LoDEg: log analytics and user profiling for the LODE platform

Riccardo Capraro

October 30, 2017

Contents

1 Introduction

The main purpose of this project is to gather insights on the population of users of the LODE platform. We will list here the results we have achieved and the techniques we have used to carry out this task; the present document is not meant to be a thorough documentation of the project, indeed it will be an overview of all its parts, focusing on why each choice was made. At the end of this article the reader will have a vision of the big picture, the building blocks and the how-tos.

1.1 Project objectives

Three are the key objectives:

1. To gather data from the system (with respect to both the Apache server and the LODE platform) and display them in a meaningful way through statistics, via a simple and manageable UI;

2. To understand the correlations between user data and to point out some interesting non-evident information;

3. To build models that can understand user behaviours using only simple features that we can extract from the log files of the platform.

2 Overview

This is the main section of the article; requirements lead the development of the project and reading this section is necessary to fully understand some of the main implementation choices made throughout the project.

2.1 System requirements

The extraction system has to be extensible, configurable and should expose a simple API; it should work in-memory, with capabilities that ensures that the system can work while keeping both memory and DBMS access levels low; the computational part of the website has to be done at server side and statistics should travel the network only when requested; the whole process has to work fast enough to allow synchronous interfaces. The system representation has to be flexible, yet well-defined, in order to allow other statistics to be included and excluded dynamically while relying on a consistent underling structure. At least one way of exporting the raw statistics has to be provided.

2.2 System architecture

The LoDEg platform is made out of two main components:

- a web user interface built on Django that can be reached through the Apache server;

- the lodegML(should rename this!!) library, fully integrated into the Django application via its facade API.

When can then consider other two components: the MongoDB database where the logs are stored and a generic caching system that supports the website interface (the details depend on the implementation).

3 Apache server logs

3.1 Why AWStats?

3.2 How we integrated it

4 Data structure

In this section we provide a short description of the log system, focusing on the records structure; we then dive into the system internal representation (with respects to both the model and the implementation).

4.1 Log structure (LODE v3.X)

The log structure presented in this paragraph refers to LODE v3.x; as historic reference we mention that a first logging system was developed in version 2.x and has been used during the early development of the LoDEg system [reference]. Yet, no working version of LoDEg has been developed for LODE v2.x.

All log records have two mandatory fields: **user_id** and **session_id**; they then belong to one of the following categories:

Type	Value1	Value2	Value3	Description
title	lesson_id	course_id	device	Used to link the session to the lesson
play	time_rel	time_abs	_	It marks whenever the user presses the play button
pause	time_rel	time_abs	_	Used to track when the user pauses the video
alive	time_rel	_	_	Every 5 min without any interaction we acknowledge that the session is still running
note	note_type	note_duration(???)	_	Stored when the note is taken
mute	time_rel	_	_	_
layout	layout_type	_	_	layout_type $\in$ {ps,p,s}
speed	time_rel	new_speed	_	Used to track video speed changes
jump	from(time_rel)	to(time_rel)	jump_type	jump_type $\in$ {click_or_drag, keyframe, note}

4.1.1 Granting consistency

Consistency is one of the driving paradigms of the new record structure; to ensure session validity we set the following:

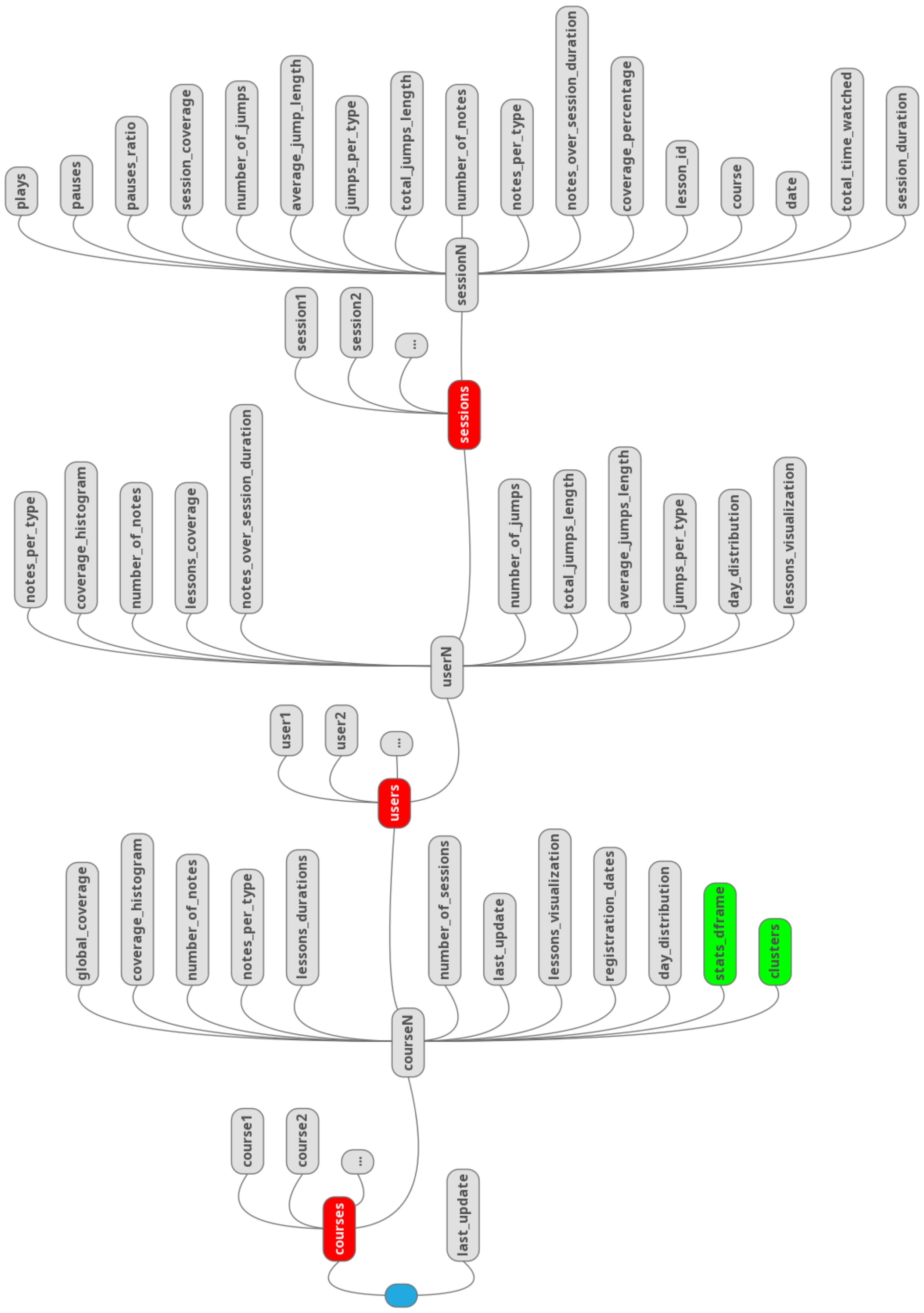
- Alive records are added to allow us not to lose the whole session if the user abruptly closes the browser;
- The session end is always known: it is marked by the last record of type play, pause or alive;
- No jump is logged if either of from or to fields is unknown;

4.1.2 Why time_rel and time_abs?

Time_rel represents the relative time position in the video the user is watching; it is used throughout the session to track reproduction intervals and jumps. As a workaround to network delay, which can create a gap between MongoDB object creation timestamps and client_side time, time_abs are kept to improve precision.

4.2 System structure

The internal structure skeleton has four logical levels: systemInfo, courseInfo, userInfo and sessionInfo.



courses
last_update
course1
course2
...
courseN
global_coverage
coverage_histogram
number_of_notes
notes_per_type
lessons_durations
number_of_sessions
last_update
lessons_visualization
registration_dates
day_distribution
stats_dframe
clusters
users
user1
user2
...
userN
notes_per_type
coverage_histogram
number_of_notes
lessons_coverage
notes_over_session_duration
number_of_jumps
total_jumps_length
average_jumps_length
jumps_per_type
day_distribution
lessons_visualization
sessions
session1
session2
...
sessionN
plays
pauses
pauses_ratio
session_coverage
number_of_jumps
average_jump_length
jumps_per_type
total_jumps_length
number_of_notes
notes_per_type
notes_over_session_duration
coverage_percentage
lesson_id
course
date
total_time_watched
session_duration

4.2.1 Why dictionaries instead of classes

There are several reasons why the system has been modeled using dictionaries instead of classes; though the system can be rewritten using classes we chose dictionaries while keeping in mind some considerations; the system is meant to be:

- Exportable. It is directly jsonazible (and pickable), property that allows to implement efficient caching systems.
- Expandable. To extend the system capability (i.e, to add statistics) is as simple as to add a value to a python dictionary.
- Modular. Every part of the system (but the skeleton structure) is not mandatory; this provides a great flexibility and can enhance memory usage optimization when computations are made just-in-time.

5 Data extraction

In this section we will list the objectives of the first refinement step of the data analysis, the data extraction; we will then provide a description of the process, focusing on the main implementation choices.

5.1 Extraction objectives

A simple list of the features that the system extracts form the data:

1. Videos:
   (a) For every single session/user, which parts of which videos have been watched (a list of disjoint intervals for each session/user);
   (b) For every single video, which users/sessions have watched which parts of the video;
   (c) When users watch the videos during the day (morning, afternoon, ...) ¡-Missing
2. Notes:
   (a) The number/type of notes taken;
   (b) Number of notes over time (n/t);
3. Data to train a clustering algorithm (number of pauses/jumps, pauses length, ...).
4. Other information such as mute, speed, layout and device information (detail!).

5.2 Extraction process

The process of data extraction is supported by the MongoDB database. The system queries the database on user demands (using either the web interface or the LodegSystem API). Two options are provided:

- a single query that stores all the data into the system (minimum DBMS access rate, high memory levels, average computing time);
- a batch query set, with the number of queries linearly dependent to the number of users (medium DBMS acess rate, low memory levels, average computing time).

The data is split in user sessions. The raw records are queried, parsed and immediately discarded (by default) after the session extraction is completed; this is the mechanism that helps keeping the memory usage low when the batch query policy is adopted. Other flags can be set to change the system memory management policy, though optimal defaults have already been provided. The extraction is modular (i.e., we do not need all the statistics to be computed for the system to work); the extraction can be at system, course, user or session level. At the end of this stage the statistics are processed to compute horizontal statistics for the higher levels (sessions for users, users for courses, courses for the system).

6 Data manipulation

In this section we explain some of the main ; the last step of data manipulation is user clustering.

6.1 Statistics that matter

6.2 User profiling: clustering users

7 User interface and lodegML integration

A lodegML

lodegML is the library that handles the processes of data extraction and manipulation. The library exposes its functionalities via a facade class (LodegSystem) in the module system.py; the API is structured in such a way that command line, scripts and the WebApp can interface with it. Some general information worth noting are:

- The system is highly configurable, i.e. all parameters ¡but the caching system¿ can be modified by users after class initialization; three optimized configurations are provided (default, low_mem and web).
- A caching system is provided with two development implementations: MongoDB and SQLite. There are some limitations: the MongoDB cache can work with little

data (max 16MB); the SQLite cache current implementation is based on Django internal SQLite database and it has been developed for the WebApp only. This limitations are overcome by the import/export system.

- Both parts of the system and the system as a whole can be exported/imported in json or binary ¡pickle, .p¿ format.
- Some CourseInfo statistics can be exported to CSV from the WebApp.

A.1 Documentation

The Sphinx tool has been used to compile the documentation of the library; we chose google docstrings format since it is more readable and compact than Sphinx native syntax. The Napoleon plugin is then used to map Google docstrings to sphinx syntax.

A.2 Algorithm complexity

A.3 Test suite

Testing has been developed to ensure code robustness and correctness. Main tests have been set for key algorithms, database queries and the extraction processes. The project has been developed while LODE v3.X was under construction; we added a muck-up population to the test suite to allow early testing before the platform release (LODE v3, October 2017).