CrowdLearner: Rapidly Creating Mobile Recognizers Using Crowdsourcing

Shahriyar Amini*
Carnegie Mellon University
Pittsburgh, PA
shahriyar@cmu.edu

Yang Li
Google Research
Mountain View, CA
yangli@acm.org

ABSTRACT
Mobile applications can offer improved user experience through the use of novel modalities and user context. However, these new input dimensions often require recognition-based techniques, with which mobile app developers or designers may not be familiar. Furthermore, the recruiting, data collection and labeling, necessary for using these techniques, are usually time-consuming and expensive. We present CrowdLearner, a framework based on crowdsourcing to automatically generate recognizers using mobile sensor input such as accelerometer or touchscreen readings. CrowdLearner allows a developer to easily create a recognition task, distribute it to the crowd, and monitor its progress as more data becomes available. We deployed CrowdLearner to a crowd of 72 mobile users over a period of 2.5 weeks. We evaluated the system by experimenting with 6 recognition tasks concerning motion gestures, touchscreen gestures, and activity recognition. The experimental results indicated that CrowdLearner enables a developer to quickly acquire a usable recognizer for their specific application by spending a moderate amount of money, often less than $10, in a short period of time, often in the order of 2 hours. Our exploration also revealed challenges and provided insights into the design of future crowdsourcing systems for machine learning tasks.

Author Keywords
Crowdsourcing; crowdsensing; mobile interaction; gesture recognition; activity recognition; machine learning.

ACM Classification Keywords
D.2.2 **[Software Engineering]:** Design Tools and Techniques; H.5.2 **[Information Interfaces And Presentation]:** User Interfaces; I.5.2 **[Pattern Recognition]:** Design Methodology.

INTRODUCTION
Mobile users experience diverse interaction scenarios throughout everyday activities. These diverse scenarios call for novel input modalities and the use of context to provide users with intuitive and fast access to their mobile resources. Such interaction techniques often involve recognizing user context and intentions from mobile sensory input, such as accelerometer readings. However, creating recognizers is a time-consuming and expensive feat, often requiring expert knowledge in machine learning. This is beyond the reach of many developers. Thus, it is important to investigate how to empower developers to rapidly create recognizers for improving mobile user experience.

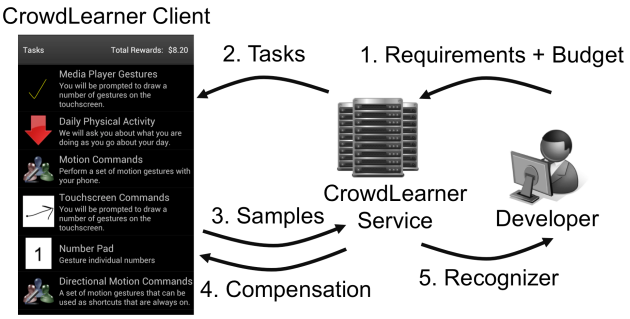

Figure 1. A developer submits a task and provides funds to run it on CrowdLearner. CrowdLearner publishes the task and compensates mobile workers who answer questions and label samples. CrowdLearner continuously builds and updates a recognizer and presents its performance to the developer.

Several off-the-shelf machine learning tools such as Weka [28] and GART [27], and lightweight recognition algorithms such as the $1 recognizer [29] have gained traction with developers and researchers. However, these tools often act more as libraries or even source code that do not encapsulate the complexity of creating recognizers from developers. High-level tools, such as Gestalt by Patel *et al.*, have been developed for using and understanding machine learning techniques [20]. Though promising, prior work has often left out data collection, a critical but time-consuming and labor-intensive step for producing efficient recognizers.

In this paper, we present CrowdLearner, a framework that employs crowdsourcing to enable an end-to-end solution for generating recognizers for mobile interaction. These recognizers, referred to as *mobile recognizers*, take in built-in mobile sensor input such as touchscreen events or accelerometer readings and infer target user actions, such as gestures or physical activities. CrowdLearner allows non-expert users (such as mobile app developers, researchers, and interaction designers), referred to here as *developers*, to easily create mobile recognizers by creating a recognition

UIST'13, October 8–11, 2013, St. Andrews, United Kingdom.
ACM 978-1-4503-2268-3/13/10.
http://dx.doi.org/10.1145/2501988.2502029

* This work was done while the author was an intern at Google Research.

task and publishing it to mobile users, referred to here as *workers* (see Figure 1). Through CrowdLearner's web interface, a developer can create a recognition task by specifying a set of targets for recognition (i.e., classes in machine learning), the sensor inputs to use, data collection strategies and the amount of funds allocated for the task. With a high-level task description, CrowdLearner automatically determines the technical details for constructing the recognizer and scheduling data collection. CrowdLearner then collects and labels samples by relying on the wisdom and scale of the crowd, generates a recognizer, evaluates it, and presents its accuracy to the developer as the recognizer evolves with additional data collection. Once satisfied, the developer can export the recognizer for use in the intended application.

Our focus in this paper is to design and experiment with new frameworks and methods for creating mobile recognizers using crowdsourcing, instead of focusing on specific details such as featurization and classification techniques. This paper makes the following contributions:

First, we present an end-to-end framework for developers to rapidly create mobile recognizers that are specific to their tasks by ordering them. In doing so, we eliminate the overhead of recruiting and data collection, and reduce the need for expert knowledge of machine learning.

Second, we demonstrate two strategies as well as the supporting UI for collecting training data in the wild: *participatory* and *opportunistic* sampling, which complement traditional, laboratory-based data collection by allowing a more diverse and realistic set of observations. We also provide a *collaborative temporal* sampling model to further improve opportunistic sampling.

Finally, through deploying and experimenting with CrowdLearner, we reveal how developers and crowd workers reacted to this kind of system, and provide insights into designing such systems in the future.

RELATED WORK

Crowdsourcing has been used in the past to accomplish complex tasks by reducing them to smaller ones and validating the results from the crowd. Bernstein *et al.* use a Find-Fix-Verify approach to bring the wisdom of crowds to word processing [2]. Kittur and Kraut explore crowdsourcing with respect to creation of Wikipedia articles [15]. Heer and Bostock use crowdsourcing to assess visualization design [12]. Song *et al.* use real-time crowdsourcing to label unrecognized activities in activity recognition systems [25]. JANA is a commercial mobile crowdsourcing platform that rewards workers with airtime for performing tasks such as consumer research and product promotion. CrowdLearner uses crowdsourcing in a new light by automatically creating mobile recognizers for non-experts.

Prior work has explored how participants game crowdsourcing platforms and how to mitigate this problem [8, 14]. CrowdLearner workers perform simple tasks such as drawing a stroke gesture or labeling data samples (similar to Captchas [26]) to receive monetary compensation. Although we touch on mechanisms for validating sensor data, it is not our focus. As the first step to addressing the issue, we concentrate on enabling the end-to-end workflow for non-experts to create mobile recognizers using a crowdsourcing approach. However, the body of prior art focusing on validating crowd input would also prove useful in the case of CrowdLearner.

CrowdLearner is also closely related to crowdsensing [5, 7, 24] and *in situ* experience sampling [9, 10]. CrowdLearner shares the goal of previous work for collecting samples in users' natural environments. However, CrowdLearner goes beyond prior work by making sense of collected mobile sensor data and user labeling, and employing machine learning techniques to automatically generate recognizers for high-level events and user actions. In particular, CrowdLearner's data collection mechanisms and interfaces were designed for capturing mobile on-device sensor data and facilitating mobile user labeling, which are not designed for general mobile data collection purposes. As such, the outcome of CrowdLearner can be directly incorporated into developers' applications. Existing mobile crowdsensing platforms (e.g., CrowdLab [6], PRISM [7] and Medusa [21]) enable researchers and developers to run custom experiments on crowd devices. However, the level of knowledge required to setup and use these platforms place them beyond the reach of non-experts. CrowdLearner obviates the need for developers to have expert knowledge of sensing, data analysis, feature extraction, and classification.

Past work has looked at high-level tools for producing classifiers while reducing the need for expert knowledge of machine learning, e.g., MAGIC [1] and Exemplar [11]. CrowdLearner expands prior work by taking the event generation and data collection to mobile users' natural environments. It minimizes the burden on developers by eliminating recruiting and data collection. Furthermore, it creates recognizers for diverse devices that might be in use in the wild, rather than only a limited set of devices and sensors that are accessible to the developer.

INTERACTING WITH CROWDLEARNER

In this section, we discuss how a developer can create a recognition task and monitor its progress in CrowdLearner and how a mobile crowd worker interacts with the system to contribute labeled training data.

Developers: Creating a Recognition Task

To create a recognition task, a developer uses a web interface that streamlines task creation with a wizard. The wizard only requests a small amount of information, such as the targets to be recognized, sensors to use on a mobile device, data collection strategies and the amount of funds that the developer is willing to allocate for the task.

Adding a Task Description

The wizard includes four tabs for task *Description*, *Sensors*, *Targets*, and *Strategy*. On the Description tab, the developer

provides basic information about a task, such as the task title, a short summary about the task, and a set of instructions for crowd workers to perform the task. Once the task is published, all this information is displayed on a worker's mobile device. For example, for the Motion Commands Task, the basic task information is as follows:

Title: Motion Commands; Summary: Perform a set of motion gestures with your phone; Instructions: Imagine you can operate your phone by moving it in certain ways, e.g., shaking the phone to ignore a phone call. We will ask you to perform a set of motions with your phone. Please hold your phone in one hand and make sure that your phone's screen is facing you when conducting these gestures. The top of your device should point away from you.

Selecting Mobile Sensors

On the Sensors tab (Figure 2), the developer selects sensor input for the recognizer to use. The developer can directly pick specific sensors from the list, such as Accelerometer, Microphone or Touchscreen. Based on the selected sensors, CrowdLearner automatically generates an appropriate feature vector for recognition (classification).

Alternatively, the developer can select a group of necessary sensors for recognizing high-level event types, such as Motion, Audio or Touch. These high-level event types are useful for developers who are less familiar with built-in sensors on mobile devices. For example, when a developer selects Motion, both Accelerometer and Gyroscope are selected, as they are typical sensors for inferring motion. Another advantage of this feature is that a developer can also explore how a sensor is typically used. When a sensor is selected, the corresponding high-level event type will be selected as well. For instance, selecting Microphone will also select the high-level event type, Audio.

Composing Recognition Targets

After choosing the sensors, the developer specifies the targets for the recognizer in the Targets tab (see Figure 3). The developer can give each target a name and describe it further by uploading an image or a video clip. Targets are presented to workers to perform actions or answer questions as instructed by the task. With images and video, developers can explain target actions that would otherwise be difficult to describe in text. For example, when instructing a worker

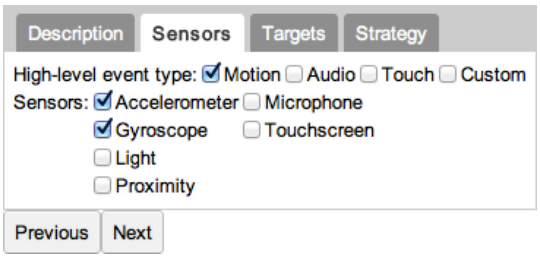

Figure 2. The developer selects which sensors CrowdLearner should sample. CrowdLearner decides on the set of features for training a recognizer based on the sensors.

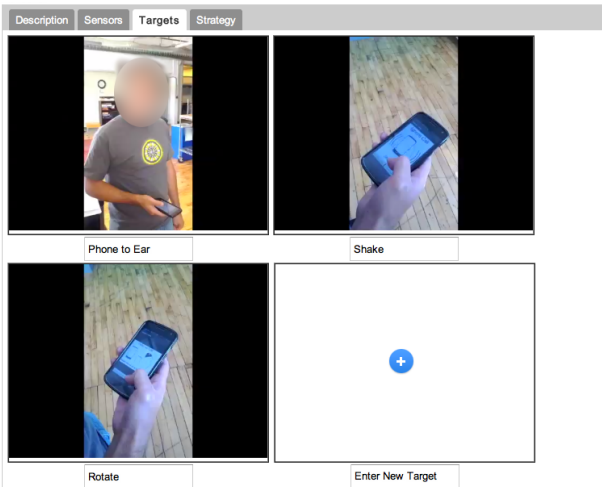

Figure 3. The developer specifies a set of targets along with images or videos for the task. The developer can add or edit targets in the wizard and view/play the media.

to perform a *Rotate* action in the Motion Commands task, the accompanying video for *Rotate* can easily explain how the worker should move his phone to perform the action.

Specifying Task Strategies

On the Strategy tab, developers provide the learning strategy and monetary constraints for performing the task. CrowdLearner supports two learning strategies: participatory and opportunistic sampling. The developer chooses between "*learn when I ask the user to do something*"—in a participatory fashion—or "*learn when something interesting happens*"—an opportunistic fashion. The developer decides on how she wants to collect data. For instance, for a task concerning touchscreen handwritten symbols, she would choose participatory sampling because this kind of behavior—writing specific symbols on the touchscreen—only happens when the worker is asked to do so. However, for a task concerning daily physical activity, she would use opportunistic sampling, as events can occur even when the phone is not in active use. The developer also selects the maximum number of questions a worker could answer, the compensation per question, and the total funds allocated for the task. These constraints ensure that the generated recognizer does not rely on samples from only a small group of workers, and that CrowdLearner does not go beyond the developer's budget. Once the developer is satisfied with the task, she can publish it to CrowdLearner, which automatically distributes the task to the workers.

Crowd Workers: Performing a CrowdLearner Task

CrowdLearner distributes a task to a worker's mobile device if the sensors required by the task are available on the device. On the mobile client's main screen, workers can see all available tasks and also check the rewards accumulated so far (Figure 1). After selecting a task, the worker will be greeted with the task instructions. On the next screen, he will see the amount of compensation received for each question answered (i.e., each sample contributed). If moti-

vated, he can proceed to participate in the task. Depending on the type of task chosen, participatory or opportunistic, the worker performs the task through a different procedure.

Participatory Tasks

During a participatory task, CrowdLearner prompts the worker to perform a particular action (considered a question in CrowdLearner) at a time. The task repeats a Question and Answer pattern. On the Question screen (Figure 4), CrowdLearner instructs the worker the action to perform by showing the action name along with a pictorial or video description. On the following Answer screen (Figure 4), the worker has the opportunity to perform the action, redo it if needed, or skip it. The order of questions is randomized.

Once the worker understands the target action, he can go to next screen. Depending on the type of sensors that the task involves, CrowdLearner presents different screens to the worker. For a touchscreen gesture task, because a worker is expected to interact on the touchscreen, such as drawing a gesture symbol, CrowdLearner leaves the screen blank with only the name of the symbol and its associated media displayed at the right corner. The worker needs to recall the target gesture and draw it on the screen. Once finished, the worker can click on a button to go to the next question.

Unlike a touchscreen that stops sensing when a worker lifts his finger from the screen, sensors such as accelerometers continuously sense. As a result, we need a mechanism for the system and the worker to agree on the beginning and the end of sensing. For example, if a worker has to click a button to signal the system to stop sensing, the motion for stopping the sensing will be captured as part of the target action. In addition, the action that a worker is to perform puts the phone in a state where he cannot interact with the display. For instance, for the "Put the phone to ear" action in the Motion Commands task, we do not want workers to place the phone to their ear and then bring it down to confirm the answer. To address this issue, CrowdLearner employs a count down timer. Once a worker moves to the Answer Question screen, a timer starts and the worker is to perform the action before the timer ends (see Figure 4). The worker is instructed to not move the phone or perform any other actions until the timer runs out and he hears an audio notification. We currently use 3 seconds for the timer, which we empirically found is an upper bound for the execution time for many motion-related actions.

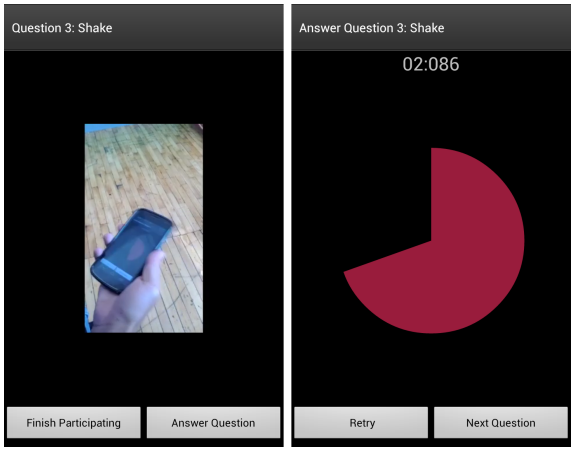

Figure 4. On the Question screen, workers are prompted to perform an action through a target and a pictorial or video description. On the Answer screen, workers perform the action instructed on the previous Question screen. A count down timer informs when to perform the action.

Opportunistic Tasks

An opportunistic task is designed to capture events (training samples) that do not occur at a predefined time, such as walking or biking. These events are difficult to capture using a participatory approach in which a worker is asked to perform a target action. When a worker signs up for an opportunistic task, such as one collecting daily physical activity samples, the worker does not need to perform any action at the signup. Instead, CrowdLearner will notify the worker to answer a question regarding an observed event. The answer to this question effectively labels a collected sample.

However, there is a dilemma in executing opportunistic tasks. When the system observes an event of interest during the day, asking a question about the event may disturb the worker's ongoing activity, e.g., the worker might be biking and having difficulty to take out the phone and answer a question. Therefore, it is impractical to ask a worker to answer the question immediately when an event occurs. To address this problem, we employ a notification and expiration mechanism. When an event of interest occurs, the system sends the worker's phone a notification along with a distinct vibration pattern. The worker does not have to answer the notification immediately.

When a worker joins an opportunistic task, the system plays a sample of the vibration pattern and instructs what the vibration means: "*CrowdLearner will send you a notification question along this vibration pattern whenever it observes something relevant to the task. You don't have to respond to the question right away, but remember your state at the time of vibration for answering the question later.*" We designed the vibration pattern to be similar to SOS in Morse code, which is distinctive from many vibration patterns used by other applications. However, a worker may not be able to answer the question for a while, which may lead to him forgetting the event. Therefore, CrowdLearner keeps the notification question available for a limited amount of time only. We chose 30 minutes to ensure that the workers are able to recall their state at the time of the vibration.

When selecting their state at the time of the event, workers have other choices available in addition to the targets specified by the developer. These choices consist of, "*Didn't have the phone on me*", "*Don't remember*" and "*None of the above*". Based on the answer selected by the worker, the event is used as a positive sample, a negative sample, or ignored. Positive samples confirm a certain type of event; whereas negative samples are discounted for a particular

event. In an actual deployment, the system could avoid workers who provide too many negative samples to prevent workers from gaming the system.

Developers: Monitoring the Progress via a Dashboard

The developer can monitor the progress of the task through a web-based dashboard. The dashboard shows a list of all the tasks created by the developer. By selecting a task, the developer can view the accuracy of the recognizer and its progression through a chart. A table presents the accuracy and number of samples for each target. The developer can also view the total amount of funds allocated and remaining, the number of participating workers and sample, and the average number of samples per worker. The dashboard also allows the developer to allocate more funds to the task.

IMPLEMENTATION

In this section, we describe the technical details of our current implementation. These specific technical aspects are not our focus, but are essential for creating a fully-functional system that can be deployed in the wild to examine the workflow of CrowdLearner. We built CrowdLearner as a cloud service. It uses Google App Engine (GAE) to serve tasks and store training data, recognizers, and performance measures. We used Google Web Toolkit (GWT) for implementing the web-based task creation wizard and presenting recognizer evaluation. The CrowedLearner client, used by workers to perform tasks, is implemented based on the Android platform.

Sampling

CrowdLearner aims to abstract away sampling and data collection. Currently, it allows a task to leverage several popular mobile sensors as shown in Figure 2. CrowdLearner collects the profile for each type of device presented in the workers' pool of devices. These device profiles enable CrowdLearner to account for variations in device sensors and packaging by producing a recognizer for each type of device. A default sampling rate is set for each type of sensor at which samples from all devices are resampled, e.g., 40Hz for accelerometers.

Sampling Conditions for Opportunistic Tasks

For opportunistic tasks, we have a set of requirements before any data collection takes place to improve user experience and also to respect workers' computational resources. Opportunistic sampling takes place at random intervals throughout the day. In our current implementation, data collections are set to occur 30 to 90 minutes apart, which enable CrowdLearner to work around workers' daily routines. The sampling duration is set to be 30 seconds. Nevertheless, the sampling interval and duration should be developer-configurable in actual deployments.

Opportunistic sampling is activated upon satisfying several requirements. The mobile client checks local time to ensure that data collection takes place between 9am and 8pm. We made a usability trade-off here, in that we do not want to disturb workers at night. However, this prevents capturing events such as eating breakfast or brushing in the morning. We also require the worker's device to have at least 25% of its battery capacity charged, to prevent CrowdLearner from depleting the device's battery power.

Validating Samples

Prior work in crowdsourcing has used verification questions and obtaining multiple samples in order to validate samples [14]. CrowdLearner requests multiple samples from workers for each of the targets. Since CrowdLearner focuses on building recognizers, it can use classifiers that are resistant to outliers such as SVM as is the case with our current prototype. Alternatively, we can train a k-nearest neighbor classifier in parallel to detect and discard outliers, although false detection may skew sample distributions.

Feature Extraction

Feature extraction is dependent on the types of sensors selected. In CrowdLearner, each sensor employs a different process to generate features, e.g., a featurization for touchscreen traces might not be appropriate for accelerometer readings. When multiple sensors are selected for a task, CrowdLearner combines the feature vector of all sensors to form a single vector. This modular design allows us to easily incorporate new sensors when they become available. For accelerometer or gyroscope readings, we currently use common features such as means, standard deviations, FFT coefficients, cepstral coefficients, spectral entropy, correlations, and integrations [17]. For the touchscreen, we use features similar to prior work for gesture and handwriting recognition [19]. When combining the features from different sensors, we always concatenate them in the same order.

Generating a Recognizer through Continuous Learning

CrowdLearner continuously updates the recognizer as workers provide more samples and allows the developer to monitor the progress of the task and acquire the latest recognizer or one generated in the past at any time. We first seed our learning process with a *null* recognizer, which simply returns a null target for any sample received. We keep a counter for the number of errors that the recognizer has made for predicting each newly observed sample. Once the error count reaches a threshold—20 in our case, we generate a new recognizer with the data that has been collected, and reset the error counter and repeat the process.

In the beginning of the process, as more samples come in, the classifier is likely to make more errors and rebuilds (retrains) more frequently. With this *valved continuous learning* process, CrowdLearner continuously learns from the samples that it receives, but also does not spend too many cycles retraining a recognizer when it would only provide marginal benefit. As a proof of concept, we use an implementation of Linear SVM, which performs reasonably well and quickly for many classification purposes [3].

To present the learning progress to the developer, CrowdLearner visualizes the performance of a generated recognizer based on 10-fold cross validation that splits the collected samples based on workers.

EVALUATION

We deployed CrowdLearner internally in a large IT company to 72 participants (M=51, F=21) for a period of 2.5 weeks. We recruited workers by emailing company-wide lists, and sent the mobile client via email. We never met or had any direct contact with the workers. Our workers were between 18 and 55 years old, with approximately 75% of ages falling uniformly between 18 and 35. Workers had a variety of occupations including Administrative, Legal, HR and engineering. We evaluated CrowdLeaner through 6 recognition tasks. At the end of our study, we awarded workers monetary compensation for the credits earned by emailing gift certificates. Workers were paid $0.10 for each question they answered, and up to $10 per task. This remote hands-off deployment is a good approximation for a realistic crowdsourcing platform.

Experimental Tasks

We created 4 of the recognition tasks ourselves. The Media Player Gestures task was used to generate a recognizer for 10 stroke gestures for controlling a media player application: Play, Pause, Stop, Rewind, Fast Forward, Previous, Next, Accept, Reject, and Help. The $1 Recognizer Gestures task concerned 16 stroke gestures as defined by Wobbrock *et al.* [29]. The Phone Call Motion task involved 3 motion gestures for receiving phone calls: Shake, Rotate, and Place Phone To Ear. The Daily Physical Activities task was to detect four user activities: Still (sitting or standing), Walking, Biking, and Riding in a car or on a bus. The Daily Physical Activities task employed opportunistic sampling.

To understand how developers would react to CrowdLearner, we invited two developers, who were not involved with our project, to use CrowdLearner to create mobile recognizers that can be useful to their projects. One developer wanted to recognize handwritten digits 0 through 9 for a Number Pad application. The other developer created the Directional Motion task for recognizing 5 motion gestures: Left, Right, Up, Down, and Double Flip [22]. The directional commands are flicks toward the designated direction and back to the initial position similar to those defined in [18].

Experimental Results

To deal with device variation, CrowdLearner builds a recognizer for each type of device available in the worker pool. Table 1 shows the recognition performances for Galaxy Nexus devices, which were most common in our pool of devices. In general, the tasks involving touchscreen stroke gestures performed well and it took less than 2 hours and $10 for their recognizers to reach 80% of the final accuracy. With respect to motion gesture recognition, recognizers reached 80% of the final accuracy within 3 hours.

The Phone Call Motion task performed reasonably with 77% final accuracy. However, the Directional Motion task performed poorly with final accuracy of 42%. Upon our examination of the results, we noticed that the recognizer performed well with respect to detecting Double Flip and separating vertical and horizontal flicks. However, it failed to discern different directions within vertical or horizontal movements—it could not separate Up from Down or Left from Right. By carefully examining the collected data, we found although our current featurization mechanism for accelerometer and gyro readings is able to capture the overall movement profile of the phone, e.g., rotating versus shifting, it is ineffective to capture detailed sequence variation in motions such as these quick directional flicks. We can address this issue by introducing features that can better capture the temporal characteristics of motion or employing more advanced featurization techniques (e.g., [17]).

Opportunistic Learning

While the Daily Physical Activities task achieved 78% accuracy, it was most effective at recognizing sitting or standing events. Specifically, the produced recognizer detected over 99% of sitting events. However, it only recognized 2.5% of walking and 2.5% of riding in a car or bus activities and did not recognize any of the biking activities (3 total). We attribute this performance to the scarcity of non-stationary activities during data collection. Moreover, physical activity recognition using a single sensor pack with a non-static location on the body is challenging [17]. Nevertheless, we still observed some interesting trends when examining the data for the Daily Physical Activities task.

Based on our logs, the workers answered only 25% of all CrowdLearner opportunistic notifications. We observed that motion activities were generally sparse. In one week of data collection (including the weekend), we collected 285 samples for Still (sitting or standing), 40 for Walking, 3 for Biking, and 44 for Riding on a bus or in a car.

There are several ways to address the scarcity of sampled target events. One approach is to employ a base event detector that decides if an event of interest is likely happening so as to help CrowdLearner determine when to perform opportunistic sampling. However, developing an effective

Task (Number of Targets)	Workers	Samples	Time to 80% A*	Funds to 80% A*	Final Accuracy (A)
Media Player Gestures (10)	52	2119	1 hr	$5	99.2%
$1 Recognizer Gestures (16)	48	2200	1.5 hrs	$5	98.4%
Number Pad Gestures (10)	47	2473	1 hr	$10	95.2%
Phone Call Motion (3)	45	1487	3 hrs	$12	77.4%
Directional Motion (5)	34	1227	3 hrs	$5	42.2%
Daily Physical Activities (4)	30	368	2 hr	$3	78.1%

Table 1: Participation and Accuracy for CrowdLearner tasks (*80% of Final Accuracy).

base event detector itself might be challenging. A developer can create the detector in the laboratory or also in CrowdLearner.

By examining the occurrence of these activities (see Figure 5), we found the workers were most likely to be on the move during the evening, and often moved around noon-time. We conjecture that these events correspond to heading home from work and going for lunch, respectively. Considering the emerging trends, we devised a model for sampling events based on their occurrence, which we will discuss in detail in the Collaborative Temporal Sampling section.

Sampling in the Wild

To understand how CrowdLearner data, which is collected in the wild, is different from data collected in laboratories, we compared the dataset of CrowdLearner's $1 Recognizer Gestures task to the original $1 Recognizer dataset. The original dataset was collected in the laboratory from ten individuals and is published on the web [29]. CrowdLearner aims to collect and train on data generated in users' natural environments. To test out the theory that CrowdLearner data collection exhibits more variation than that conducted in a laboratory environment, we ran the $1 Recognizer algorithm—based on template matching (or the Nearest Neighbor)—on both datasets. To be comparable with the original $1 dataset, we limited our dataset to samples collected from 10 individuals only, and manually checked them for mislabeled data. When manually (visually) examining our entire dataset of 2200 samples, we found 19 outliers. 12 of these outliers were gestures for another label and 7 were not distinguishable to be any of the targets. Since each gesture might have an unequal number of samples in the CrowdLearner dataset due to the uncontrolled collection process, we sampled the original $1 dataset to have the same distribution as our dataset across the three gesture speeds present in the original $1 dataset. We then trained and tested the $1 Recognizer on all possible training and testing pairs across the two datasets. In all tests, we trained on half the data and tested on the remaining half.

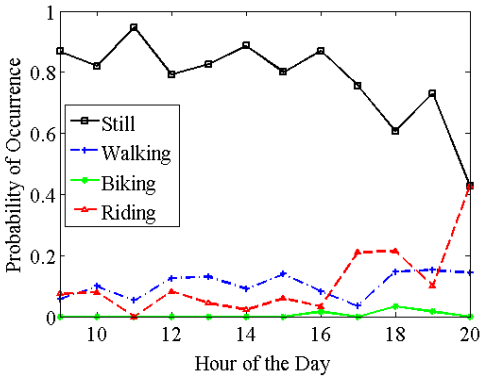

Figure 5. Probability of observing activities aggregated across users. Our workers were more likely to be on the move at noon and during the evening.

Training and testing on the $1 dataset produced an accuracy of 99.2%. This was not surprising as it is in line with the published performance of the $1 Recognizer. Training and testing on the CrowdLearner's $1 dataset produced 81.2% accuracy. However, the most interesting results were obtained when we trained on CrowdLearner's $1 dataset and tested on the original $1 dataset (87.9% accuracy) versus training on the original $1 dataset and testing on CrowdLearner's $1 dataset (77.7% accuracy). As $1 Recognizer is a template matching gesture recognition algorithm, these results indicated that gestures collected in the wild (via CrowdLearner) exhibit more variation. Specifically data produced by the crowd is more representative as it can describe data present in the $1 dataset, however, is not as describable using the $1 dataset templates.

Feedback from the Workers

At the end of our study, we asked our worker participants to provide feedback through a web-based survey. We asked workers general questions regarding CrowdLearner and also specific questions regarding each of the tasks. Participating workers rated ease of use, usability of the user interface, awareness of availability of tasks, adequacy of compensation per question, awareness of total rewards, and the possibility that they would perform tasks using the CrowdLearner app in the future on a 5-point Likert scale. More than 75% of the workers responded positively to each of the aforementioned attributes of the system (Agree or Strongly Agree).

Two worker participants commented that the instructions for motion-related tasks (target and associated video) were given too often and requested to be able to skip them faster. One worker mentioned that because of the audio notification for the count down timer, he was not able to perform the Phone Call Motion and Directional Motion tasks quietly during down times (e.g., on his subway commute) without disturbing others in his vicinity. This would suggest providing an option of audio or vibration notifications at the end of the count down timer. He preferred more silent tasks during these periods (e.g., Media Player Gestures).

With respect to the Daily Physical Activities task, most workers found the vibration helpful in remembering their state (85% Agree or Strongly Agree). Similarly, they found that knowing the time that the activity occurred was helpful (over 85% Agree or Strongly Agree). However, workers were divided with respect to usually not having their phone on them when the phone vibrated (median=Neutral) or not being able remember the event (median=Disagree). This confirmed our log-based finding that workers answered 25% of the opportunistic notifications. One worker did not find the vibration notifications useful as she used a pouch to hold her phone. Another worker suggested giving workers the ability to inform CrowdLearner about their activities beforehand: "allow the user to say 'I'm about to go Biking'" to enable sampling of non-stationary activities.

Feedback from the Developers

Both developers found CrowdLearner easy to use with respect to creating tasks and targets, associating media with targets, and monitoring the performance of the recognizers, responding positively with Agree and Strongly Agree on a 5-point Likert scale.

The creator of the Directional Motion task (referred to as C1 hereafter) commented that it would be useful to allow workers to try out a task without affecting the results for the recognizer, specifically for when they are unsure as to how to perform a certain gesture. This would be a valuable improvement as some worker participants also mentioned that they were not sure what to do when first starting a task. C1 felt that the produced recognizer was muddled by too many poor initial samples. He also felt that it would be useful to allow the developer to select a particular set of samples to train the final recognizer. The creator of the Number Pad Gestures task (C2) also suggested being able to select a specific set of samples for training and being able to look at and validate the samples provided by workers. C2 recommended adding information about whether the recognizer was able to classify the samples successfully, and presenting the developer with visualizations for clusters of samples to support inference of general types and trends.

COLLABORATIVE TEMPORAL SAMPLING

One important lesson that we learned from the experiment is that it is challenging to determine when to perform opportunistic sampling. Sampling uniformly in time can be inefficient, especially when some of the target events occur less frequently, such as biking or driving. By examining the dataset of the Daily Physical Activities task, we discovered that the occurrence of target events follows temporal trends. As a result, we devised a model to discover these trends in order to enable more intelligent sampling.

With this model, CrowdLearner can sample conditionally based on when an event of interest is likely to occur. More specifically, when a worker's device wakes up, it first checks with the CrowdLearner service to determine if the wake up time is associated with a period of uninteresting samples, e.g., when workers would be most likely sitting. If this is the case, CrowdLearner would inform the device not to sample. We present here the result of an offline analysis of the Daily Physical Activities task to evaluate our conditional sampling model.

The conditional sampling model consists of two components. One component captures general trends for the entire worker population, while the other captures trends for a specific worker. These two components perform in an online fashion in that they are trained based on prior days of data and then combined to decide whether to sample. When an opportunistic task first starts, any time of the day is a "good" time for sampling—a uniform distribution. However, each day, the model is updated to learn from prior days' samples. Both the individual and population components are trained in the same fashion. For the individual component, only a single worker's data is used for training, whereas for the global component, data from the entire population is used.

We train each component using a system of linear equations for each target, and add extra constraints to smooth the data. Specifically, we divide days to N timeslots, where N is dependent on the sampling interval. For our case, since we sample approximately once an hour, N is 168. Timeslots are indexed in day-major order with the first timeslot starting at 00:00 on Sunday. We add an equation to the system for each timeslot for a particular label, where the probability of occurrence of event x in that timeslot is defined as:

$$p_i = n_x / n_{total}$$

We also add constraints to smooth the probabilities:

$$(\lambda/2)\,p_{i-1} + (1-\lambda)\,p_i + (\lambda/2)\,p_{i+1} = p_i$$

λ is between 0 and 0.5 and controls the smoothing of each p_i. Additional constrains may be added to control variations in weekdays or weekends. Using this system of equations and a least squares solver, the probabilities of occurrence for each target can be computed to create a sampling model. Figure 6 presents the model for riding in a vehicle.

Once probabilities are generated for each target, sampling can be done to either obtain a more diversified set of samples, or a different criterion based on targets deemed interesting. We combine the individual and global models linearly:

$$p_i = (1-\alpha)\,p_i^{global} + \alpha\,p_i^{individual}$$

where α is between 0 and 1. α is learnt based on correct predictions using the individual component, in the following fashion:

$$\alpha(t+1) = \alpha(t) + \gamma\left(F\left(p_{i,t}^{individual}\right)\right)$$

where γ controls how quickly α updates, and F outputs 1 for correct predictions and -1 for incorrect ones. For our evaluation, we set alpha to be a constant of 0.5.

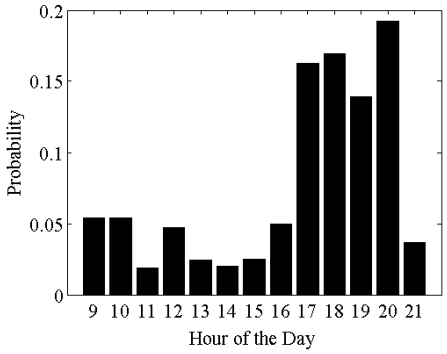

Figure 6. Global sampling model trained for riding in a vehicle based on data collected for the Daily Physical Activity Task. Workers are more likely to ride in a vehicle early in the morning and in the evening hours.

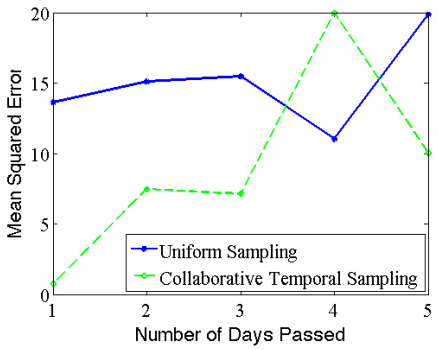

Figure 7. Collaborative temporal sampling performance compared to uniform sampling for the Riding in a Car or on a Bus target. The Collaborative Temporal Sampling model has a jump after the 4th day as the worker shifts to also drive in the morning.

Using the aforementioned approach, we created a sampling model for the Daily Physical Activity task and verified our results by training on prior days of data, and testing on current day's events. Specifically, we tested the model against a uniform sampling approach for the riding in a car or on a bus target. The results are based on data from the global model and one worker's individual model. Figure 7 shows the mean squared error (MSE) for the conditional sampling model and the baseline approach using uniform sampling. The error is computed as the temporal distance (as the number of hours) between when the model predicts an event would occur and when the event actually occurred. In Figure 7, we see a jump in MSE for our model after day 4. This corresponds to a shift from the worker's habits to also drive in the morning. We see great potential in using collaborative temporal sampling models to learn about people's activities in an opportunistic fashion. The model helps improve the answering rate for opportunistic sampling and optimizes the power and computation consumption on the worker's device to obtain useful samples.

CRITICAL REFLECTION

CrowdLearner is the first general end-to-end framework for non-machine learning experts to quickly create recognizers with ecologically valid data. It eliminates the effort for data collection and the technical barrier for training and testing recognizers. However, its machine learning component currently offers a single classification model and a limited set of features, which are insufficient for all kinds of recognition tasks. As our experiments indicated, CrowdLearner performed well for stroke gestures but less than ideal for motion-based recognition. We can address this issue by offering a suite of machine learning techniques and featurization mechanisms and enabling CrowdLearner to evaluate these alternatives in parallel—automatically selecting an optimal one for a specific task.

It is also possible to expose more learning parameters to the developer for finely tuning a recognizer. For example, popular ML toolkits such as Weka allow developers to adjust various ML parameters specific to a model. However, adjusting these parameters often requires the developer to have a deeper understanding about the machine learning techniques, which deviates from our motivation to support developers who have no ML expertise. We can potentially benefit from extensive work in interactive ML (e.g., [20]) that attempts to make it easy for average developers to adjust and iterate on ML techniques.

The performance of CrowdLearner is also impacted by sensors that are available on a target mobile device and the type of recognition task that the developer orders. Given the sensors on a mobile device, the targets in a task might be fundamentally ambiguous (e.g., sitting or standing still) or too fine-grained to be captured (e.g., minor gradual movements of the device). As such, it is crucial to communicate with developers about what CrowdLearner is capable of recognizing and explain reasons for poorly performed tasks.

Similar to much design-oriented work, iteration is essential. More relevantly, Patel *et al.* reveal the critical role that iteration plays in designing a machine learning method [20]. Currently, CrowdLearner supports in-flight changes such as adding more funds to a task, editing the task's name and instructions, adding or removing targets, and changing the targets' associated media. It would be valuable to explore a more systematic support for iteration in CrowdLearner. We can potentially add advanced iteration support that are conceptually akin to Version Control System mechanisms, e.g., versioning, merging and branching [23], for better managing changes and supporting collaboration in CrowdLearner.

CrowdLearner adopts a continuous learning concept by rebuilding and improving recognizers as more workers are added and more samples are collected. This process can be furthered by also automatically synchronizing the performance of the CrowdLearner recognizers that are active in the field. Specifically, these recognizers could send their incorrectly recognized observations to the CrowdLearner service for further improvement of recognition. CrowdLearner could also adopt active-learning concepts to achieve greater accuracy with fewer samples by requesting workers to perform specific gestures or actions [4], e.g., gestures that are often misclassified.

CrowdLearner mitigates the problem of variability in devices by maintaining profiles of workers' devices and building a recognizer for each type of device present. As an artifact of this approach, popular devices will have more training data and potentially more accurate recognizers. However, there still exists variability in samples with respect to workers. A problem that is inherent to the use of crowd workers is that characteristics such as age, gender, and physical attributes can affect the samples provided. We do not explore this problem ourselves, however, prior work by Lane *et al.* has presented promising results in using Social Networks to identify people with similar characteristics to build recognizers that are tailored towards individuals and more specific user types [16]. Similar approaches may be

used in CrowdLearner to improve recognizer performance. Another solution would be to obtain demographic information from workers and perform intelligent crowd sampling based on demographics and trends, similar to how we use activity trends for the temporal conditional sampling.

CrowdLearner currently enables a "free market" in which developers determine the allocations of funds for tasks and workers freely opt in or out. However, it is valuable to offer additional support for fund allocation, such as suggesting how much a sample or interaction is worth. It is also worth investigating how to incentivize workers for specific targets, e.g., by increasing compensation for targets with fewer samples. In addition to using monetary compensation as a general incentive, we can potentially leverage game-based schemes (e.g., [13]) for certain tasks such that workers can motivate each other to contribute more quality data.

CONCLUSION

We presented CrowdLearner, a crowdsourcing platform that automatically creates recognizers for non-machine learning experts for mobile applications. CrowdLearner reduces the barriers presented by recruiting participants, data collection and labeling, and the technical knowledge required to train recognizers. We discussed our design and implementation of CrowdLearner as a cloud service and presented the findings of our evaluation through experimenting with 6 recognition tasks with 72 workers and 2 developers. Our results provide insight into designing future crowdsourcing systems for machine learning tasks.

REFERENCES

1. Ashbrook, D. and Starner, T. MAGIC: a motion gesture design tool. In *Proc. CHI 2010*, ACM Press (2010).
2. Bernstein, M.S., Little, G., Miller, R.C., Hartmann, B., Ackerman, M. S., Karger, D.R., Crowell, D. and Panovich, K. Soylent: a word processor with a crowd inside. In *Proc. UIST 2010*, ACM Press (2010).
3. Boser, B., Guyon, I.M. and Vapnik, V.N. A training algorithm for optimal margin classifiers. In *Proc. COLT 1992*, ACM Press (1992).
4. Burr, S. Active Learning Literature Survey. Comp. Sciences Tech. Report 1648, University of Wisconsin-Madison, 2009.
5. Campbell, A.T., Eisenman, S.B., Lane, N.D., Miluzzo, E. and Peterson, R.A. People-centric urban sensing. In *Proc. WICON 2006*, ACM Press (2006).
6. Cuervo, E., Gilbert, P., Wu, B. and Cox, L.P. CrowdLab: An Architecture for Volunteer Mobile Testbeds. In *Proc. COMSNETS 2011*, IEEE (2011).
7. Das, T., Mohan, P., Padmanabhan, V.N., Ramjee, R. and Sharma, A. PRISM: platform for remote sensing using smartphones. In *Proc. MobiSys 2010*, ACM Press (2010).
8. Downs, J.S., Holbrook, M.B., Sheng, S., and Cranor, L.F. Are your participants gaming the system? screening mechanical turk workers. In *Proc. CHI 2010*, ACM Press (2010).
9. Froehlich, J., Chen, M.Y., Consolvo, S., Harrison, B. and Landay, J.A. MyExperience: a system for in situ tracing and capturing of user feedback on mobile phones. In *Proc. MobiSys 2007*, ACM Press (2007).
10. Gajos, K.Z., Reinecke, K. and Herrmann, C. Accurate Measurements of Pointing Performance from In Situ Observations. In *Proc. CHI 2012*, ACM Press (2012).
11. Hartmann, B., Abdulla, L., Mittal, M. and Klemmer, S.R. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proc. CHI 2007*, ACM Press (2007).
12. Heer, J. and Bostock, M. Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In *Proc. CHI 2010*, ACM Press (2010).
13. Henze, N., Pielot, M., Poppinga, B., Schinke, T. and Boll, S. My App is an Experiment: Experience from User Studies in Mobile App Stores. *International Journal of Mobile Human Computer Interaction* 3, 4 (2011), 71-91.
14. Kittur, A., Chi, E.H. and Suh, B. Crowdsourcing user studies with Mechanical Turk. In *Proc. CHI 2008*, ACM Press (2008).
15. Kittur, A. and Kraut, R.E. Harnessing the wisdom of crowds in wikipedia: quality through coordination. In *Proc. CSCW 2008*, ACM Press (2008).
16. Lane, N.D., Xu, Y., Lu, H., Campbell, A.T., Choudhury, T., and Eisenman, S.B. Cooperative Communities (CoCo): Exploiting Social Networks for Large-scale Modeling of Human Behavior. *Pervasive Computing*, IEEE (2011).
17. Lester, J., Choudhury, T. and Borriello, G. A practical approach to recognizing physical activities. In *Proc. PERVASIVE 2006*, Springer (2006).
18. Negulescu, M., Ruiz, J., Li, Y. and Lank, E. Tap, Swipe, or Move: Attentional Demands for Distracted Smartphone Input. In *Proc. AVI 2012*, ACM Press (2012).
19. Ouyang T. and Li, Y. Bootstrapping personal gesture shortcuts with the wisdom of the crowd and handwriting recognition. In *Proc. CHI 2012*, ACM Press (2012).
20. Patel, K., Bancroft, N., Drucker, S.M., Fogarty, J., Ko, A., and Landay, J.A. Gestalt: Integrated Support for Implementation and Analysis in Machine Learning Processes. In *Proc. UIST 2010*, ACM Press (2010).
21. Ra, M., Liu, B., La Porta, T.F. and Govindan, R. Medusa: a programming framework for crowd-sensing applications. In *Proc. MobiSys 2012*, ACM Press (2012).
22. Ruiz, J., and Li, Y. DoubleFlip: A Motion Gesture Delimiter for Mobile Interaction. In *Proc. CHI 2011*, ACM Press (2011).
23. Ruparelia, N. B. The History of Version Control. SIGSOFT Softw. Eng. Notes, 35, 1 (2010), 5-9.
24. Shepard, C., Rahmati, A., Tossell, C., Zhong, L. and Kortum, P. LiveLab: measuring wireless networks and smartphone users in the field. SIGMETRICS Perform. Eval. Rev. 38, 3 (2011), 15-20.
25. Song, Y., Lasecki, W., Bigham, J., and Kautz, H. Training Activity Recogntion Systems Online Using Real-Time CrowdSourcing. In *Proc. UBICOMP 2012*, ACM Press.
26. Von Ahn, L., Blum, M. and Langford, J. Telling humans and computers apart automatically. *Commun. ACM 47, 2* (2004), 56-60.
27. Westeyn, T., Brashear, H., Atrash, A. and Starner, T. Georgia tech gesture toolkit: supporting experiments in gesture recognition. In *Proc. ICMI 2003*, ACM Press (2003).
28. Witten, I. H. and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2 (2005).
29. Wobbrock, J.O., Wilson, A.D. and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. UIST 2007*, ACM Press (2007).