The Practice of Crowdsourcing: Things to Know About Using Humans and Machines for Labeling

Omar Alonso

NTCIR-13, December 2017
Tokyo, Japan
Disclaimer

The views, opinions, positions, or strategies expressed in this talk are mine and do not necessarily reflect the official policy or position of Microsoft.
Outline
Introduction
Problems
Wetware programming
Quality control
Implementation considerations
Conclusion
Introduction
Human computation

Use humans as processors in a distributed system
Workers, raters, annotators, judges
Address problems that computers aren’t good
Human Intelligence Task (HIT)

Available platforms
Amazon Mechanical Turk
CrowdFlower

A sample of HITs
In case you didn’t know
You are a computer
Some context

We assume supervised or semi-supervised learning

Large scale

Continuous

Crowdsourcing != Mechanical Turk
Why we need labels?
Information retrieval
Natural language processing
Machine learning
Active learning
Artificial intelligence
A sample of common tasks

Content moderation
Information extraction
Search relevance
Entity resolution
What is a label?

Query = mount fuji
Task: Given the query, is the page relevant?
Answers: very, somewhat, not
Labels: 1, 0.5, 0
Careful with that axe data, Eugene

In the era of big data and machine learning
labels -> features -> predictive model -> optimization
Labeling perceived as boring
Tendency to rush labeling
Quality is key
Garbage in, garbage out
Lifecycle of a label

Information retrieval example

Using a crowd to label a data set

Using ML to process the complete data set
Three types of labeling tasks

Objective
Objective question has a correct answer

Partially objective
Judgment question has a best answer

Subjective
Subjective question has consistent answer
HC & crowdsourcing in the field
The state of the field

Human-labeled data is more important than ever

Requirements
Throughput -> ASAP; I need the labels for yesterday
Cost -> cheap; if possible free
Quality -> top

Performed as a one-off by 3rd party (crowd or editors)
Non trivial amount of work to get good results
Very limited functionality in current platforms
Problems

Monolithic HITs
The structure of a HIT mirrors the structure of the task the developer is working on
Similar to Conway’s law in software engineering

Task complexity

Lengthy instructions
RTFM doesn’t work

We don’t think of HC/crowdsourcing as programming

How to improve
Use established programming practices
Careful, we are dealing with humans and not machines
Wetware programming

“Machines have no common sense; they do exactly as they are told, no more and no less” - D. Knuth
“Errare humanum est” - Seneca
Generic approach
Well-known techniques for writing programs
Humans executing a task on a machine
A programming view for humans and machines
Humans executing code

Instruction set is somewhat unknown
Latency
Cost/incentives
Errors
Task difficulty
Human factors
Asking questions

Part art, part science

Instructions are key

Workers may not be experts so don’t assume the same understanding in terms of terminology

Show examples

Hire a writer

Engineer writes the specification

Writer communicates
HIT design

Self-contained, short, and simple
Document presentation & design
Engage with the worker
Need to grab attention
Localization
Reliability

What to look for
Agreement, reliability, validity

Inter-rater agreement
Agreement between judges
Agreement between judges and the gold set

Some statistics
Cohen’s kappa, Fleiss’ kappa, Krippendorff’s alpha
kappa or alpha values $> 0.8$ is unrealistic

Patterns of disagreements
Program structure

Design HITs that humans can do well

Data pipelines and workflows

Taxonomy creation

Cascade

3 HIT primitives and global structure inference

Near-dupes evaluation

1 HIT for identifying a news article (Mechanical Turk) and 1 HIT for near-dupes detection (UHRS)

Different quality strategies and parallelization


Testing and debugging
The problem

Testing
Attempt to break a program

Debugging
You know the program is broken

How do we test & debug a HIT?

<table>
<thead>
<tr>
<th></th>
<th>Machine computation</th>
<th>Human computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Throw away</td>
<td>Reluctant to throw away</td>
</tr>
<tr>
<td>Testing</td>
<td>Systematic</td>
<td>Ad-hoc</td>
</tr>
<tr>
<td>Debugging</td>
<td>Programmer’s fault</td>
<td>Worker’s fault</td>
</tr>
</tbody>
</table>
A background story

Twitter classifier
Detect if a tweet is interesting or not?

Standard ML approach
Get labels
Feature engineering
Modeling with a tool (e.g., Weka, etc.)
Production classifier

Moderate kappa values

What’s going on?
Debugging framework

Human computation tasks are difficult to debug
Multiple contingent factors

Framework

- Data-worker-task
- Rapid iteration
- Small data sets
- Emphasis on testing before scaling

---

**Q1. Do you think the tweet is interesting to a broad audience?**

- Yes
- No

<table>
<thead>
<tr>
<th></th>
<th>B1 (older, random)</th>
<th>B2 (recent, random)</th>
</tr>
</thead>
<tbody>
<tr>
<td>% interesting</td>
<td>16.7%</td>
<td>14.3%</td>
</tr>
<tr>
<td>Krippendorff’s α</td>
<td>0.013</td>
<td>0.052</td>
</tr>
</tbody>
</table>
Worker reliability and expertise
Borrowed idea from reCAPTCHA: use of control term
Human Intelligence Data Driven Enquires (HIDDEN)
2 more questions as control
1 algorithmic
1 semantic
Adapt your labeling task
HIT with HIDDENs

Tweet de-branded

HIDDENs

The main question

Q1. How many hashtagged words (words that begin with a "#") are in this tweet?

- 0 (no hashtags)
- 1
- 2
- 3 or more

Q2. Does the tweet name a specific person?

- Yes
- No

Q3. Do you think the tweet is interesting to a broad audience?

- Yes
- No

Q1 (alpha = 0.888)
Q2 (alpha = 0.708)
Q3 (alpha = 0.160)
HIT re-design

Tweet de-branded

HIDDENs

Breakdown by categories to get better signal

Q1 (alpha = 0.910)

Q2 (alpha = 0.758)

Q3 Worthless (alpha = 0.384)
Q3 Trivial (alpha = 0.097)
Q3 Funny (alpha = 0.134)
Q3 Makes me curious (alpha = 0.056)
Q3 Contains useful info (alpha = 0.079)
Q3 Important news (alpha = 0.314)
Algorithms for quality control
Algorithms used in practice

Majority vote
Programmatic gold
EM
Get another label
Vox populi


Crowd-workers reviewing work

Soylent
Find-fix-verify
Interactive crowdsourcing

FamilySearch
Arbitration
Peer review

D. Hansen et al. “Quality control mechanisms for crowdsourcing: peer review, arbitration, & expertise at familysearch indexing”, CSCW 2013

M. Bernstein et al. “Soylent: A Word Processor with a Crowd Inside”, UIST 2010
Behavioral features

Focus on the way workers work instead of what they produce

Task fingerprinting

High correlation with work quality

Wernicke

Information extraction

Weighted majority voting

Behavioral features outperform performance-based methods


Practical considerations

What to use?
Depends on complexity and infrastructure access

Voting and honey pots
Cheap and easy to implement

EM-based approaches
Assumes historical performance

Worker verification
More HIT development
Implementation

"Hence, plan to throw one away; you will, anyhow" - F. Brooks
So far ...

This is all good but looks like a ton of work

The original goal: good labels

Data quality and experimental designs are *preconditions* to make sure we get the right stuff.

Labels will be used for rankers, ML models, evaluations, etc.

Don’t cut corners
Development
Coding
Patterns
Modularization
Testing and debugging
Maintenance
Monitoring
## Implementation details

<table>
<thead>
<tr>
<th>Phase</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coding</td>
<td>One language for extracting data from clusters and compute metrics. Avoid moving data from different tools; encoding, data formats, etc.</td>
</tr>
<tr>
<td>Design</td>
<td>Use patterns as much as possible. Examples: iterative refinement, find-fix-verify, do-verify, partition-map-reduce, price-divide-solve. Get ready to throw away HITs and results.</td>
</tr>
<tr>
<td>Modularization</td>
<td>Design HITs that humans can do well. Think in terms of pipelines and workflows.</td>
</tr>
<tr>
<td>Testing and debugging</td>
<td>Don’t patch a bad HIT: rewrite it. Identify problems with data, workers, and task design.</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Version all templates and metadata including payment structure.</td>
</tr>
<tr>
<td>Monitoring</td>
<td>Dashboard and alerts.</td>
</tr>
<tr>
<td>Documentation</td>
<td>Document the essence of the HIT and its mechanics/integration points.</td>
</tr>
</tbody>
</table>
Machines and humans in sync
Delicate balance but lots of potential
When to use a machine or human for computation
Labels for the machine != labels for humans
Best algorithms for the machine may not be the best choices for humans
Takeaways

Repeatable label quality at scale works but requires a solid programming principles

Three aspects that need attention: workers, work and task

Lots of different skills and expertise required

Programming machines is hard, programming applications that involves computations by machines + humans is harder
Thank you!

Book under development

Email: omalonso@microsoft.com
Twitter: @elunca