Approach to analysis of OpenAIRE affiliations - lessons learned. Exploratory data analysis using Makefiles, SQLite, R + knitr.

Date: 2016-05-17
Author: Mateusz Kobos
Addressee: Audience of the ADA Lab seminar

Links to files of the report described in this document:

- Source code
- Produced HTML report

Comments by the audience members that were made during the presentation are highlighted in yellow.

Table of content:

1 Goal of the task

My task was to do exploratory analysis of affiliation information extracted from PDF and XML files (in JATS format) by CERMINE in IIS data mining subsystem of OpenAIRE system. The goal of this task was to provide basic information about the data to be used in a task of matching extracted affiliations with a register of organizations coming from OpenAIRE's Information Space.

In CRISP-DM data mining methodology, this is the stage in a data mining project called "**data understanding.**" According to a side note in "CRISP-DM 1.0 Step-by-step data mining guide", this stage takes 20%-30% of the project's time and consists of: computing basic statistics for each attribute, analyzing attributes correlations, checking missing data, identifying noise and inconsistencies in the data.

2 Directory structure and general setting

The output produced by this task was a report from the exploratory analysis of the data.

- Whole analysis was done in a **single directory**. Its most important contents:
  - Report files:
    - analysis.rmd - file with the source code of the report in RMarkdown (R + Markdown supported by knitr) format
    - analysis.html - the final report file generated from analysis.rmd
  - Data directories:
    - input - directory with input data that has not been pre-processed yet, i.e. raw input data
    - working_dir - directory with intermediate data
    - output - directory with data that has already been pre-processed; this data is used in analysis.rmd document
  - Makefile - rules for taking data from input directory through working_dir and placing the result in output directory. It can also be used to generate the analysis.html file.
  - README.markdown - various notes, including information where the data in the input dir came from
  - Source files:
    - R - auxiliary R scripts used in analysis.rmd file
    - bin - mainly Python scripts for doing the pre-processing; their execution is coordinated by Makefile
  - Other subdirectories contain sub-parts of the analysis. Each one has the same general structure as described above.
- The report was written and compiled using **RStudio IDE**.
- Previously, I experimented with putting the whole analysis into a **single R package where the report is a R package's vignette** in RMarkdown format (approach described in a [blog post]). I gave it up because of the following disadvantages.
  - Apparently caching code chunks in knitr doesn't work. As a result, it takes a lot of time to generate the report after introducing any change.
    - MBojan: It works on my computer. BTW, you can define dependencies between code chunks so when one changes, the dependent ones are re-generated. I don't use it however, because it's too much effort.
  - After updating the R code (in the R directory) used by the report, you need to install the whole package in the system in order for its functions to be visible in the report. This can be done easily by pressing Ctrl+Shift+B in RStudio but it takes a while and is not done automatically. This is not a problem when the report is not a part of a R package and you simply use source function to use the external R code.

3 Makefile

Excerpt from the Makefile[1]:

```
all: analysis.html

## Generate the report. This target defines dependencies on the data files,
## auxiliary R source code files, and the source RMarkdown document.
analysis.html: analysis.Rmd R/utils.R ~/current-local/icm/output/affiliations-sample.db input/data_sources.tsv
	./render_rmd.sh $(word 1,$^)
## Mateusz: the `./render_rmd.sh` script is a workaround because generation of the code using a standar
## one-liner didn't work on one of my two computers for some reason.
## MBojan: Maybe the absolute paths that you are using here are to blame. It's also important not to cache
## code where libraries are loaded.

## Generate affiliations.db file. Note the usage of `$(word 1, $^)` and `$@` - this lets you reference consecutive elements in the dependency list in the first case and the target path in the second case. This allows you to follow the Don't Repeat Yourself rule.
~/current-local/icm/output/affiliations.db: ./bin/extend_table.py ~/current-local/icm/working_dir/affiliations.db
	$(word 1, $^) --input_db $(word 2,$^) --output_db $@ --create_table_script
create_affiliations_with_additional_columns.sql --create_indexes_script
create_affiliations_with_additional_columns-indexes.sql

## (...)

## Copy data generated in a sub-project
input/data_sources.tsv: ./data_sources/output/data_sources.tsv
	cp $(word 1,$^) $@

./data_sources/output/data_sources.tsv: make-data_sources

## Run make in sub-project
make-data_sources:
	$(MAKE) -C data_sources

## Clean knitr cache
clean-cache:
	rm -rf analysis_cache
	rm -rf analysis_files

## Remove all temporary and intermediary data here and in sub-projects
clean: clean-cache
	cd working_dir; rm -rf *
	cd output; rm -rf *
	cd bin; rm -rf __pycache__
	$(MAKE) -C data_sources clean
```

4 Data pre-processing

- A trick that can save some space on your disk in case your processing can be done on an input stream: **unpack compressed file on-the-fly and process it**, e.g.

[1] You can observe that some of the files are placed in an external directory ~/current-local/icm due to a special way my environment and its backup policy was set. You normally wouldn't reference any external directory.

```
tar -zxOf input.tar.gz | process_data.py > output.csv
```

    - Bolo: If it's a single file, you don't have to use tar, you can simply use gzip.
    - MLopusz: R and some other programs read gzipped files out-of-the-box, so in such cases you don't have to unpack them explicitly.
- I operated on data big enough that R wasn't able to ingest it using standard read.csv function (~6 million records, ~2GB of data in a SQLite table without indexes). Because of this, I decided to put the data in a **SQLite database** (the data was held in a single table). This sped up things, but not enough. Thus, finally I settled on analyzing only a **10% random sample** of the data.
    - Bolo: I work on a data that is even bigger than this and SQLite works quickly.
- If you use SQLite through standard Python interface (sqlite3 package) and want to **load the database with many rows**, use cursor.executemany instead of using many separate cursor.execute calls. This makes the insertion much faster (since in the former case, many inserts are put into the table in a single transaction).
- If you create a **SQLite database and then fill it with data and you want to have indexes**: first create the database without indexes, then insert data, and then create indexes. This makes the whole operation much faster (in my case this reduced the time from hours to minutes).

5 RMarkdown report

- **Code chunks were cached by default** (use opts_chunk$set(cache = TRUE) code i n the first code chunk) to speed up re-generation of the report after introducing changes.
    - Note that knitr re-executes only the code chunks that have been changed. So if you change value of a variable in one chunk, but the variable is also used in another chunk, this dependent chunk won't be re-generated. My solution: remove the directory with cache in such case (make clean-cache command).
    - MBojan: You shouldn't cache libraries nor any other external resources.
- You **cannot cache code chunks with DB** connection, thus you need to set caching to false for chunks that introduce them.

```
```{r cache=FALSE}
db <- src_sqlite("~/current-local/icm/output/affiliations-sample.db")
df <- tbl(db, "affiliation")
```
```

- I used dplyr to query the data. However, I often found myself translating SQL query into a dplyr query in my head to arrive at dplyr construction that I want. **Maybe using some SQL interface for data frames (say sqldf package) instead of dplyr** would be a better approach? Example of a complex dplyr query:

```
top_repos <- infix_doc %>%
  group_by(id_infix) %>%
  summarize(count = n()) %>%
```

```
arrange(desc(count)) %>%
mutate(percentage_of_pdf_documents = (count*100)/documents_count) %>%
as.data.frame() %>%
left_join(data_sources, by = c("id_infix" = "prefix")) %>%
select(id_infix, name, percentage_of_pdf_documents, count)
```

- MBojan: Instead of as.data.frame(), you can use collect() to obtain a dplyr frame.

- I accessed the data using **dplyr's tbl backed by the SQLite database** rather than a standard data.frame. This provides an illusion that you're operating on normal data frames instead of on database.
  - Pros:
    - You can manipulate a database-backed table as if it was a normal data frame.
  - Cons (their common feature: dplyr's tbl provides a leaky abstraction of a data frame):
    - Some dplyr functions are not implemented when working on a database-backed table. However, there's usually an easy workaround, e.g. instead of n_distinct, use distinct followed by count; instead of top_n, use head.
    - The very useful summary function provides much more rudimentary information than in case of a data frame.
    - To access specific fields of the table, you need to convert the database-backed table to a standard data frame.
- If you want to show a long table in the report, but don't want it to occupy a lot of space, you can wrap it in a widget embedded in the output HTML page using **DataTable javascript library**.

Show 10 entries Search:

organization	count	percentage
NA	1356	0.2
Harvard-Smithsonian Center for Astrophysics	608	0.1
University of California	463	0.1
Department of Physics, University of California	430	0.1
Space Telescope Science Institute	334	0.1
European Southern Observatory	320	0.1
Delft University of Technology	294	0.0
Department of Physics	287	0.0
California Institute of Technology	276	0.0
University of Twente	272	0.0

Showing 1 to 10 of 100 entries Previous 1 2 3 4 5 … 10 Next

  - One way of doing this is to [blog post]:
    - 1) place the following snipped somewhere at the top of the RMarkdown file:

```
<!-- The code below turns tables into their interactive JavaScript versions using
DataTable library -->
<link rel="stylesheet" type="text/css"
```

```
href="http://cdn.datatables.net/1.10.5/css/jquery.dataTables.min.css">
<script src="http://code.jquery.com/jquery-2.1.2.min.js"></script>
<script src="http://cdn.datatables.net/1.10.5/js/jquery.dataTables.min.js"></script>
<script type="text/javascript">
  $(document).ready(function() {
    $(".dtable").DataTable({
      "order": []
    });
  });
</script>
```

- 2) instead of standard call of kable function to print table, use kable(..., table.attr = "class='dtable'", format = "html").

- Pros:
  - You can cache the code snippets where the DataTable is used (the "standard" R package DT that lets you use DataTable library doesn't allow this).
- Cons:
  - I'm not sure if you can bump up the versions of the JavaScript libraries that used. I tried to replace these versions in the snippet above with the most current versions but it didn't work properly.
  - Rarely, the DataTable wrapper doesn't want to display a specific table - a popup with a not very informative error message is shown.