Amazon Mechanical Turk basics: a tutorial

Anna Di Natale

February 2021

1 Introduction

Amazon Mechanical Turk (MTurk) is a "crowdsourcing marketplace that makes it easier for individuals and businesses to outsource their processes and jobs to a distributed workforce who can perform these tasks virtually"[1]. This platform is usually used for surveys, including market surveys, and labelling jobs. MTurk offers also the possibility of using the manually labelled data to pretrain algorithms to label a larger amount of data of the same type. MTurk is not the only crowdsourcing platform for this type of tasks, but a review of all of them is beyond the scope of this tutorial. I personally chose MTurk to collect data useful for my analysis because it seems to give some kind of continuity. Other companies, as for example Crowdflower, are more likely to be bought and change their policies. As a consequence, everything you already learnt on the platform has to be re-learnt and updated. On the opposite, until now MTurk has undergone very little changes, allowing my survey to still work on the platform one year after I set it up for the first time.

The aim of this tutorial is to help you set up your first survey on MTurk. This introduction is based on my knowledge and research on the topic. I will also include some tips that I think are useful if you want to start using this platform. This tutorial is not intended to substitute any official tutorial of the Amazon Mechanical Turk platform. Every time you are in doubt, you should consult the official API documentation, that will be shared together with this handout.

MTurk and in general crowdsourcing platforms are widely used by companies and their use is increasing also among researchers. The main drivers for this choice are that these platforms can be cheaper than other traditional ways of data collection, they allow to reach a wider audience, of which you can select particular features you are interested in. Moreover, it can be flexible in the layouts and can be very fast, especially for experienced requesters. Unfortunately, it can be very hard to learn, especially when using the API. This is why I decided to collect all my knowledge in the topic and write this handout.

As a starting note, I would like to draw your attention to the topic of research ethics. You are going to work with real human beings, even if it might seem not the case. You should consider if your study requires ethical approval, before

[1] https://www.mturk.com/

putting it online.

1.1 How MTurk works

MTurk collects workers who complete tasks uploaded by requesters into a space called Marketplace (see Figure 1). When workers successfully complete tasks they get a monetary reward, which is set by the requester in advance. The reward is taken into account by the worker when choosing which task to take.

Figure 1: Principle on which MTurk works.

The first step for setting up MTurk is to register to the MTurk website (`http://www.mturk.com`) as requester. You can also register as worker, which I find very useful when testing your own surveys. Moreover, with a worker account you can preview surveys from other requesters, which, in my opinion, give you a nice insight into the kind of job workers are asked to perform, the pays they earn and so on. Therefore, I advice you to get both a requester and a worker account. Unfortunately, my worker account is still waiting for approval (which I guess will never come).

1.2 Terminology

Before starting, I would like to define some terms that are often used on MTurk. As I already mentioned, a **worker** is a person that has an MTurk worker account, while a **requester** is a person that has a requester account. Workers complete tasks uploaded by the requesters and as a reward they earn money paid by the requesters. Workers have **qualifications**, which are characteristics of the worker that a requester could be looking for. Some examples of qualifications are the type of job of the worker, which social media the worker uses, and how active he/she is on the MTurk platform (see subsection 1.4 for a more detailed description). Tasks uploaded by the requesters are called **HITs**, Human Intelligence Tasks, which will be referred to as HITs or tasks. Requesters can seek for multiple workers to complete the same HIT, e.g. to assess the consensus on some answers. These are called **assignments**. Assignments are basically copies of the same HIT, each of which can be completed by different

workers, also simultaneously. See Figure 2 for a graphical representation.

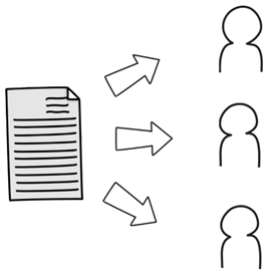


Figure 2: The same HIT can be completed by different workers. Each copy of the HIT, which is assigned to a different worker, is called assignment. In this representation there is 1 HIT and 3 assignments.

1.3 Fees

As I already mentioned, workers complete tasks and earn money. The requester decides how much to pay workers per accepted assignment. Note that you will pay only the assignment that you accept, e.g. the ones that satisfy your quality requirements. It is also possible to give bonuses to workers in order to promote them. On top of the reward you pay to workers, MTurk earns 20% of that amount as fee if your HIT has less than 10 assignments. If your HIT has 10 or more assignments, you will pay an additional 20%. Therefore my suggestion is to always run at most 9 assignments. If you need more than 9 workers, run the same HIT multiple times.
Moreover, the requester can select the workers on the basis of their qualifications. Some of these, called premium qualifications, are paid in addition to the price of the assignment. We will address the topic of qualifications in the next section.

1.4 Qualifications

Workers can be selected on the basis of their qualifications. Some qualifications are free of charge, other qualifications, called premium qualification, are not. Free qualifications are the location of the workers, the percentage of accepted assignments (i.e. the quality of their work) and the number of completed assignments (i.e. the experience of the worker). Premium qualities are numerous and have a different pricing. They can be, for example, the kind of job of the worker (e.g. administrative, management, education and training), their household income, the fluency of the language they speak (available at the moment only for a few languages), whether they voted in the US presidential elections of 2016 and many more. For the full list of premium qualifications and their pricing visit `https://requester.mturk.com/pricing`. Qualification of workers are paid per accepted assignment.

When setting the free qualifications and if you are interested in obtaining data of a good quality, it is important to set a requirement for both quality and experience of the workers. In fact, if workers have completed less than 100 HITs, the quality of their work is automatically set to 100%. Therefore, if you set a quality requirement it is also important to put a minimum requirement on the experience of the worker (at least 100 HITs completed).
The requester can also define some qualifications to give to workers that have taken their HITs. For example, you can give a qualification to all the workers that completed your HITs with a satisfying quality. This way, in the future you can allow only these people to participate to your studies. In the case the worker does not meet a qualification set up by the requester, he/she can request it and the requester can decide whether to grant it or not. For example, for your HIT, you would like only workers that live in the US and have already participated to a previous study of yours. If a worker does not live in the USA, it will be impossible for him/her to start your HIT. On the other hand, if the worker lives in the USA but does not have the second qualification, i.e. the worker has not completed one of your previous HITs, he/she can ask you to grant it. It is up to the worker to decide whether to give the qualification to the worker that requested it or not.
Let's now begin with the different ways you can set up your HIT in MTurk. I divided this topic in two parts: the lazy survey, which is the easiest to set up and the API, which requires more time but is also more flexible to your needs.

2 Lazy survey

I called this approach "lazy" because it requires little time to set up, therefore it is the best approach to get started and explore the potentialities of MTurk. Unfortunately, this approach has some limitations that I will discuss later. The lazy approach consists in the creation of a HIT using the MTurk interface.
After logging in as requester on the MTurk website, go to Create > New Project. This interface shows the templates that MTurk offers for the HITs. You will be able to choose the one that fits your survey best (Figure 3 shows one example). The next step consists in the choice of all the specifications of your study. Figure 4 shows the page where you will have to enter them. Title, description of the task and keywords will help the worker to decide whether they are interested in the kind of questions of your HIT. Additionally, you will decide how much to pay the workers per assignment, how many assignment to request, how much time the task should take, how much time your task will be available in the marketplace and finally when to auto-approve workers. This last specification states how much time you want to have in order to reject assignment. If you do not reject an assignment within the time frame declared, the assignment will be automatically approved and the worker will be paid. The maximum allowed is 30 days. The last specifications you can set regard the workers. In this form you can enter all the qualifications you ask from workers. You can also ask for Masters, i.e. workers that have received the Masters qualification from MTurk.

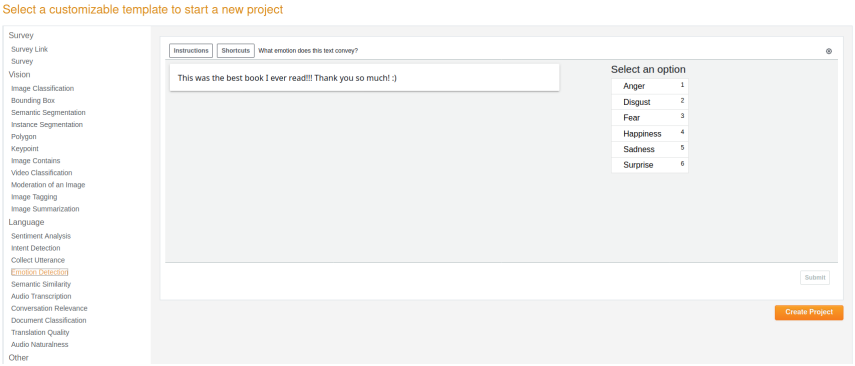


Figure 3: Layout options for the lazy survey. On the left, you can see all the options that are offered. Your survey has to fit the layout very well, because the allowed modifications to the original layout are very minimal. Moreover, all the pages of your HIT will look like the one of the template, which forbids, for example, the presence of a feedback form at the end of your HIT.

This qualification is granted to workers who deliver consistently good results on a wide variety of tasks. If you decide to hire only masters workers, you will have to pay an additional fee. If your HIT contains adult content you have to specify it here. This will warn the workers that will be able to decide whether to take it or not. Finally, you can choose if the HIT can be visible and previewable by any worker (public), visible by all the workers but previewable only by workers that match the required qualifications (private) or visible and previewable only by workers with the qualification (hidden). I honestly do not know which choice is best.

Figure 4: HIT specifications that you need to set on MTurk.

Subsequently, you will be able to work on the layout of the HIT (for example, selecting the number of choices the workers have to choose from) and to upload the data that has to be labelled by the workers. At this point, the HIT can be published in the marketplace.
I called this approach 'lazy' because it is a fast way to collect data for your study, as you will not have to spend time learning the MTurk API. Unfortunately, the available layouts have little margin of modification, thus are recommended only if your study fits one of them. Additionally, your HIT has to display only one kind of question. This means that your HIT cannot consist of some multiple choice questions and a feedback form at the end, unless framing the feedback into a multiple choice question, evaluated on the same scale as the task itself. For all these reasons, I decided to learn the MTurk API.

2.1 Semi-lazy survey

Before addressing the MTurk API in detail, I would like to present what I call a **'semi-lazy'** approach. This approach corresponds to the 'survey link' option when choosing the HIT layout. In this case, you can design your HIT on a different platform, making sure that every worker that completes it gets a unique code. The HIT will be hosted on an external platform of your choice and enter

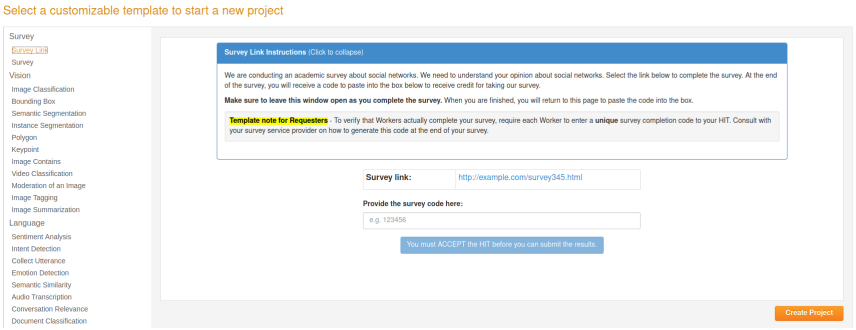


Figure 5: Semi-lazy approach. Workers that completed your task will enter the code displayed at the end of your HIT. The codes have to be always different.

the completion code on the MTurk platform. This allows you to design your HIT in a more flexible way, either relying on a different platform or programming it on your own from scratch. I would not suggest to use a second fee-charging platform on top of MTurk in order to design the HIT. On the contrary, you can design your own HIT and program a code generator for the workers that complete the HIT. I honestly have never tried it. In fact, since you have to design your own HIT and put it online with your own resources, I find it more rewarding to use the API. The API has the same requirements but allows for a more flexible experience.

3 Survey like a pro - the API

In this section I gathered what I learnt about the MTurk API. In my experience, the learning process of the API was demanding but using it has many advantages.

3.1 Design of the HIT

One important prerequisite for running a HIT with the MTurk API is to have a working survey on a website, i.e. an interactive website where your survey can be visualized and completed by the workers. In this case, you can design the survey as you like. However, the website page you are planning has to be connected to MTurk in order to submit the results. Here I share the code I use on my survey. In my case, I use a JavaScript file, whose chunks of code follow. The main lines of the JavaScript function are reported below. In this chunk,

```
/* MAIN */
$(document).ready(function() {
    $.getJSON("config.json").done(function(data) {
        config = data;
        if (config.meta.aggregate) {
            state.taskOutputs = {};
        }
        custom.loadTasks(config.meta.numSubtasks).done(function(taskInputs) {
        $.getJSON("questions.json").done(function(data) { questions=data;
            state.taskInputs = questions.quest;
            populateMetadata(config);
            setupButtons(config);
        });
      });
    });
});
```

the file 'config.json', where all the specifications of the HIT are saved, is loaded. I also load the questions for the survey from the file called 'questions'.
While designing the survey, I set up the final button (the one that submits the results to MTurk) so that when it is clicked it activates the function 'submitHIT'. In the next chunk of code you can see the function that sets up all the interactive buttons of the survey. The submit button is the one that activates the function to submit the results.

```
function setupButtons() {
    $("#next-button").click(nextTask);
    $("#consent-button").click(toggleInstructions);
    $(".instruction-button").click(toggleInstructions);
    $("#submit-button").click(submitHIT);
    if (state.assignmentId == "ASSIGNMENT_ID_NOT_AVAILABLE") {
        $("#submit-button").remove();
    }
}
```

When the submit button is clicked, the function 'submitHIT' is called. Such function is shown in the next code chunk. Here, I first recall from config.json whether the survey has to be uploaded to the MTurk Marketplace or to the Sandbox. Depending on the destination, I load the right url. Then I save the last answers collected in the last page of the survey, as the gender of the worker.

I also call the function saveTaskData that saves the data in a variable called state. ClearMessage is useful to clear the message field. I also change the class of

```
function submitHIT() {
    var submitUrl = config.hitCreation.production ? MTURK_SUBMIT : SANDBOX_SUBMIT;
    state.gender.push($("#gender").val());
    saveTaskData();
    clearMessage();
    $("#submit-button").addClass("loading");
    var form = $("#submit-form");
    for (var i = 0; i < config.meta.numSubtasks; i++) {
        var err = custom.validateTask(getTaskInputs(i), i, getTaskOutputs(i));
        if (err) {
            $("#submit-button").removeClass("loading");
            generateMessage("negative", err);
            return;
        }
    }

    addHiddenField(form, 'assignmentId', state.assignmentId);
    addHiddenField(form, 'workerId', state.workerId);
    var results = {
        'inputs': state.taskInputs,
        'outputs': state.taskOutputs
    };
    addHiddenField(form, 'results', JSON.stringify(results));
    var times = {
        'time':state.timeOutputs,
        'actions':state.action
    };
    addHiddenField(form, 'times', JSON.stringify(times));
    addHiddenField(form, 'feedback', $("#feedback-input").val());
    addHiddenField(form, 'gender', state.gender);

    $("#submit-form").attr("action", submitUrl);
    $("#submit-form").attr("method", "POST");
    $("#submit-form").submit();
    $("#submit-button").removeClass("loading");
    generateMessage("positive", "Thanks! Your task was submitted successfully.");
    $("#submit-button").addClass("disabled");
}
```

the submit button, changing its appearance. Subsequently I define the variable form. In this variable I save all the data I want as output of the survey. I also check that the inputs comply with the format they should have (e.g. I do not allow workers to leave the answers blank). The addHiddenField function is used to save the data in form. Finally, I connect to MTurk through the url defined at the beginning and send the output. The last lines communicate to the worker that the submission was successful.
After having designed a working survey on your website and having connected it to MTurk, we can use the API.

3.2 Set up

To set up the API, you need to follow the steps listed in the page Developer > Getting Started. In particular, you will have to activate an Amazon Web Services (AWS) account, link it with your MTurk requester account, register for the MTurk Developer Sandbox and download an AWS Software Development Kit (SDK). Another prerequisite to use the MTurk API is to design your own HIT and put it online on a website. In case you do not want to pay an external service to host your own website, you can use the GitHub website, as I did for some time before buying my own domain. If you are curious, you can see my

own HIT here: `https://annadinatale.eu/index_survey.html`.

3.2.1 Sandbox

The Sandbox is a very useful tool that MTurk makes available to developers. It is an interface that allows you to visualize your HIT exactly in the same way the workers will see it. This way you can run your own mock tasks to test the functionalities you implemented. Figure 6 shows the Sandbox, which is the exact copy of the MTurk interface for workers. Only people with a MTurk requester account can access the Sandbox. People (usually yourself or members of your own team) that complete your HIT on the Sandbox will not be payed. This platform is just an help to develop your HIT. The Sandbox interface gives us

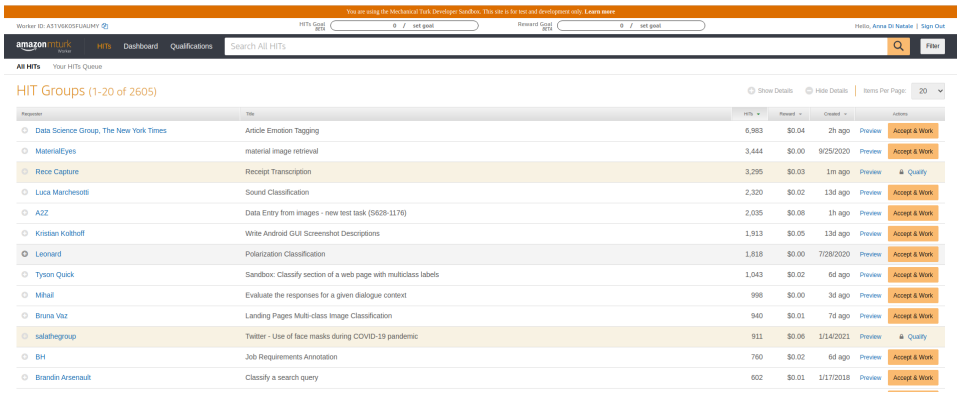


Figure 6: Sandbox interface. Only mock tasks are displayed here. On sandbox you can experiment with the functionalities of your own HIT.

also an idea of how the worker platform looks like. As we can see in Figure 6, workers see the name of the requester (the person that uploaded the HIT), the name of the HIT, the number of HITs the requester has uploaded, the reward for the completed task, when they HIT was created and whether the worker can start working right away or if he/she has to undergo a quality check first.

3.3 The API

The last step to set up the MTurk API is to download an AWS Software Development Kit. In my case, I have always worked with the Python package, **boto**. In this section I will show and explain the lines of code that allow me to perform my HITs on MTurk through the API. Together with this handout, you should also read the API documentation for a better understanding of all the functions.
As first step, I import the client from the boto package. I also import packages to handle the data collected with the HIT.

In [ ]:
```
from boto3 import client
import json
import xmltodict
import pickle
```

Subsequently, I open config.json, a file where I save all the specification for the HIT, e.g. how much to pay each assignment, the qualifications of workers and so on.

In [12]:
```
data=open('../config.json', 'r').read()
config = json.loads(data)['hitCreation']
```

In the next chunk of code, I select whether the HIT will be posted on the MTurk market or on the Sandbox, i.e. whether I want to perform a mock task or a real one. This information is stored in config.json. As you can see, the difference is just in the url. I also print two variables to check the settings. In the case of this example, since production value is False, the HIT will be uploaded on Sandbox. I also print the number of assignment of this HIT to make sure that they are always less than 10.

In [13]:
```
if config['production']:
    endpoint_url = 'https://mturk-requester.us-east-1.amazonaws.com'
else:
    endpoint_url = 'https://mturk-requester-sandbox.us-east-1.amazonaws.com'
print(config['production'])
print(config['numAssignments'])
#cl = client('mturk', region_name='us-east-1', endpoint_url=endpoint_url)
```

```
False
9
```

Next I call the client function with the specifications of my requester account (access key, secret key and region). This allows me to access my MTurk requester account.

In [14]:
```
cl = client('mturk',
    aws_access_key_id = [redacted],
    aws_secret_access_key = [redacted],
    region_name='us-east-1',
    endpoint_url = endpoint_url
    )
```

Now that the specifications for the HIT are defined and I am connected to my worker account, I define a function to create the HIT. In this function, I first set up the qualifications of the workers. As you can see, I define 3 of them. Every qualification has an Id that can be found in the documentation for the API. With the first qualification I ask for workers that live in the USA. The second is a restriction on the quality of the work: I want only workers whose work has been accepted 97% of the times. With the third qualification, I put a threshold on the experience of my workers: they have to have completed at least 100 HITs. Next I define a 'questionText', which is the concatenation of the MTurk

url (either the real MTurk Marketplace or the Sandbox) and the url where my HIT has been uploaded to (my website). Eventually I call the function to create the HIT from the boto package with all the pre-set arguments.

In [ ]:

```
def create_hit():
    quals = [
        {
            'QualificationTypeId': '00000000000000000071',
            'Comparator': 'EqualTo',
            'LocaleValues': [{
                'Country': 'US',
            }],
        },
        {
            'QualificationTypeId': '00000000000000000040L0',
            'Comparator': 'GreaterThanOrEqualTo',
            'IntegerValues': [
                97
                ],
        },
        {
            'QualificationTypeId': '00000000000000000040',
            'Comparator': 'GreaterThanOrEqualTo',
            'IntegerValues': [
                100
                ],
        },
    ]
    questionText = "<ExternalQuestion xmlns=\"http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/"
    questionText += "2006-07-14/ExternalQuestion.xsd\">\n<ExternalURL>" + config['taskUrl']
    questionText += "</ExternalURL>\n  <FrameHeight>700</FrameHeight>\n</ExternalQuestion>"
    new_hit=cl.create_hit(
        MaxAssignments=config['numAssignments'],
        AutoApprovalDelayInSeconds=604800,
        LifetimeInSeconds=config['lifetime'],
        AssignmentDurationInSeconds=config['duration'],
        Reward=config['rewardAmount'],
        Title=config['title'],
        Keywords=config['keywords'],
        Description=config['description'],
        Question=questionText,
        QualificationRequirements=quals,
        )
    hit_id=new_hit['HIT']['HITId']
    print(hit_id)
    return(new_hit)

```

Finally, I call the previously defined function and print its url. In the case in

Creation of the HIT

In [12]:

```
url=create_hit()
print("A new HIT has been created. You can preview it here:")
if config['production']:
    print("https://worker.mturk.com/mturk/preview?groupId=" + url['HIT']['HITGroupId'])
else:
    print("https://workersandbox.mturk.com/mturk/preview?groupId=" + url['HIT']['HITGroupId'])

print ("HITID = " + url['HIT']['HITId'] + " (Use to Get Results)")
```

```
3MJ28H2Y1E8FWIXIOXCCIUCKS7AO5O
A new HIT has been created. You can preview it here:
https://workersandbox.mturk.com/mturk/preview?groupId=3GSJT927Q5F13YURZSQ4TS0CIV9MXD
HITID = 3MJ28H2Y1E8FWIXIOXCCIUCKS7AO5O (Use to Get Results)
```

which the HIT has been uploaded to the MTurk Marketplace, you will be able to see it only if you have an MTurk worker account. If you set the HIT to be uploaded to Sandbox, the url will bring you there. In Sandbox you can verify that your HIT works as you expected. In the code I also print the HITID, which is necessary to collect the results of the HIT.

Once run this last chunk of code and published the task to the MTurk Marketplace, I just have to wait for workers to complete the task. The waiting phase can take relatively long. In fact, even though most of the workers will complete the HIT in less time than you expected, for some of them it will take longer. If some of the workers that started your HIT get distracted, the time you allocated for the worker to complete the HIT may run out. In this case, the HIT is put back to the Marketplace, and chosen and completed by another worker. Thus, even if the HIT stays in the Marketplace only for a few minutes, it could take longer than you expected to collect the results.

In order to check the status of the HIT and see how many workers completed

it, I run the following script:

Lists the last HITs requested

```
In [14]: cl.list_hits(MaxResults=1)
```


```
Out[14]: {'NextToken': 'p1:h9f+VdiIWfMKrg6lGvUrJTeJkWlq0Lp2V26t2HMjtgHQjU37O9sXlnZUUfYR',
 'NumResults': 1,
 'HITs': [{'HITId': '3MJ28H2Y1E8FWIXIOXCCIUCKS7AO5O',
   'HITTypeId': '37UC2S9IBM1BM6KO592GA6G5EOCM53',
   'HITGroupId': '3GSJT927Q5F13YURZSQ4TS0CIV9MXD',
   'CreationTime': datetime.datetime(2021, 2, 8, 13, 29, 59, tzinfo=tzlocal()),
   'Title': 'Meaning similarity',
   'Description': 'How similar are the meanings of these two words?',
   'Question': '<ExternalQuestion xmlns="http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2006-07-1
4/ExternalQuestion.xsd">\n<ExternalURL>https://annadinatale.github.io/index_survey.html</ExternalURL>\n  <FrameHeigh
t>700</FrameHeight>\n</ExternalQuestion>',
   'Keywords': 'meaning, words, labeling, semantic similarity',
   'HITStatus': 'Assignable',
   'MaxAssignments': 9,
   'Reward': '1.00',
   'AutoApprovalDelayInSeconds': 604800,
   'Expiration': datetime.datetime(2021, 2, 9, 13, 29, 59, tzinfo=tzlocal()),
   'AssignmentDurationInSeconds': 1200,
   'QualificationRequirements': [],
   'HITReviewStatus': 'NotReviewed',
   'NumberOfAssignmentsPending': 0,
   'NumberOfAssignmentsAvailable': 8,
   'NumberOfAssignmentsCompleted': 0}],
 'ResponseMetadata': {'RequestId': 'afbb31ac-e454-4f1f-8797-7bb722808266',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'afbb31ac-e454-4f1f-8797-7bb722808266',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '1008',
   'date': 'Mon, 08 Feb 2021 12:47:03 GMT'},
  'RetryAttempts': 0}}
```


With the function `list_hits`, it is possible to visualize the status of the last n HITs (where n=MaxResults) you uploaded. The function prints all the details relative to the selected HITs. If 'NumberOfAssignmentsAvailable' is 0, all the assignments have been taken by a worker. The number of workers that are working on an assignment is visualized on 'NumberOfAssignmentsPending'. When the workers complete the task, they are counted in 'NumberOfAssignmentsCompleted'. Therefore, when the 'NumberOfAssignmentsCompleted' equals to the number of assignments of your HIT you can retrieve the full results.

In order to retrieve the results, I run the following code chunk. As you can see,

```
In [16]: worker_results = cl.list_assignments_for_hit(HITId=url['HIT']['HITId'], AssignmentStatuses=['Submitted'])
         print(worker_results)
```


```
{'NextToken': 'p1:y3VLNHEmyCXgsjG/vhq64y8xF2H5KMyrzcE1PHgbsPSGgTSfsGJ2H2MKogMjhg==', 'NumResults': 1, 'Assignments':
[{'AssignmentId': '3JW0YLFXRTGJ1UJX1MVXE8XZEAJWW0', 'WorkerId': 'A31V6K05FUAUMY', 'HITId': '3MJ28H2Y1E8FWIXIOXCCIUCK
S7AO5O', 'AssignmentStatus': 'Submitted', 'AutoApprovalTime': datetime.datetime(2021, 2, 15, 13, 32, 26, tzinfo=tzlo
cal()), 'AcceptTime': datetime.datetime(2021, 2, 8, 13, 30, 58, tzinfo=tzlocal()), 'SubmitTime': datetime.datetime(2
021, 2, 8, 13, 32, 26, tzinfo=tzlocal()), 'Answer': '<?xml version="1.0" encoding="ASCII"?><QuestionFormAnswers xmln
s="http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2005-10-01/QuestionFormAnswers.xsd"><Answer><Que
stionIdentifier>workerId</QuestionIdentifier><FreeText>A31V6K05FUAUMY</FreeText></Answer><Answer><QuestionIdentifier
>results</QuestionIdentifier><FreeText>{"inputs":["mountain egg","drink beverage","belly intestines","long eye","bec
ome do","climb field","crouch sit","carry bring","beggar orphan","say sand","food full","flow retreat","louse mothe
r","fall house","certain yes","acquit rescue","aircraft plane","country land","plate plain","cut wood","love chi
n","head roof","plate dish","border pill","new fresh","full father","bowl soon","town splash","give say","press happ
iness","test test "],"outputs":["4","5","6","4","2","1","2","dunno","3","4","6","5","1","2","3","4","5","6","dunn
o","dunno","6","5","4","3","2","1","2","3","4","5",""]}</FreeText></Answer><Answer><QuestionIdentifier>times</Questi
onIdentifier><FreeText>{"time":[1612787461992,1612787463816,1612787484257,1612787485721,1612787487270,1612787488718,
1612787490276,1612787491992,1612787493464,1612787496095,1612787499423,1612787501218,1612787503147,1612787505098,1612
787507702,1612787509632,1612787511316,1612787513206,1612787514834,1612787516756,1612787518604,1612787520454,16127875
22462,1612787525098,1612787526964,1612787528933,1612787530954,1612787532764,1612787534740,1612787537032,161278753956
6,1612787541267,1612787545691],"actions":["agree","continue","next","next","next","next","next","next","next","nex
t","next","next","next","next","next","next","next","next","next","next","next","next","next","next","next","nex
t","next","next","next","next","next","next","submit"]}</FreeText></Answer><Answer><QuestionIdentifier>feedback</Que
stionIdentifier><FreeText/></Answer><Answer><QuestionIdentifier>gender</QuestionIdentifier><FreeText>f</FreeText></A
nswer></QuestionFormAnswers>'}], 'ResponseMetadata': {'RequestId': '2e37aea1-ed61-45ea-bef6-3285cf48978d', 'HTTPStat
usCode': 200, 'HTTPHeaders': {'x-amzn-requestid': '2e37aea1-ed61-45ea-bef6-3285cf48978d', 'content-type': 'applicati
on/x-amz-json-1.1', 'content-length': '2500', 'date': 'Mon, 08 Feb 2021 12:49:55 GMT'}, 'RetryAttempts': 0}}
```


in order to retrieve the results of the HIT you have to enter its HITID. The format and data that I retrieve has been programmed in the survey design.

3.4 Quality of the collected data

Once you retrieved the results, you can start analysing it. You will soon realise that some workers were distracted while completing your task and you will not want to use their answers. In order to spot distracted workers or workers that did not understand your instruction, I suggest putting some control questions in your HIT. In my case, I put some of the examples from the instructions to complete the HIT. In theory, if the workers read the instructions and/or understood what they are expected to do, it will be very easy for them to answer to those control questions. When you find a worker that did a good job and you want to accept their work, you can run the following line of code:

```
In [ ]: cl.approve_assignment(AssignmentId='3NGMS9VZTLIFFMB7L8CQAM01065FFR',RequesterFeedback="Thank you!")
```

It is not necessary to accept a worker. In fact, when creating the HIT you have defined a maximum amount of time within which you will have to reject workers. If a worker has not been rejected when the time runs out, their work will be automatically accepted and the worker will be paid. Therefore, it is more important to know how to reject one assignment. Here is the function to use:

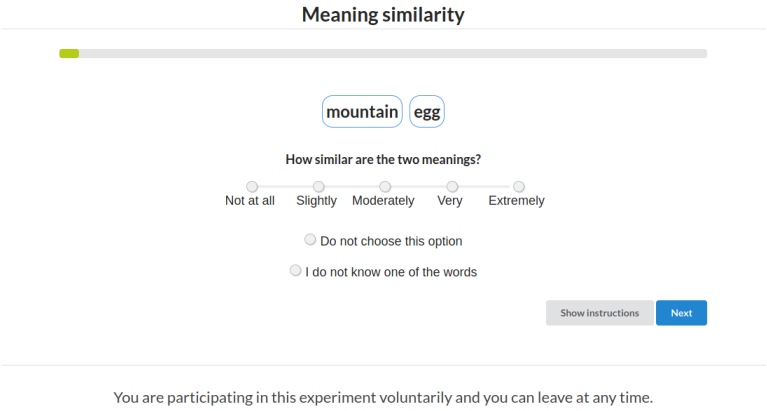
```
In [53]: cl.reject_assignment(AssignmentId='3DHE4R9OCWB1SSVQ8X318O6OYUPG2S',RequesterFeedback="You failed the control ques

Out[53]: {'ResponseMetadata': {'RequestId': '3c4ca299-fa64-47d0-9394-edc1c9f7540c',
           'HTTPStatusCode': 200,
           'HTTPHeaders': {'x-amzn-requestid': '3c4ca299-fa64-47d0-9394-edc1c9f7540c',
            'content-type': 'application/x-amz-json-1.1',
            'content-length': '2',
            'date': 'Fri, 13 Nov 2020 14:13:04 GMT'},
           'RetryAttempts': 0}}
```

When rejecting a worker, make sure to add a feedback. This will help the worker to understand what was his/her mistake. Remember that if you are too late and the time within which you could have rejected the worker is over, you will have to pay for the assignment, even if the quality of the job is not satisfying. Rejected assignments will not be paid for, nor you will pay for fees on those assignments. Therefore, it pays off to reject assignments. However, when you reject an assignment, the worker will not be paid either. Workers that feel they are being treated unfairly can appeal to your decision. This can result in numerous requests from workers to rethink your rejection. In order to accept an assignment previously rejected, you can run the following chunk of code, i.e. you have to add the variable OverrideRejection.

```
cl.approve_assignment(AssignmentId='xxx',RequesterFeedback="Thank you!",OverrideRejection=True)
```

From a few data I collected during by MTurk journey, I'd say that most of the requesters do not reject any worker. Usually the pay per assignment is quite low and it does not pay off the time spent answering angry emails of rejected workers. This results, of course, in distracted workers having fairly high rates of accepted assignments. In my case, in order to exclude unreliable workers, I had to higher my quality requirements, which now is 97%.

My survey displays a 'do not choose this option' button, with the purpose of detecting very distracted workers and bots (see Figure 3.4). During my



journey in MTurk and the history of this survey, uset by the quality of the data retrieved, I repeatedly changed the quality requirements of workers. Up to now, I require 97% acceptance rate and I have uploaded nearly 1,400 assignments to the Marketplace. However, so far 5.7% of the workers that completed my HITs clicked the 'do not choose this option'. In addition to this, more than 33.7% of the workers did not pass the control questions (whose answers are given in the instruction page as examples). As a consequence, more than 39% of the answers collected is not viable for analysis. Moreover, the more I higher the quality requirement for workers, the more workers decide to appeal when I reject their assignments.
In my opinion, in order to reduce the amount of noise we get from MTurk, everybody that runs HITs there has to reject workers whose work does not reach the minimum standards of quality.

4 References

I collected the information here reported from different sources. The pages created by MTurk to support developers were very useful: `https://docs.aws.amazon.com/AWSMechTurk/latest/AWSMturkAPI/Welcome.html`. I also found useful the GitHub page related to MTurk, and in particular: `https://github.com/aws-samples/mturk-code-samples`. In addition to the official documentation, I gathered bit and pieces of code and information from different sources. In particular, the layout of my own MTurk survey was inspired and obtained modifying codes from `https://github.com/kimberli/mturk-template`.

Contents