Introduction to Adaptive Surveys and Amazon Mechanical Turk

Computational Social Science Skills Workshop

Nandana Sengupta

March 2, 2017

Motivation

- Crowdsourcing: employers connect with global pool of free-agent workers to complete a specialized or repetitive tasks
- Traditionally:
  - One way communication
  - Many tasks in-built on crowd-sourcing clients
  - Passive sampling
  - May have biases requester has not controlled for
- Adaptive Surveys:
  - Two-way communication (more efficient)
  - Usually not in-built on clients (harder to code)
  - Active sampling
  - Biased by design

Roadmap for this Talk

- Introduction to Crowdsourcing
- Introduction to Adaptive Surveys
    - Quicksort
    - Multi-armed Bandits
    - Active Sampling
- Skill 1: Run a simple quicksort on python
- Skill 2: Collect crowdsourced data via MTurk
- GitHub repository for session material:
  github.com/NandanaSengupta/MTurk_Adaptive_CSS

Introduction to Crowdsourcing

Crowdsourcing and Mechanical Turk

- Employers connect with global pool of free-agent workers to complete a specialized or repetitive tasks
- Workers: mostly from US (then India), supplementary income
- Employers: mostly academics and non-profits + few big companies
- Academic Impact: In 2015, 800 published studies used crowdsourced data
- Economic Impact: In 2013, $2 billion revenue 48 million registered workers (5 million active)
- Type of tasks
  - Annotation, Tagging, Classification
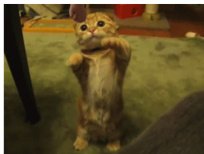 - Ratings, Comparisons

Crowdsourcing and Mechanical Turk

► Sentiment Analysis:

> This was the best book I ever read!!! Thank you so much! :)

Sentiment expressed by the content:

Strongly Positive
Positive
Neutral
Negative
Strongly Negative

► Image Tagging:

Tag 1:

Tag 2:

Tag 3:

What emotion does this GIF invoke:
- ○ Awesomeness
- ○ Funniness
- ○ Sweetness
- ○ No emotion
- ○ Sadness
- ○ Creepiness

Introduction to Adaptive Surveys

Adaptive Surveys

▶ Exploration vs Exploitation

▶

▶ Where can it be used?

   ▶ Information tasks where the space of options is very large

   ▶ When some choices are very informative of whole space

   ▶ Surveys where options space is expanding

▶

QuickSort

- Objective: ranking a list of objects $(k_1, k_2, \cdots k_n)$
- How? (Ratings, Rankings, Pairwise Comparisons)
- Pairwise comparions costly: $n(n-1)/2$ queries
- Quicksort algorithm:

Choose 31 as the pivot

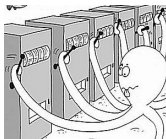| 31	98	13	57	91	20	2	63
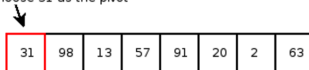
Recursively sort subsequence on each side of pivot

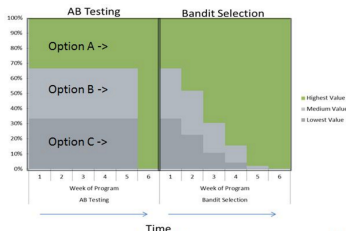| 13	20	2	31	63	91	57	98

- Benefits: On average requires $O(n \log(n))$ comparisons $<<$ random sampling $O(n^2)$
- In the social sciences?
- Ranking of streetviews, ranking of policy options

Multi-Armed Bandits

- Eg: Three options, 'A' 'B' and 'C',
- If 'A' performs best, probability of selecting 'A' higher than 'B', 'C'.
- Still random but depends on current best guess of option values.

- ► Popular algorithms: $\epsilon-$ greedy, UCB, Duelling Bandits
- Disadvantages: Hard to implement, significant result takes time, can converge to suboptimal
- In the social sciences?
- Marketing online products, marketing policies

Active Sampling

- With massive unlabeled pool, sometimes cannot afford exhaustive labelling
- Active sampling: At each time 't', pick 'most uncertain' or 'most informative' items to label
-
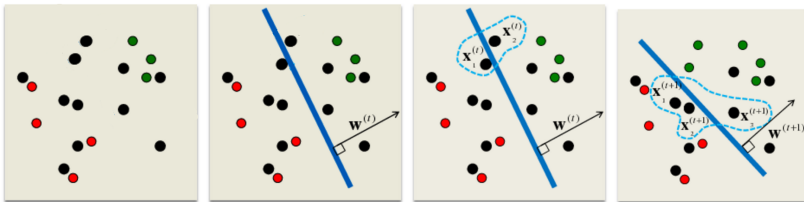
- Disadvantages: Hard to implement, can converge to suboptimal, some comparisons truly uncertain
- In the social sciences?
- Estimating individual preferences via pairwise comparisons

Skill 1: Run a simple quicksort on python

Skill 1: Quicksort demonstration

```python
from random import randrange
from random import sample

# given list, sub-list (defined by start and end indices), random pivot index
def partition(lst, start, end, pivot):
    # place pivot at the end of the sub-list
    lst[pivot], lst[end] = lst[end], lst[pivot]
    lst_below_pivot = []
    lst_above_pivot = []
    query_counter = 0

    # dividing into two lists (above and below pivot)
    for i in range(start, end):
        query_counter +=1
        if lst[i] < lst[end]:
            lst_below_pivot.append(lst[i])
        if lst[i] >= lst[end]:
            lst_above_pivot.append(lst[i])

    next_pivot = start + len(lst_below_pivot)

    lst[start : next_pivot ] = lst_below_pivot
    lst[next_pivot] = lst[end]
    lst[(next_pivot + 1): end+1 ] = lst_above_pivot

    return next_pivot, query_counter

def quick_sort(lst, start, end):
    if start >= end:
        return 0
    pivot = randrange(start, end + 1)
    new_pivot, nqueries = partition(lst, start, end, pivot)
    nqueries += quick_sort(lst, start, new_pivot - 1)
    nqueries += quick_sort(lst, new_pivot + 1, end)
    return nqueries

def sort(lst):
    nqueries = quick_sort(lst, 0, len(lst) - 1)
    return nqueries
```

Skill 2: Collect crowdsourced data via MTurk

Introduction to Amazon Mechanical Turk

- What is M(echanical) Turk?
  - MTurk is a crowdsourcing internet marketplace. *Requesters* get access to an on-demand workforce (*workers*) to perform surveys and tasks which computers are unable to do.
- Is this tutorial for Workers or Requesters?
  - This tutorial is for MTurk Requesters who want to collect and analyze crowdsourced data.
- What is a HIT?
  - HIT stands for Human Intelligence Task – these are the tasks requesters design and workers complete.

Step by Step MTurk Tutorial

- **Step 1:** Setup a Requester Account
- **Step 2:** Purchase Prepaid HITS
- **Step 3:** Create a new survey
  - **Step 3a:** HIT properties
  - **Step 3b:** Design Layout
  - **Step 3c:** Preview and Finish
- **Step 4:** Publish HIT
- **Step 5:** Track Progress and Manage Results
- **Step 6:** Approve Workers and Download Results

Thanks!