HarvardX Research Committee

Subcommittee Report on edX Recorded Course Data – Content, Format, and Analysis Procedures

Joseph K. Blitzstein (Harvard)
Isaac Chuang (MIT)

December 17, 2012

1. Purpose and Scope of Report

Research with edX courses potentially offers transformational insight into educational methods, but hinges upon a basic understanding of the data.

The first step in doing research with data from edX courses is to understand what data are recorded. This is complex, because of the expansive and detailed records kept about virtually every interaction a student has with the edX platform, including not just grades and responses to questions, but also the entire history of attempts, conversations with other students on the forums, and timestamps for interactions with provided text and video material. Recorded course data also intimately involve references to course content and other state held within the platform, such in the discussion forums.

A second step is to appreciate the size and format of the data. The 6.002x recorded course data from the Spring, 2012 semester is largely a text file in excess of 30 GB in size (though much smaller when compressed). The bulk of this data is encoded in "JSON" format, with fields and field value formats which have varied as the underlying edX platform code has evolved. For example, the number of attempts made by a student on a given problem was not logged in this file, until sometime during the Fall, 2012 semester. Other parts of the data are recorded in SQL and no-SQL databases.

A third step is to get a basic start in analyzing the data. How can edX data be decoded into formats used by traditional statistics packages? What are typical procedures for obtaining basic analyses, such as the number of active students versus time?

The purpose of this report is to provide clarity on what edX recorded course data is comprised of, what format it has, and how this data can be analyzed. A comprehensive definition is given for edX recorded course data, and formats are detailed, including a simple analysis example.

2. Definition of “Recorded Course Data”

Each course run on the edX platform involves the recording of the following six categories of data:

a) Course content
b) Tracking logs
c) Student state data
d) Student identification data
e) Forum data
f) Wiki data

These are briefly described below.

Data Category	Description	Data Location
1. Course content	XML files defining the course, including e-text, assessment problems, and links to videos (videos themselves are stored on YouTube).	github (LMS workflow) MongoHQ database (Studio workflow)
2. Tracking logs	Text files recording student interaction “events” with the edX platform, including low-level GET and POST requests to the web server, as well as higher-level server records of assessment responses (answers, grades, question location).	Disks on individual servers in the Amazon Web Services cloud
3. Student state data	SQL records (“StudentModule” table) of student state for each “XModule”, including last viewed location within course, state of responses (including correctness / incorrectness) for each assessment problem, position within videos and sequences.	Amazon RDS (mysql) databases in the cloud, on “instance” servers
4. Student identification data	SQL records of username, userid, name, user profile, courses enrolled in, mapping to external authentication tables.	Amazon RDS (mysql) databases in the cloud, on “portal” servers

5. Forum data	noSQL documents recording forum contributions, comments, responses, votes, links to course content.	MongoHQ database, accessed by comments service running on Heroku (Fall 2013 and later) mysqldb (pre Fall 2013)
6. Wiki data	SQL records of wiki entries made by individual students.	Amazon RDS (mysql) databases in the cloud, on "instance" servers

3. Format of edX Recorded Course Data

Course Content

The detailed XML format of course content is outside the scope of this document. Suffice for here to note that each piece of course content is known as an "XModule" (nb this may be renamed "XBlock" in the future).

XModules include the following: course, chapter, sequential, vertical, videosequence, problem, html, video. There may also be custom XModules, such as book, lecturenotes, slides, transparency.

Each XModule is uniquely identified by a Location, which has the format:

```
i4x://ORG/COURSEID/…
```

where ORG is the X organization, and COURSEID is a unique course identifier number/string. For example, here is the Location of a specific problem in 8.02x:

```
i4x://MITx/8.02x/problem/lec4_Q3
```

These Location strings are used to identify many data records, eg in the wiki, forum, and tracking logs.

Tracking Logs

The "Tracking Logs" category of data comprises the bulk of the recorded course data, by a large margin. This is because these logs include detailed low-level records of virtually every interaction the student's web browser has with the edX server. These records are produced by "middleware" in the server stack, which intercept all GET and POST transactions, and record them to the tracking logs. Because of the design of the edX user interface, subsections of each web page are loaded dynamically upon request (using "AJAX" transactions) from the server.

In addition, the edX server also produces “server source” records, which are higher-level. These include records about user profile changes, submission and grading of problems, including correctness / incorrectness, and Location strings for content being accessed.

The elementary unit of tracking logs is known as an *event*. Each event is stored as a single-line JavaScript Object Notation (JSON) format text string, with the following information (example given for GET event):

```
event = {"username" : username,
         "session" : scookie,
         "ip" : request.META['REMOTE_ADDR'],
         "event_source" : "browser",               # may also be “server”
         "event_type" : request.GET['event_type'],
         "event" : request.GET['event'],           # may contain large amount of data
         "agent" : agent,
         "page" : request.GET['page'],
         "time": datetime.datetime.utcnow().isoformat(),
}
```

Types of events logged include:

- User-interface events: accordion navigation, sequence navigation (seq_prev, seq_next, seq_goto), book navigation (gotopage, prevpage, nextpage), page navigation (page_close)
- Problem events: problem_check, problem_reset, problem_show, problem_save, save_problem_check
- Profile events: email change, name change, password change
- Web server events: GET, POST
- Video player events: load_video, play_video, pause_video, stop_video, player_error

Illustrative tracking log interaction

Consider, for example, what happens when a student clicks on the “check” button, while viewing an assessment problem named “Angular Momentum of a toy Gyroscope”. This single click produces *three* events in the tracking log. The most interesting of these is the “save_problem_check” event, generated by the server, which looks like this:

```
{
  "username": "user101",
  "event_source": "server",
  "ip": "18.99.99.99",
  "event_type": "save_problem_check",
  "agent": "Mozilla\/5.0 (Macintosh; Intel Mac OS X 10_8_2)
AppleWebKit\/536.26.17 (KHTML, like Gecko) Version\/6.0.2 Safari\/536.26.17",
  "event": {
    "success": "incorrect",
    "correct_map": {
      "i4x-MITx-8_01rq_MW-problem-Angular_Momentum_of_a_toy_Gyroscope_2_1": {
        "hint": "",
        "hintmode": null,
        "correctness": "incorrect",
        "msg": "",
        "npoints": null,
        "queuestate": null
      }
    },
    "state": {
      "student_answers": {
      },
      "seed": 1,
      "done": false,
      "correct_map": {
      }
    },
    "answers": {
      "i4x-MITx-8_01rq_MW-problem-Angular_Momentum_of_a_toy_Gyroscope_2_1":
"choice_7"
    },
    "attempts": 1,
    "problem_id":
"i4x:\/\/MITx\/8.01rq_MW\/problem\/Angular_Momentum_of_a_toy_Gyroscope"
  },
  "time": "2012-12-04T22:49:14.437612",
  "page": "x_module"
}
```


Note the “problem_id” is Location string. “seed” is a random number generator seed (for randomized problems). “correct_map” is a dictionary mapping each problem Location to information about the correctness of the response given. And “answers” records responses given by the student, here “choice_7”.

4. Analysis Procedures: Examples

Consider a mundane task, such as producing histogram plots of the responses given, for each question in a course. This can be obtained from the tracking log data. A simple python program can be used to generate a comma-separated-values (CSV) format file from the tracking logs; the CSV file can then be imported into R (a common statistical analysis package).

Here is a python script, which produces such a CSV file from a tracking log file named “tracking.log.deid”:

```
#!/usr/bin/python
#
# user, problemid, questionid, answer, timestamp, for all questions on save_check

import json, csv

fp = open('answerhist.csv','w')

cw = csv.writer(fp,dialect='excel')
cw.writerow(['user','problemid','questionid', 'answer', 'attempts', 'timestamp'])

for k in open('tracking.log.deid').readlines():
    x = json.loads(k)
    etype = x['event_type']
    if not etype=='save_problem_check':
        continue

    ev = x['event']
    state = ev['state']
    attempts = ev['attempts']
    probid = ev['problem_id']
    sa = state['student_answers']

    u = x['username']
    ts = x['time']

    for q in sa:
        cw.writerow( [ u, probid, q, sa[q], attempts, ts ] )
```

The columns of the CSV file are user, problemid, questioned, answer, attempts, and timestamp. The timestamp is given as a datetime stamp, which needs to be converted into a format suitable for R.

Here is an import script for loading the CSV file produced, into R:

```
answers <- read.csv("~/Desktop/edX/answerhist.csv")
for (j in 1:10) x[j] <- as.numeric(strptime(m2[j,2], "%Y-%m-%dT%H:%M:%S"))
```

A similar set of scripts can be used to obtain plots of when “checks” are requested, versus time. Here is the python script:

```
#!/usr/bin/python
#
# timestamp for each problem check

import json
import csv

fp = open('checktimes.csv','w')

cw = csv.writer(fp,dialect='excel')
cw.writerow(['username','problemid','timestamp'])

for k in open('tracking.log.deid').readlines():
    x = json.loads(k)
    etype = x['event_type']
    if not etype=='save_problem_check':
        continue

    u = x['username']
    ts = x['time']
    event = x['event']
    cmap = event['correct_map']
    problems = cmap.keys()

    for p in problems:
        cw.writerow( [ u, p, ts ] )
```

And this is the R script:

```
usertimes <- read.csv("~/Desktop/edX/checktimes.csv")
matrix(usertimes)
```

The result is a matrix “usertimes” which contains the username, problemid, and timestamp.