Automated Question Generation for Quality Control in Human Computation Tasks

Dominic Seyler¹, Mohamed Yahya¹, Klaus Berberich², Omar Alonso²
¹Max-Planck Institute for Informatics ²Microsoft
{dseyler,myahya,kberberi}@mpi-inf.mpg.de
omalonso@microsoft.com

ABSTRACT

When running large human computation tasks in the real-world, honeypots play an important role for assessing the overall quality of the work produced. The generation of such honeypots can be a significant burden on the task owner as they require specific characteristics in their design and implementation and continuous maintenance when operating data pipelines that include a human computation component. In this extended abstract we outline a novel approach for creating honeypots using automatically generated questions from a reference knowledge base with the ability to control such parameters as topic and difficulty.

CCS Concepts
Information systems → Crowdsourcing; Trust;

Keywords
Honeypot Generation; Natural Language Questions

1. INTRODUCTION

Labeling assignments such as relevance evaluation, entity resolution, information extraction, and image classification are very popular crowdsourcing tasks. The use of crowdsourcing as a platform for executing those tasks has been adopted due to its lower cost, processing speed, and overall good quality. However, managing quality control in any crowdsourcing task requires implementing techniques for checking workers’ performance to ensure good results. One of the most widely used techniques in practice, called “honeypots”, involves including a question with an already known answer as part of the task. If a worker does not pass a honeypot it can be an indication of low performance, that is, bad work [8, 1]. The technique has a number of advantages: 1) it looks like a regular assignment so the worker does not feel like she is dealing with a trap question, 2) it is easy to identify bad workers by just looking at the ones that did not answer the honeypots correctly. At the same time, there are a number of drawbacks: 1) the set of honeypots needs maintenance as workers can learn to identify them, 2) require the design of unambiguous questions and answers, 3) editorial construction has scalability issues, and 4) needs to support different levels of complexity. It would be useful if the the task owner has the ability to programmatically execute a honeypot generator as part of the data preparation step.

In this position paper, we propose to use an existing knowledge base to generate potential questions that can be used as honeypots in a crowdsourcing task. Our approach consists of a parametrized question generation technique that allows a task owner to vary the topic and difficulty of questions.

Previous work on automated question generation and difficulty assessment has focused on the learning scenario mostly in the context of generating and assessing reading comprehension and language skill [2, 6]. STEP, a system for generating test questions is described in [3]. Integration of information extraction and human computation for knowledge acquisition is presented in [7]. Micro-diversions, a technique for improving worker productivity and work quality is described in [4]. An overview of spam detection techniques is given in [5].

2. QUESTION CONSTRUCTION

We follow the method outlined in Seyler et al. [9], which enables us to generate quiz questions from a knowledge base (KB) without requiring human effort in the process. Given a topic, which is a set of entities (possibly a singleton) such as 18th Century Writers or Fortune 500 Companies, we can automatically generate questions whose answer is an entity in that topic. The question starts life as a structured triple-pattern query (SPARQL query) over the specific knowledge base. This allows us to ensure that a question has a unique answer, making the process of automatically checking answer correctness a trivial one with no room for disputes. Using a simple template-based technique, the query is verbalized resulting in a natural language question, which can be understood by a lay person. Furthermore, we introduce a question difficulty estimator. The estimate relies on signals such as popularity of the answer entity, coherence of the entities mentioned in the question and the answer entity, and selectivity of the relations in the question, among others. These signals are computed from both the knowledge base and large entity-annotated textual corpora (e.g., Wikipedia). Utilizing this difficulty estimate, we can craft questions with varying complexity.

We extended the above question generation scheme to generate multiple-choice questions. Such questions can further simplify the process of automated answer verification by al-
allowing users to select from several options as opposed to providing textual input. To create a multiple-choice question we add distractors, which are incorrect answer entities. Starting from the unique-answer query that serves as the basis for a question, we relax this query by incrementally removing triple patterns. The answer sets of these relaxed queries provide us with distractor candidates. Each distractor candidate is automatically assigned a score reflecting how similar it is to the correct answer in the context of the question. The top $n$ scoring distractors are used for forming the multiple-choice question.

3. HONEYPOT GENERATION

3.1 System Overview

As input, the task owner provides a data set that needs labels (e.g., documents, query logs, entities, etc.), the HIT (Human Intelligence Task) template, and a configuration file that includes a number of parameters (e.g., number of honeypots, difficulty level, topics, etc.). The honeypot generator takes those inputs and produces a data set that includes the original data plus the honeypots. The newly produced data set is then uploaded to a specific platform like Amazon Mechanical Turk (AMT) or CrowdFlower. Once the task is completed, another component (honeypot assessment) checks the performance of workers on the honeypots and updates internal information in the knowledge base. We can think of this last step as a feedback loop on how effective those honeypots were. Figure 1 graphically depicts this pipeline.

3.2 Framework

We now describe our framework for integrating automated question generation into the workflow of a crowdsourcing task. We subsequently give an example instantiation of this framework for a concrete crowdsourcing task.

For a labeling task $T$, we want to generate a question to serve as a honeypot for a certain task instance $t$. The goal of a question is to judge the quality of the labels provided by a worker of a task instance. What the questions directly captures is a worker’s knowledge of a specific domain relevant to the task instance at hand. A procedure called $QGen(t)$ takes a specific task instance, maps it to a topic, and produces a question $q$, that is added to $t$. $QGen(t)$ can additionally be parametrized by a difficulty level to support questions of varying difficulty.

After a task owner has run her task through a crowdsourcing framework extended with automatically generated honeypot questions, she gathers the results for both original task instances and honeypot questions. The task owner can subsequently decide how to handle the input from each worker based on the answers they provide for the honeypots. The evaluation of a worker’s answer to a honeypot is done in a completely automated manner without the need for intervention by the task owner. The task owner, based on her specific needs, can decide to disqualify those workers who provided incorrect answers to a specific number of honeypots. She can also decide to take question difficulty into consideration when considering the trustworthiness of a worker. We next present an example instantiation of this framework.

3.3 Example: Entity Resolution

In each instance of this task, a worker is presented with a document $d$ with spans annotated as named entity mentions. Additionally, each span is associated with a set of KB entities. A worker is supposed to choose the KB entity that the mention should map to. Figure 2 shows an example instance of the entity resolution task.

In this setting, $QGen(t)$ takes a document $d$ annotated with entity mentions and returns a set of questions whose answers are entities relevant to $d$. A simple way to determine relevance of an entity $e'$ to a document $d$ is to first represent $e'$ by some document called $doc(e')$ (e.g., e’s Wikipedia page) then find $e'$ with $doc(e')$ most similar to $d$. In the example of Figure 2, this procedure has determined that $doc(BarackObama)$ (which is coincidentally the entity to which the phrase “Obama” should be resolved) is very similar to the document given in the example. The figure also shows two multiple-choice questions generated about Barack Obama, an easy one and a difficult one. The correct answer is underlined for the reader, but in a crowdsourcing setting it needs to be provided by the worker.

4. CONCLUSIONS AND OUTLOOK

We presented a framework for integrating automated questions generation into a crowdsourcing workflow for the purpose of simplifying the process of honeypot generation to assess work quality. Our goal is to reduce the burden on the task owner. Given the widespread interest and use of crowdsourcing, we believe that automating honeypot generation is an important research direction.

---

**Figure 1: System overview**

**Figure 2: Example honeypot multiple-choice questions in an entity resolution setting.**
5. REFERENCES