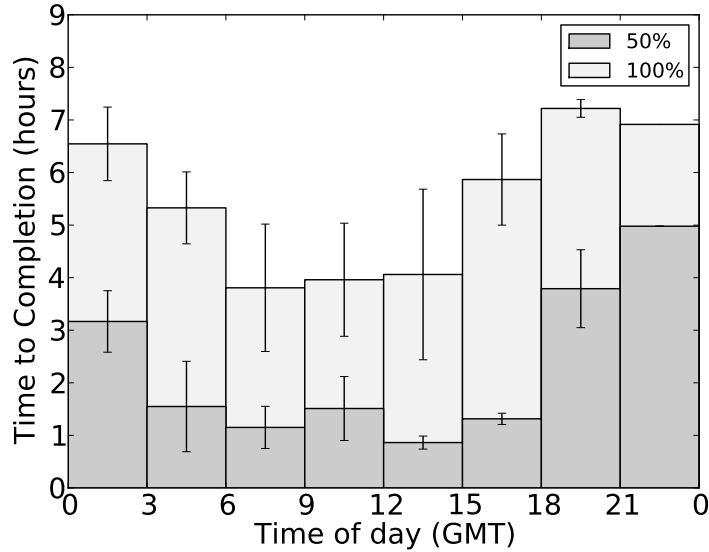


Figure 7.2: The effect of posting time on time until 50% and 100% completion. Bins depict the average completion time of runs posted within a three hour period and error bars represent the standard error. Experiment conducted from 4/12/2010 to 4/20/2010.

times correspond to waking hours in India, we expect most of the workers interested in this task to be from India. We geolocated workers based on their IP addresses by using the Linux shell command `whois`. Of the IP addresses for which we can determine the country of origin (247 out of 307), 62% were from India and 23% were from the US, which is consistent with our intuition.

7.5 Initial Experiments and Behavioral Models

From the variability measurements we learned that the completion time of a task may be highly variable, and may be difficult to predict accurately even for a fixed design. While some of the time variability can be explained by the time of day in which the task is posted, there is still a substantial amount of residual noise. In

contrast, we find that the quality of work does not vary much with system conditions. Based on these observations, we expect that task design may have a large effect on the quality of work, but will only partially influence the rate of work.

In order to understand the effect of design on worker output, we developed models for predicting the quality of labels received per HIT and the completion time. We performed a series of 38 initial experiments—which serves as our training data—in which we varied the task’s design (or configuration) by changing the reward (R), the number of images (N_{pic}) and number of labels per image per HIT (N_{tag}), and the number of HITs (N_{hits}). We considered rewards in the range of \$0.01 and \$0.10 per HIT, and varied the number of images and tags requested between 1 and 10. In choosing configurations, we aimed to cover a large range of values along each dimension, and to vary the total number of tags requested per dollar pay, i.e., $N_{pic}N_{tag}/R$. For the most part we considered jobs that consist of groups of 20 HITs (in 31 configurations), but also included a few jobs containing 30, 150, 500, and 1000 HITs, respectively. Configurations were randomly ordered and allowed to run until completion. They were automatically posted in series over a three week period from 2/2/2010 to 2/24/2010 with no gaps between postings. We fixed the number of assignments (N_{asst}) requested per HIT at five, and required all workers to have an approval rate of at least 95%.

In considering models for predicting the rate and quality of work, we measured the goodness of fit by reporting the coefficient of determination (R^2), the root mean square error (RMSE), the root relative square error (RRSE), and the mean absolute error (MAE), between predicted and actual output. All statistics are computed for the hold-out data via leave-one-out cross-validation.

7.5.1 Predicting Label Quality

We consider models for predicting the average number of quality labels received from workers. A summary of model coefficients and fitness is presented in Table 7.2.

Predicting Unique Tags

For predicting the average number of unique tags that are received per assignment (N_{unique}),⁶ we hypothesized that we would experience diminishing marginal returns as we request more tags per image, suggesting the following model:⁷

$$N_{unique} = \beta N_{pic} \log(N_{tag}) + \epsilon \quad (7.1)$$

We find that the model's predictions are somewhat accurate, with $R^2 = 0.77$. We also considered a model without diminishing marginal returns in the number of labels requested:

$$N_{unique} = \beta N_{pic} N_{tag} + \epsilon \quad (7.2)$$

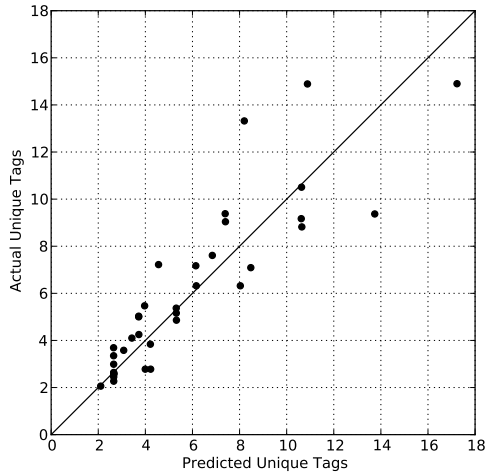
Surprisingly, we observe a significantly better fit, with $R^2 = 0.96$; see Figures 7.3(a) and 7.3(b) for a comparison between the two models' predictions. The model without diminishing returns suggests that the proportion of overlap in tags entered across the five assignments is invariant to the number of tags requested, and that at least within the range of values in our training data we do not observe workers running out of tags to describe an image.

⁶We compute the per assignment contribution by dividing the number of quality tags collected per HIT by the number of assignments, which is fixed at five.

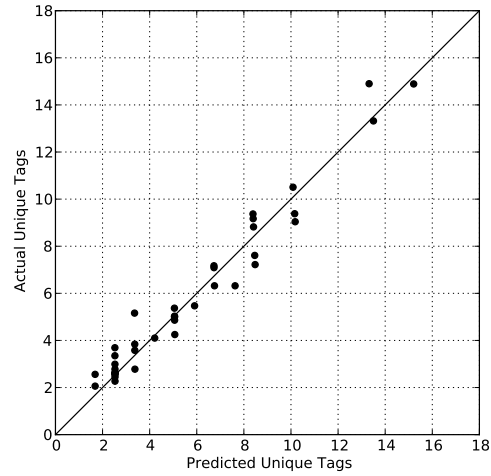
⁷When taking a log, we smooth the input data by adding 1 to the number of tags (N_{tag}) to ensure the feature has weight instead of evaluating to zero.

	Estimated Model		R^2	RMSE	RRSE	MAE
	$N_{pic}N_{tag}$	$N_{pic} \log(N_{tag})$				
Diminishing returns in tags		1.9157 (0.0743)	0.7681	1.6617	0.4816	1.1341
Linear in tags	0.8426 (0.0140)		0.9576	0.7105	0.2059	0.5491
Diminishing returns in tags		0.7241 (0.0115)	0.9576	0.2574	0.2057	0.1743
Linear in tags	0.3050 (0.0115)		0.7652	0.6064	0.4853	0.4406

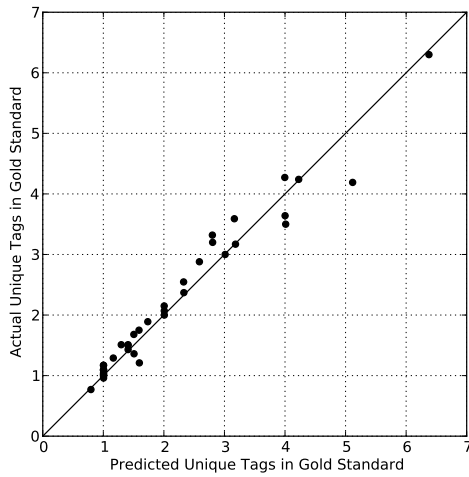
Table 7.2: Summary of model coefficients with standard errors (in parenthesis) and goodness of fit for predicting the average number of quality labels received per assignment. The top two models predict the number of unique tags collected, and the bottom two models predict the number of unique tags collected that are in the gold standard.



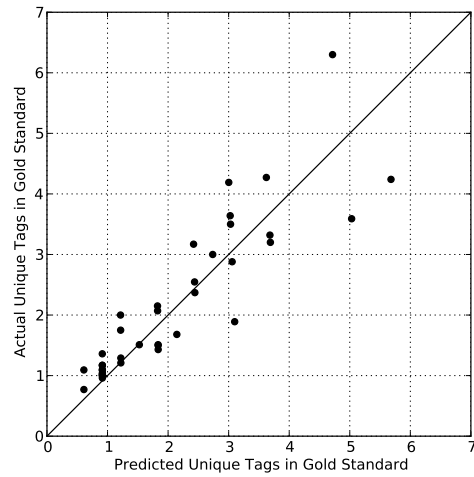
(a) number of unique tags based on diminishing returns in tags.



(b) number of unique tags based on total number of tags requested.



(c) number of unique tags in gold standard based on diminishing returns in tags.



(d) number of unique tags in gold standard based on total number of tags requested.

Figure 7.3: Predicted vs. actual number of quality tags received per assignment

Predicting Unique Tags that are in Gold Standard

For predicting the average number of unique tags received per assignment that are in the gold standard (N_{gs}), we again hypothesized that there would be an effect of diminishing marginal returns as we request more tags per image. Since there is a limited number of tags per image within the gold standard with which the collected tags can match, we would expect the effect of diminishing returns to be much stronger than for our other quality metric. We consider the following model:

$$N_{gs} = \beta N_{pic} \log(N_{tag}) + \epsilon \quad (7.3)$$

The prediction is highly accurate, with $R^2 = 0.96$. The model's fit is significantly better than the fit of a model without diminishing returns ($R^2 = 0.77$); see Figures 7.3(c) and 7.3(d).

7.5.2 Predicting Completion Time

Continuing, we consider models for predicting completion time based on a task's design. Table 7.3 provides a summary of model coefficients and fitness.

Intuitively, a task is more attractive if the pay is high but the amount of work is low. Given similar amounts of work, we would expect the number of tags requested per dollar pay (rate of pay) to be correlated with a task's completion time. We consider all 31 configurations with 20 HITs from our training data, and predict the 50% completion time ($T_{1/2}$) and 100% completion time (T) using the following model:

$$T = \beta_0 + \beta_1 \frac{N_{pic} N_{tag}}{R} + \epsilon \quad (7.4)$$

	Estimated Model			R^2	RMSE	RRSE	MAE
	$\frac{N_{pic} N_{tag}}{R}$	$\cos(t)$	$\sin(t)$				
Pay	25.58 (6.81)			0.45	12418	0.88	7898
Pay + Posting Time	21.47 (5.26)	12173 (2542)	1020 (2265)	0.70	9819	0.70	7549
Pay	49.79 (8.06)			0.68	14914	0.72	11583
Pay + Posting Time	44.71 (6.35)	13775 (3072)	-3104 (2738)	0.79	12630	0.61	9655

Table 7.3: Summary of model coefficients with standard errors (in parenthesis) and goodness of fit for predicting completion time (in seconds). The top two models predict the 50% completion time, and the bottom two models predict the 100% completion time.

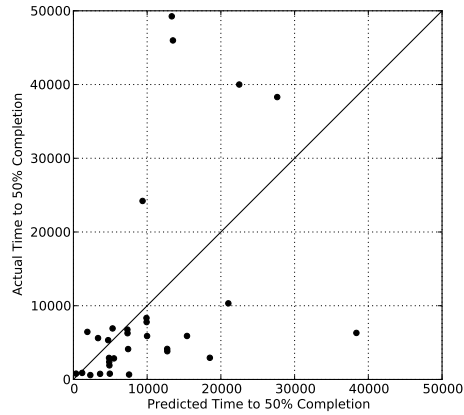
We see that the rate of pay is correlated with the completion time, with $R^2 = 0.68$ for predicting 100% completion. The correlation is weaker for predicting 50% completion time, with $R^2 = 0.45$.

From the results of our variability study, we also expect the time of posting to affect the completion time. As we saw in Figure 7.2, the effect of time of day on completion time is sinusoidal. To incorporate this effect into our model, we convert the time of day to an angle t between 0 and 2π , corresponding to 0:00 GMT and 24:00 GMT respectively, and then encode it as two units, $\cos(t)$ and $\sin(t)$. This encoding scheme ensures that each time of day has a distinct representation and that the values for times around midnight are adjacent. Adding these time variables, we fit the following model:

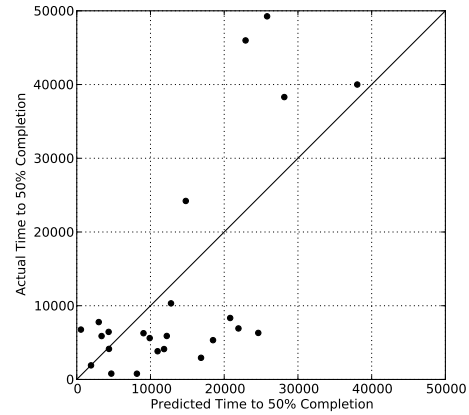
$$T = \beta_0 + \beta_1 \frac{N_{pic} N_{tag}}{R} + \beta_2 \cos(t) + \beta_3 \sin(t) + \epsilon \quad (7.5)$$

We observe an improvement in the fit, with $R^2 = 0.79$ for 100% completion time, and $R^2 = 0.70$ for 50% completion time; see Figure 7.4 for a comparison between the models' predictions. This improvement is more significant for predicting 50% completion time (R^2 from 0.45 to 0.70) than for 100% completion time (R^2 from 0.68 to 0.79). One possible explanation is that the effect of the posting time diminishes when HITs are posted for a longer time frame that includes other times of the day.

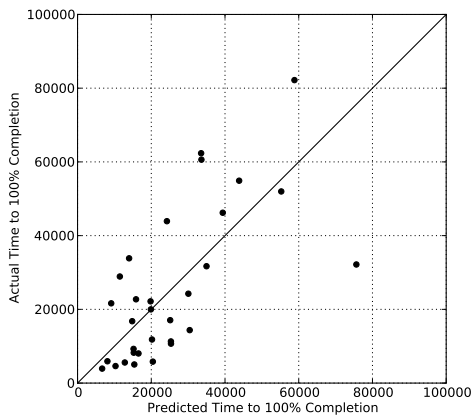
The fit of these models suggests that the rate of pay and the time of posting are correlated with the completion time, but that there is still a substantial amount of unexplained variance. To use these models for prediction and design, it would be useful to consider not only the expected completion time, but also to be mindful of the variance in the prediction. Furthermore, the current models are only trained on



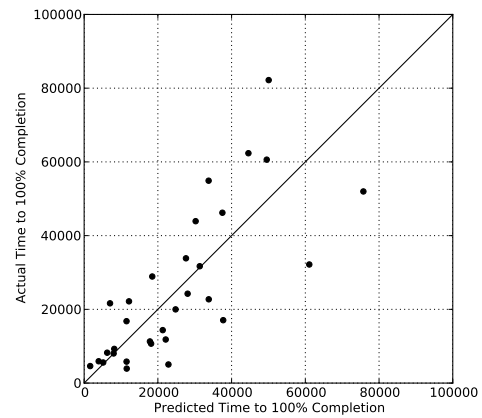
(a) time to 50% completion based on rate of pay.



(b) time to 50% completion based on rate of pay and posting time.



(c) time to 100% completion based on rate of pay.



(d) time to 100% completion based on rate of pay and posting time.

Figure 7.4: Predicted vs. actual time until 50% and 100% completion (in seconds).

configurations with 20 HITs, and do not incorporate the effect of varying the number of HITs. We leave the exploration of these directions for future work, and for now focus on using the quality prediction models for design.

7.6 Design Experiment

The initial experiments provide us with an understanding of how workers respond to different designs and thus serve as the building blocks for effective task design. Even at the same level of desirability to workers—e.g., as measured by the pay per tag, or more generally, the estimated pay per hour—we expect some designs to induce more quality output than other designs. We now investigate whether the learned models can help us make informed design decisions for particular quality metrics of interest.

7.6.1 Design Optimization and Experiment Setup

We consider a simple design experiment in which we compare different designs at a fixed pay per tag. We focus our comparison on the number of quality labels received (per dollar spent), and do not concern ourselves with the rate at which work completes.⁸ Fixing the rate of pay allows us to compare designs based on the kind of work they request, and removes the effect of assigning more work at a lower rate of pay to get more quality labels from confounding the comparison.

We consider experiments at two pay rates: a low rate that pays 1¢ for every three tags, and a high rate that pays 1¢ per tag. For each pay rate, we compare the

⁸In practice, we can set the rate of pay based on how quickly we want work to get done. But since time is not considered in this experiment, fixing the rate of pay allows for a fair comparison between designs.

output of *baseline designs* to designs optimized for each of our two quality metrics. Baseline designs are chosen by observing common practice in image labeling tasks on Mechanical Turk, which typically requests three or four tags for a single image within each HIT. Each design is given a budget of \$5, which must account for fees paid to Amazon as well as payments to workers. As in our initial experiments, the number of assignments per HIT (N_{asst}) is fixed at 5.

To optimize the task design, we choose values for the reward per HIT (R), number of images per HIT (N_{pic}), number of tags requested per image (N_{tag}), and the total number of HITs (N_{hits}), in order to maximize the total number of quality tags received as predicted by the model with the best fit, subject to budget and rate of pay constraints. We consider rewards in the range of \$0.01 to \$0.10 per HIT, and the number of images and tags requested per image in the range of 1 to 10. For example, the following formulation captures the optimization problem for finding a design that maximizes the total number of unique tags received as predicted by our model, subject to a \$5 budget and a pay rate of \$0.01 per tag:

$$\max_{R, N_{pic}, N_{tag}, N_{hits}} 0.8426 N_{pic} N_{tag} N_{hits} N_{asst} \quad (7.6)$$

$$N_{HIT} N_{asst} (R + \max(0.1R, 0.005)) \leq 5 \quad (7.7)$$

$$R / N_{pic} N_{tag} = 0.01 \quad (7.8)$$

Constraint 7.7 ensures that the cost of the design stays within budget, and constraint 7.8 ensures that the pay per tag is \$0.01. The max term in the budget constraint corresponds to Mechanical Turk's per assignment fees, which is 10% of the reward or half a cent, whichever is more.

Table 7.4 summarizes the baseline and optimized designs for both pay rates and

	pay/tag	R	N_{pic}	N_{tag}	N_{hits}	N_{asst}	Posting Fees	Total Cost
low pay baseline	$\frac{1}{3}\text{¢}$	\$0.01	1	3	66	5	\$1.65	\$4.95
optimized for N_{unique} (low pay)	$\frac{1}{3}\text{¢}$	\$0.06	9	2	15	5	\$0.45	\$4.95
optimized for N_{GS} (low pay)	$\frac{1}{3}\text{¢}$	\$0.03	9	1	28	5	\$0.70	\$4.90
high pay baseline	1¢	\$0.04	1	4	22	5	\$0.55	\$4.95
optimized for N_{unique}, N_{GS} (high pay)	1¢	\$0.10	10	1	9	5	\$0.45	\$4.95

Table 7.4: Baseline and optimized designs for an image labeling tasks with a \$5 budget.

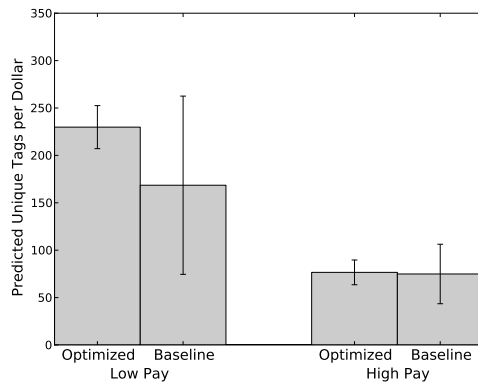
quality metrics. For the low pay rate, we consider a baseline design that requests 3 tags for one image, which is the same design that we had adopted for measuring variability (but with more HITs). For maximizing the number of unique tags collected, we see that the optimized design attempts to save on posting fees by putting more work into a HIT and paying more per HIT, which allows for more tags to be requested. For maximizing the number of unique tags that are in the gold standard, the optimized design avoids diminishing returns by requesting 1 tag per image, and also saves on posting fees by putting more work in a single HIT.

For the high pay rate, we consider a baseline design that requests 4 tags for one image. Here the optimized designs for the two quality metrics are the same. More work is put into each HIT to save on posting fees (hitting the upper bound on reward per HIT) and only 1 tag is requested per image to avoid diminishing returns.

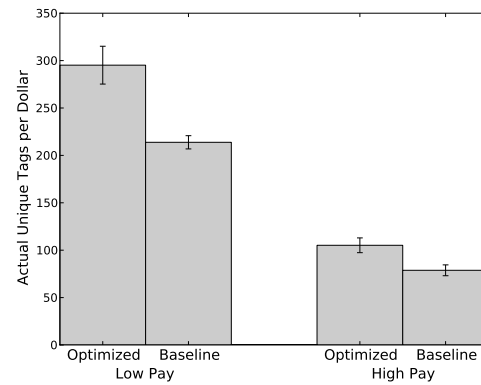
Figures 7.5(a) and 7.5(c) show the models' predictions with bars representing the 95% prediction interval for these designs. We see that the difference in the predicted numbers of unique tags per dollar spent between baseline and optimized designs is small, since the benefits of the optimized design comes only from savings in posting fees. By avoiding diminishing returns in tags, designs optimized for the numbers of unique tags that are in the gold standard are expected to perform significantly better.

We post five groups of each baseline and optimized design in round-robin order. Each group ran initially for 6 hours and was allowed to finish at a later time if needed.⁹

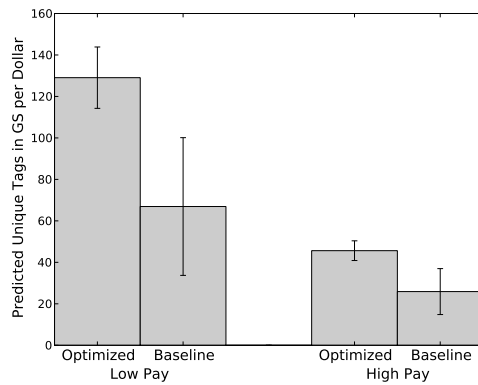
⁹We initially posted the baseline designs between 3/25/2010 and 3/29/2010, and the optimized designs between 4/22/2010 and 4/26/2010. While almost all trials of the high pay configurations completed within this time frame, many of the low pay configurations did not; these configurations were ran to completion between 4/29/2010 and 5/7/2010.



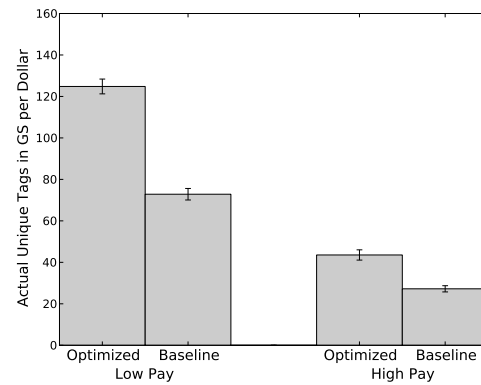
(a) Predicted number of unique tags per dollar spent.



(b) Actual number of unique tags per dollar spent.



(c) Predicted number of unique tags in gold standard per dollar spent.



(d) Actual number of unique tags in gold standard per dollar spent.

Figure 7.5: Predicted and actual number of quality tags received per dollar spent for baseline and optimized designs. Error bars in predictions indicate the 95% prediction intervals, and error bars in results represent the standard error over five runs of each design.

7.6.2 Results

Figures 7.5(b) and 7.5(d) show the average number of unique tags and the average number of unique tags in the gold standard received per dollar spent, with bars capturing the standard error of the mean.

In all comparisons, we find that the optimized designs received more quality tags than baseline designs. The optimized designs for unique tags received 38% more tags in the low pay condition, and 33% more in the high pay condition. For collecting unique tags that are in the gold standard, the optimized designs received significantly more quality tags than the baseline comparisons, with 71% more in the low pay condition and 60% more in the high pay condition. For all baseline and optimized designs, the actual number of gold standard tags received is very close to our model's predictions (within 11%), and well within the prediction intervals.

Interestingly, our optimized designs received significantly more unique tags than our models predicted: 28% more in the low pay condition and 38% more in the high pay condition. One possible explanation is that our model underpredicts the number of unique tags when the number of tags requested per image is low, as is the case in our designs. After checking the model's predictions on the training data, we noticed that our model underpredicts for 10 out of the 11 configurations that request one or two tags per image (by 15% on average). Our model also underpredicted the number of unique tags obtained by the baseline in the low pay condition by 27%, suggesting that the model may need to be refined to improve prediction accuracy. Nevertheless, the information contained in the model was still helpful in discovering optimized designs that significantly outperform the baseline designs.

7.7 Discussion

By collecting data about how workers respond to designs in our initial experiments, we are able to construct models that can accurately predict worker output in response

to different designs. These models can then be used to optimize a task's design, subject to designer constraints such as budget and rate of pay, to induce quality output from workers. The results from our experiments show that designs that are optimized based on learned models obtain significantly more high quality labels than baseline comparisons.

There are a number of possible extensions to this work. We would like to understand the effect of distributing work across multiple assignments on the quality of output, and to include the number of assignments as a design variable. We are also interested in revisiting models for predicting the rate of work, and incorporating them to design with respect to time-related tradeoffs. One possible direction is to learn the relative rates at which work completes for different designs, which may be sufficient for accurately predicting the relative output between designs. Furthermore, while we focus here on the design of a task with identical, parallel subtasks, we are interested in developing a general approach for automating the design of human computation algorithms and workflows. We discuss this in the next chapter.

We believe the active, indirect elicitation approach of learning from observations of behavior to optimize designs can be effectively used to design a variety of tasks, with respect to different performance metrics, and in richer design spaces. While linear regressions were used for this work, other modeling approaches and methods from machine learning and statistics can be incorporated into the design process. The models of behavior need to be specific to the particular task and performance metric at hand. Constructing accurate models will likely require drawing from an understanding of the task domain and the population of workers, and learning from

experimentation.

In addition to accurate models, we need methods that help to discover effective designs quickly after only a few experiments. While we trained our models on a set of manually picked designs and then used these models to optimize the design, we can develop elicitation strategies that automatically pick subsequent experiments in a way that drives the search for better designs. In the next chapter, we develop a general method for automatically synthesizing workflows in which the system optimizes the choice of experiments to maximize the value of information obtained.