Crowdsourcing with R and the MTurk API

Thomas J. Leeper*

March 15, 2013

Abstract

This article introduces the collection of social science data through the Amazon Mechanical Turk (MTurk) crowdsourcing platform, taking advantage of a new package for R that connects to the MTurk API. Through two use cases — (1) human coding of materials and (2) a panel survey-experiment — the utility of MTurk as a research platform is demonstrated, as is the use of R to leverage cloud-based data APIs.

Political scientists are in constant need of human-generated data. Responses to survey questions and reactions to experimental stimuli, manual coding of visual, auditory, and textual data, and the recorded behavior of individuals engaged in naturalistic or artificial behaviors are the bread and butter of contemporary quantitative research on politics. Yet access to large numbers of humans capable of producing these data is often a major logistic and financial challenge for researchers doing everything from large coding projects to pretesting of questionnaires to the recruitment of participants for full research studies. This search for humans willing to generate the kinds of data social scientists demand has led to major recent interest in online data collection generally (e.g., Iyengar 2010; Iyengar and Vavreck 2011; Vavreck and Iyengar 2011) and, more recently, crowdsourcing platforms in particular (Schmidt 2010; Chen, Menezes, and Bradley 2011) to provide these kinds of data at low cost. Among these platforms, Amazon Mechanical Turk (MTurk) stands out as one of the largest and most useful platforms for political science research (Berinsky, Huber, and Lenz 2010).

Leveraging MTurk to move traditionally pencil-and-paper processes managed locally (like laboratory experiments or the hand-coding of materials by undergraduate research assistants) into a cloud-based process can dramatically lower the time and resource expenditure involved with such efforts, as well as streamline research workflow. This article advocates for and describes how to move these social science data needs into the cloud using an R package called MTurkR (Leeper 2012), while also encouraging the development of packages that connect researchers to potentially valuable sources of API-based data. Integrating data APIs into R enables researchers to work with data in a familiar programming environment, directly link data collection and data analysis (thereby eliminating the number of steps and amount of time involved), and aid the reproducibility of research by focusing on using (and making) publicly available data that can be readily accessed from the cloud using code that can be shared to produce identical results.[1]

*Department of Political Science, Aarhus University

[1]Another approach is the dvn package, which provides access to The Dataverse Network repository API (**?**) or the ROpenSci project, which does the same for a large number of (mostly natural science) APIs.

1 MTurk: Introduction and Core Concepts

MTurk is a crowdsourcing platform designed to provide "human intelligence" for tasks that cannot be readily, affordably, or feasibly automated (Amazon.com 2012). The service provides researchers with useful infrastructure for the generation of common social science data. While many early adopters of MTurk as a data generation tool came from computer science (Mason and Suri 2011; Kittur, Chi, and Suh 2008), more recent attention has emerged among social scientists (Buhrmester, Kwang, and Gosling 2011; Berinsky, Huber, and Lenz 2010; Paolacci, Chandler, and Stern 2010). Use of MTurk in general reflects a clear move toward cloud-based research, yet substantial barriers to entry for sophisticated use of MTurk exist — namely the limited functionality of the service's online graphical Requester User Interface (RUI) for anything other than linking to off-site survey tools and the difficulty (for most non-engineers) of using MTurk's other access points. Frankly, using MTurk for complicated social science research tasks is quite challenging and that difficulty limits the ability of researchers to innovate and test the limits of the platform for generating useful social science data.

1.1 Key Terms and Concepts

The service connects *requesters*, who are willing to pay *workers* to perform a given task or set of tasks at a given price per task. These "Human Intelligence Tasks" (HITs), are the core element of the MTurk platform. A HIT is a task that a requester would like one or more workers to perform. Every HIT is automatically assigned a unique HITId to identify it in the system. Performance of that HIT by one worker is called an *assignment*,[2] such that a given worker can only complete one assignment per HIT but multiple workers can each complete an assignment for that HIT (up to a maximum set by the requester). Multiple HITs can be grouped as a HITType, allowing a worker to complete multiple similar HITs (e.g., coding of several different texts) with ease.

MTurk operates on basic market principles of supply and demand. Workers can choose which HITs to complete and how many HITs they want to complete at any given time, depending on their own time, interests, and the payments that requesters offer.[3] A requester can offer as low as $0.005 per assignment, but if other requesters offer HITs that add up to a higher hourly wage, workers can choose to take their labor elsewhere. Similarly, requesters can pay any higher amount they want per assignment, but that may not be cost-effective given market forces. MTurk also charges a surcharge equal to 10% of all worker payments.

Once a worker completes a HIT, the requester can *review* the assignment, determining whether the responses or answers provided by the worker are satisfactory. If so, an assignment can be *approved* and the requester pays the worker the predetermined per-assignment price for the HIT (and no more or no less; the price is fixed in advance). If the requester thinks the work merits additional compensation (or perhaps if workers are rewarded for completing all HITs of a given HITType), the requester can also pay a *bonus* of any amount to the worker at any point in the future. If work is unsatisfactory, the requester can *reject* work and thereby deny payment (but has to justify that rejection to the worker), freeing the assignment for completion by another worker.

While these are the basic functions of MTurk, additional functionality is hidden (or at least inconvenient) via the RUI. In particular, while the RUI provides some ability to control what types of workers are eligible to complete a HIT (based upon their HIT approval rate, country of residence, and a few other measures) through *qualification requirements*, the functionality provided

[2] HITs, Workers, Requesters, and Assignments each have a unique ID, which become essential for using MTurkR.

[3] Workers also communicate about the quality of HITs and requesters on fora such as `http://turkopticon.differenceengines.com/`, `http://mturkforum.com/`, and `http://www.turkernation.com/`.

by the RUI makes it difficult to organize large numbers of workers using these qualifications as well as create new qualifications. Paying bonuses to and contacting workers is similarly quite tedious. The Requester API, by contrast, provides full access to the underlying web application that runs MTurk and MTurkR accesses that API through a familiar programming environment.

2 MTurkR Package

Before using MTurk or the MTurkR package, one needs to have an MTurk requester account, which can be created at `http://www.mturk.com`, and deposit money in that account.[4] To use MTurkR (or any tool for accessing the Requester API), one additionally needs to retrieve Amazon Access Keys from `https://aws-portal.amazon.com/gp/aws/securityCredentials`. The *keypair* is a linked "Access Key ID" and a "Secret Access Key" that, in combination, allow MTurkR to access the API. In MTurkR, the keypair is a two-element character vector with the Access Key ID as the first element and the Secret Access Key as the second element. This keypair is used to authenticate API requests, which are HTTP communications sent from MTurkR running on a local workstation to the MTurk server.[5]

MTurkR provides access to every part of the MTurk API through a set of easy-to-use, but powerful R functions that provide both simplicity for the beginning requester and robust functionality for managing everything from a single survey-type HIT with a few hundred responses to a massive HITType with large numbers of HITs, assignments, and workers. The package additionally provides an array of novel tools for managing workers (i.e., with qualifications, email notifications, and bonus payments).[6]

MTurkR automates the request and authentication process, such that no knowledge of HTTP requests or authentication is required to use it.[7] One need only provide a keypair and know the particular operation to be performed. Once performed, the MTurk service verifies that the request is valid. The API then returns an XML *response*, which MTurkR (generally) converts into R data structures that can be directly used in analysis with no need for manual conversions to R-readable data.[8]

As a brief example, the simplest operation is to check the balance in one's requester account. To do so (or before doing any operation with MTurkR), the keypair should be recorded with a call to `credentials()`, which takes two character strings as its parameters: (1) an AWS Access Key ID, and (2) an AWS Secret Access Key.[9] A call to `AccountBalance()` then queries the API and returns a simple character string showing the remaining balance in the requester account.

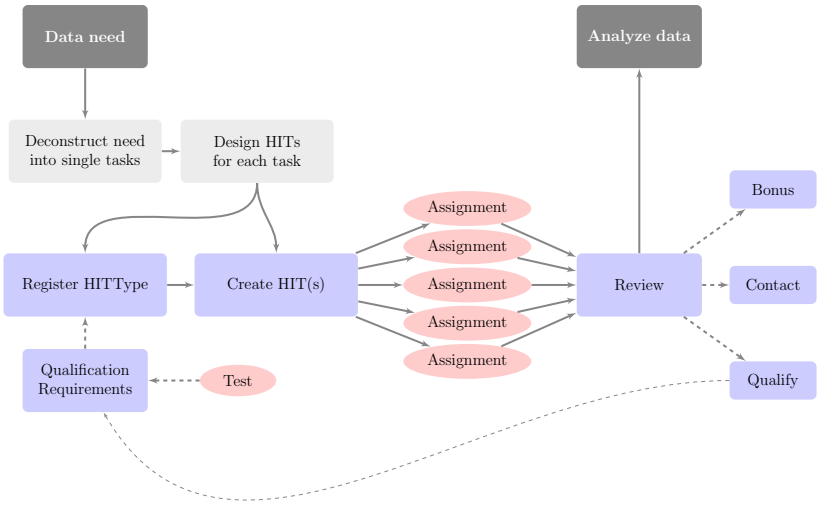

Figure 1: An MTurk Workflow

3 Two Political Science Use Cases

By providing direct and complete access to the API, MTurkR removes rather than imposes (as the RUI does) limitations on what researchers can crowdsource. To describe how researchers might utilize MTurk, this section provides two examples of MTurk as a social science data platform using MTurkR code to demonstrate how to easily manage that data collection. In both cases, a data need — first, for human coding of newspaper articles and, second, participation in a panel experiment — starts the research workflow and that need is broken down into MTurk HITs, which are completed by workers and reviewed by the requester via MTurkR, before the completed data are extracted from MTurk for analysis immediately in R. Figure 1 lays out this basic process with solid lines representing necessary actions and dashed lines showing optional actions.

3.1 First Use Case: Content Analysis

One of the most prevalent research techniques in political science is content analysis, be it coding of newspaper articles, images, speeches, campaign advertisements, or websites. While this is often done with an army of undergraduate research assistants or, more recently, via a number of automated techniques, crowdsourcing of human coding remains underutilized. Following the workflow described in Figure 1, a coding task consists of a need for a rectangular, item-by-attribute dataset for all of the items to be coded (items for our purposes are newspaper articles). Deconstructing

[4]Note: The API does not allow you to add funds to your account, which must therefore be done through the web interface: `https://requester.mturk.com/mturk/prepurchase`.

[5]To move further into the cloud, one could also run MTurkR remotely through a server-based implementation of R, such as RApache (`http://rapache.net/`), RStudio Server (`http://www.rstudio.com/ide/docs/server/getting_started`), or Amazon's EC2 (`http://aws.amazon.com/ec2/`).

[6]MTurkR also includes a lightweight graphical user interface, which will not be discussed here.

[7]This functionality is provided by calls to the RCurl package (Lang 2012*a*).

[8]The XML parsing is provided by the XML package (Lang 2012*b*). The raw XML responses are stored, by default, in a tab-separated value log file in the user's working directory.

[9]The function stores this as a two-element character vector, which is then referenced by default by all other MTurkR functions.

this need into individual tasks, requires the construction of a coding sheet to be used on each item (a HIT will consist of coding one item), which can be written as an HTML form (or prepared in WYSIWYG form in the RUI).

To ensure quality, however, it may also be useful to restrict who can code the items to those who have demonstrated that they can accurately perform the task. Thus, in addition to designing each of the HITs, a qualification test should also be designed that will simulate the coding process and allow the requester to evaluate whether or not a particular worker qualifies to work on the coding.[10] Once the coding sheet and qualification test are written, implementing the coding process via MTurkR simply proceeds left-to-right across the tasks in Figure 1.

First, a HITType is created with the qualification test to display all of the coding items together on the MTurk worker site. Then, each HIT is created by associating it with the HITType. Workers then complete the qualification test and, if successful, are eligible to complete assignments. Once assignments are completed, the requester can review those assignments (approving acceptable work and rejecting all else) and then extract the data and analyze. Workers who pass the qualification test but fail to perform well on the work can have their qualification revoked, preventing them from completing more of the assignments. The HITType has four required and three optional characteristics, but good practice is to specify all of them:

- Title (required)
- Description (required)
- Reward (required)
- Duration (required)
- Keywords
- Assignment Auto-Approval Delay
- Qualification Requirements

To register a HITType, these characteristics need to be defined in a call to `RegisterHITType()`. But, first, we will create a Qualification that tests workers' ability to code, along with an AnswerKey that MTurk will use to automatically score and qualify workers who complete the test.

```
newqual <- CreateQualificationType(name="Coding Test", description="Test of coding ability",
                                   status="Active", test.duration=seconds(hours=1),
                                   test=QuestionForm, # a character string containing a QuestionForm
                                   answerkey=AnswerKey) # a character string containing an AnswerKey
```

That QualificationRequirement can then be attached to a new HITType, along with the other parameters:

```
q1 <- GenerateQualificationRequirement(newqual$QualificationTypeId, ">=", 100, preview=TRUE)
register <- RegisterHITType(title="20-Question Survey",
                            description="Take a five-question survey about your political opinions
                                         from researchers at Aarhus University.",
                            reward=".25", duration=seconds(days=1,hours=8),
                            keywords="survey, question, answers, research, politics",
                            qual.req=q1)
```

Creating a HIT using HITLayout parameters is also incredibly easy. The coding sheet template can be saved in the RUI with a placeholder of the style `${message}` where the text of each article will be inserted. Once the HIT template is created in the RUI, you can retrieve its LayoutId from `https://requester.mturk.com/hit_templates` by clicking on the "Project Name" on the

[10] For technical reasons, this test needs to be written in the proprietary QuestionForm (XML) format, with the advantage being that MTurk can automatically score workers' qualification tests.

Figure 2: Retrieving a HIT LayoutId from the RUI

left-hand side of the screen. A pop-up window will appear displaying the LayoutId and the name of any placeholders (see Figure 2). Those placeholders can be replaced with the relevant material when `CreateHIT()` is called, by specifying `hitlayoutid` and `hitlayoutparameters`. Assuming the newspaper texts are loaded into R in a list of character strings called `articles`, one can simply iterate through that list to create each HIT, placing the text of each article in the `message` layout parameter:

```
layout <- "2J8JQ2172Z9990XQXO3VKBU13LWRZO"
hits <- vector(length=length(articles), mode="character")
for(i in 1:length(articles)){
   hits[i] <- CreateHIT(hit.type=register$HITTypeId, hitlayoutid=layout,
                        hitlayoutparameters=GenerateHITLayoutParameter("message",articles[[i]]),
                        annotation=paste("Article to code",i),
                        expiration=seconds(hours=1),
                        assignments=2)$HITId
}
```

This loop will return the HITId for each new HIT, which the above code stores in a character vector called `hits`. The HITId for each HIT (and the `annotation` value, which provides a private description of the HIT visible only to the requesters) can be retrieved at any time using `SearchHITs()`. Once a HIT is created, the simplest (and perhaps modal) management strategy is to simply let it run its course, with workers completing some or all of the available assignments before the HIT expires. But, it is also helpful to be able to make certain changes to HITs after they have been created

To delay the expiration of HIT (e.g., because not all assignments have been completed), `ExtendHIT(hit=hits$HITId[1], add.seconds=seconds(days=1))` extends the specified HIT(s) by the time specified in `seconds()`. If intercoder reliability appears to be low, a call to `ExtendHIT(add.assignments=1)` increases the number of available assignments for all specified HITs by one (or more) to help resolve disagreement.[11]

```
ExtendHIT(hit.type=register, add.assignments=1)
```

To instead expire a HIT early (e.g., because there is an unanticipated problem with the HIT), simply call `ExpireHIT()` with one or more HITIds specified. At the completion of data collection,

[11] MTurk also provides "review policy" functionality to automatically respond to agreements or disagreements between workers' coding. See MTurk documentation.

the easiest method of reviewing workers' assignments is simply to retrieve all of the assignments and them approve them. `ApproveAllAssignments()` can be specified with either a HITId or a HITTypeId. Once a HIT and all of its assignment data are no longer needed, `DisposeHIT()` can optionally delete all data from the MTurk server.

```
a <- hits$HITId[1]
b <- GetAssignments(hit=a, return.all=TRUE)
c <- ApproveAllAssignments(hit=a)
```

3.2 Second Use Case: Panel Experiments

The viability of MTurk as a platform for implementing social science experiments is well-understood, but the platform's comparative advantage for implementing complicated panel data collection is underappreciated, in part for technological reasons. The ability to recontact and pay large numbers of workers falls outside the functionality of the RUI. In this use case, the objective is to implement a three-wave panel experiment, where respondents are randomly assigned to a condition at $t1$, recontacted to participate in a follow-up wave with additional block-randomization at $t2$, and finally a second follow-up wave at $t3$. In contrast with the first use case, the first stages of the workflow here are relatively easy. First, an experimental survey instrument is constructed with any tool (e.g., Qualtrics or one's own HTML). Second, a single HIT is created (without a qualification test) to which workers respond by completing the survey-experiment. Note that when only one HIT is needed, the parameters normally assigned with `RegisterHITType()` can be specified in the `CreateHIT()` command.

```
newhit <- CreateHIT(question=GenerateExternalQuestion("http://www.test.com/surveylink", 400)$string,
                    title="20-Question Survey",
                    description="Take a five-question survey about your political opinions.",
                    reward=".25",
                    duration=seconds(hours=4),
                    expiration=seconds(days=7),
                    assignments=1000,
                    keywords="survey, question, answers, research, politics, opinion",
                    auto.approval.delay=seconds(days=15),
                    qual.req=GenerateQualificationRequirement("Location","==","US",preview=TRUE))
```

Once a sufficient number of responses are collected (i.e., assignments completed) — this can be checked with `HITStatus(hit=newhit$HITId)`, assignments can be reviewed such that only those who pass an attention check are approved and the remainder rejected:

```
review <- GetAssignments(hit=newhit$HITId)
correctcheck <- "7"
approve <- approve(assignments=review$AssignmentId[review$check==correct])
reject <- reject(assignments=review$AssignmentId[!review$check==correct])
```

After reviewing these assignments, MTurkR is leveraged via the three tasks at the right side of Figure 1: Bonus, Contact, and Qualify. To implement a panel, workers who completed the original HIT (the $t1$ survey) are randomized to receive either a Democratic or Republican message in the $t2$ survey, with separate links for each condition. These workers are then contacted to complete the $t2$ survey and are paid a bonus if they complete it.

```
random <- rbinom(dim(approve)[1], 1, .5)
b1 <- "If you complete a follow-up survey, you will receive a $.50 bonus.\n
       You can complete the survey at the following link: http://www.test.com/link1?WorkerId="
w1 <- approve$WorkerId[random==1]
t2cond1 <- ContactWorkers(subjects="Complete follow-up survey for $.50 bonus",
                          msgs=sapply(c1,FUN=function(worker) paste(b1,worker)), workers=w1)
```

```
b2 <- "If you complete a follow-up survey, you will receive a $.50 bonus.\n
       You can complete the survey at the following link: http://www.test.com/link2?WorkerId="
w2 <- approve$WorkerId[random==2]
t2cond2 <- ContactWorkers(subjects="Complete follow-up survey for $.50 bonus",
                          msgs=sapply(c2,FUN=function(worker) paste(b2,worker)), workers=w2)
```

The process could be repeated for the $t3$ survey. Paying bonuses for completing the $t2$ and $t3$ surveys is similar. The below example shows paying a single bonus, but replacing the single character strings with vectors of character strings allows multiple bonuses to be paid with one called to `GrantBonus()`.

```
bonus <- GrantBonus(workers="AZ3456EXAMPLE",
                    assignments="123RVWYBAZW00EXAMPLE456RVWYBAZW00EXAMPLE",
                    amounts="1.00", reasons="Thanks for completing the follow-up survey!")
```

Workers can optionally be granted qualifications, e.g., to allow those who respond to all panel waves to complete a future study or, alternatively, to prevent participants in this study from completing a similar study launched in the near future bringing the process back to the left side of Figure 1. With all data collected, analysis can proceed.[12]

4 APIs, Cloud Data, and Political Science

As these two use cases have demonstrated, crowdsourcing (via MTurk) is a powerful addition to the political scientist's toolkit. By managing cloud-based data collection directly in R, this process is also relatively easy, done in a comfortable programming environment, and scientifically reproducible. But using MTurkR to connect with the MTurk API also shows that R can be an effective and easy-to-use way to work with cloud-based data. Currently, R has few packages that provide easy access to data APIs. One, twitteR provides access to Twitter data. Alone it may not be immediately useful, but when used in combination with the U.S. federal government's Social Media Registry API, it should be relatively easy to track the Twitter activity of numerous government agencies. A second R package, RWeather provides access to current weather from NOAA weather stations; a more useful package would tap the wunderground.com API, which is capable of providing robust, geocoded weather data going back decades. An enormous amount of data is also available from other APIs, such as those from Bitly to track social sharing of links and YouTube Analytics API to track liking, sharing, and commenting on YouTube videos, The World Bank, U.S. Census Bureau, the Federal Register, OpenCongress and GovTrack, several Washington Post APIs that track political contributions and White House visitors, and eventually the new Congress.gov, which aims to be a central repository for Congressional data available via APIs. Developing software that enables researchers to readily tap into these immense sources of data is an important task for political methodologists, who can develop tools that enable non-programmers to easily access these data.

References

Amazon.com. 2012. "Amazon Mechanical Turk Getting Started Guide.".

[12] If all three panel waves were created as HITs, qualifications would restrict the $t2$ and $t3$ surveys to workers who had completed $t1$ and the data could also be directly extracted from MTurk via MTurkR rather than having to be pooled from another survey tool.

Berinsky, Adam J., Gregory A. Huber, and Gabriel S. Lenz. 2010. "Using Mechanical Turk as a Subject Recruitment Tool for Experimental Research." Unpublished paper, Massachusetts Institution of Technology.

Buhrmester, Michael, Tracy Kwang, and Samuel D. Gosling. 2011. "Amazon's Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data?" *Perspectives on Psychological Science* 6(1): 3–5.

Chen, Jenny J., Natala J. Menezes, and Adam D. Bradley. 2011. "Opportunities for Crowdsourcing Research on Amazon Mechanical Turk." Unpublished paper, Amazon.com.

Iyengar, Shanto. 2010. "Experimental Designs for Political Communication Research: From Shopping Malls to the Internet." In *Sourcebook for Political Communication Research: Methods, Measures, and Analytical Techniques*, eds. Erik Page Bucy, and R. Lance Holbert. New York: Routledge.

Iyengar, Shanto, and Lynn Vavreck. 2011. "Online Panels and the Future of Political Communication Research." In *Handbook of Political Communication Research*, eds. Holli A. Semetko, and Margaret Scammell. New York: Sage Publications , 225–240.

Kittur, Aniket, Ed H. Chi, and Bongwon Suh. 2008. "Crowdsourcing User Studies with Mechanical Turk." Paper presented at the CHI 2008, New York, New York, USA.

Lang, Dustin Temple. 2012*a*. "RCurl: General network (HTTP/FTP/...) client interface for R." R package version 1.91-1.1.

Lang, Dustin Temple. 2012*b*. "XML: Tools for parsing and generating XML within R and S-Plus." R package version 3.9-4.1.

Leeper, Thomas J. 2012. "MTurkR: Access to Amazon Mechanical Turk Requester API." R package version 0.1.

Mason, Winter, and Siddharth Suri. 2011. "Conducting Behavioral Research on Amazon's Mechanical Turk." Unpublished paper, Yahoo! Research. `http://www.ncbi.nlm.nih.gov/pubmed/21717266`

Paolacci, Gabriele, Jesse Chandler, and Leonard N. Stern. 2010. "Running Experiments on Amazon Mechanical Turk." *Judgment and Decision Making* 5(5): 411–419.

Schmidt, Lauren A. 2010. "Crowdsourcing for Human Subjects Research." Paper presented at the CrowdConf 2010, San Francisco, CA.

Vavreck, Lynn, and Shanto Iyengar. 2011. "The Future of Political Communication Research: Online Panels and Experimentation." In *Oxford Handbook of Public Opinion and Media Research*, eds. Robert Y. Shapiro, and Lawrence R. Jacobs. New York: Oxford University Press.