Data acquisition, extraction, and storage

Processing other (non-HTML, non-tabular) data formats

Pierre Senellart

15 November 2024

Variety of data formats

- We already discussed how to extract data from HTML (CSS/XPath selectors, scrapy) and how to process tabular data (SQL, pandas, command line tools)
- But open data, standard datasets, Web data, API results, etc. comes in a variety of formats beyond HTML and tabular data
- For instance, see statistics of Common Crawl data (not necessarily representative of the Web, but gives some illustration): `https://commoncrawl.github.io/cc-crawl-statistics/plots/mimetypes`

High-Level Categories (1/2)

Print-Ready Documents PDF, PostScript

Image Formats PNG, JPG, WebP, SVG, GIF, DejaVu

Video and Audio (Container) Formats MP3, MP4, OGG, Matroska

Office-Suite Documents OpenDocument, Office Open XML, RTF, legacy formats

E-Book Documents EPUB, MobiPocket

High-Level Categories (2/2)

Semi-Structured Data Exchange Formats JSON, XML, YAML

RDF Data N-Triples, Turtle, JSON-LD, RDF/XML

Application-Specific XML formats Atom, RSS, KML

Application-Specific Text-Based Formats iCalendar, BibTeX, vCard, EndNote, MARC

Application-Specific Binary Formats ProCite, netCDF

Compressed and Archived Data ZIP, Gzip, BZip2, tar, RAR, xz, 7z

Plain text

Print-Ready Documents

- PostScript: programming language for describing printed pages (most professional printers include a PostScript interpreter)
- PDF: simplification of PostScript, removing the Turing-complete nature of the language but keeping operators for positioning text and other material on pages
- In both, focus on description of printed pages, most of the structure (styles, paragraphs, environments, etc.) of the original (LaTeX, desktop publishing or office suite software) document is lost
- Some tools (Adobe Acrobat, office software, `pdfalto`, `pdftohtml`) aim at recovering the original structure, but imperfect; sometimes OCR is the only solution!
- Grobid: state-of-the-art in information extraction from scholarly articles

Example PostScript code

$$\sum_{n=0}^{+\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

```
TeXDict begin 1 0 bop 1722 227 a Ff(+)p Fe(1)1721 252
y Fd(X)1718 428 y Fc(n)p Ff(=0)1890 275 y Fb(1)p 1867
312 88 4 v 1867 388 a Fa(n)1917 364 y Ff(2)1987 331 y
Fb(=)2085 275 y Fa(\031)2135 244 y Ff(2)p 2085 312 V
2108 388 a Fb(6)p eop end
```

(`TexDict`, `bop` defined elsewhere in the PostScript document.)

Example PDF code

$$\sum_{n=0}^{+\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

```
/F19 6.9738 Tf 278.63 742.659 Td [(+) ] TJ
/F25 6.9738 Tf 6.116 0 Td [(1) ] TJ
/F13 9.9626 Tf -6.282 -2.989 Td [(X) ] TJ
/F22 6.9738 Tf -0.311 -21.099 Td [(n) ] TJ
/F19 6.9738 Tf 4.925 0 Td [(=0) ] TJ
/F20 9.9626 Tf 15.677 18.374 Td [(1) ] TJ
/F23 9.9626 Tf 296.021 723.371 Td [(n) ] TJ
/F19 6.9738 Tf 5.98 2.878 Td [(2) ] TJ
/F20 9.9626 Tf 8.433 3.956 Td [(=) ] TJ
/F23 9.9626 Tf 11.711 6.74 Td [({19}) ] TJ
/F19 6.9738 Tf 6.037 3.615 Td [(2) ] TJ
/F20 9.9626 Tf 324.907 723.371 Td [(6) ] TJ
```

Image, Video, Audio Formats

- Require specific processing for image, audio, video data
- Often accompanied with metadata (e.g., EXIF for JPG or PNG) sometimes reach in structured information (date of production, title, geolocation, camera characteristics, etc.)
- Video and audio containers sometime context textual data (subtitles, possibly in different languages), that can also be exploited

Office-Suite Documents

- OpenDocument and Office Open XML: two competing "open" formats for office software: word processing, spreadsheet, presentation, drawing
- In both cases: ZIP-archive with XML documents and accompanying files; sometimes feasible to work with these XML files
- Microsoft's Office Open XML is somewhat hard to handle by following the standard (lots of legacy annotations), OpenDocument somewhat cleaner
- Specific libraries exist to handle specific data (e.g., Excel OOXML spreadsheets)
- Contain much metadata (authors, revisions, etc.) which may be of independent interest

E-Book documents

- Formats adapted for screen-readable books
- Historically many competing formats, but nowadays one common format (by far): EPUB
- ZIP archive containing:
  - metadata, table of contents, etc., in XML
  - actual text within XHTML files
  - styling in CSS, possibly animation in JavaScript
  - Images in common Web formats
- Basically, a Web site (without any server-side interaction) packaged within a file

Semi-Structured Data

- A data model seen as alternative to the classical (structured) relational data model.
- Main principles:
  - Based on an acyclic graph (or most often tree) structure instead of relations
  - Self-describing data: metadata (names, sometimes types) and values represented together, no need for separate description
  - Flexible typing: no typing by default, mix of very structured and unstructured (text) content, but possible to type
  - Serialized form: well-specified, open standard, and self-contained serialized form, which can be stored and exchanged

Semi-Structured Data Formats

- XML (eXtensible Markup Language): engineered as an ideal semi-structured data format; simplification of SGML, standardized, coming with a wide collection of companion standards for typing (DTD, XML Schema), navigation (XPath), querying and transformation (XQuery, XSLT), etc. Standardized by the W3C.
- JSON (JavaScript Object Notation): emerged as an alternative data format by the fact that JavaScript was the programming language of the Web (definitely on the client side, to some extent on the server side), which made it convenient. Now standardized by ECMA, independently of ECMAScript/JavaScript.
- YAML (Yet Another Markup Language): proper superset of JSON, more feature-rich (node references, making it a real graph structure, explicit typing, multi-document serialization, etc.). Indentation-dependent, but JSON-compatible bracket/braces notation also available. De facto standard, see `https://yaml.org/`

Implementations and Application Formats

- Implementations of XML, JSON, YAML parsers and serializers in most programming languages
- JSON is now the most common, followed by XML, followed by YAML (whose complex features make it a bit less intuitive)
- Specific applications may rely on an XML, JSON, or YAML syntax with specifications of which documents are valid for this application
- Examples:

XML	XHTML, SVG, MathML, KML, Atom, RSS, OOXML, OpenDocument, XSLT, RDF/XML
JSON	JSON-LD, every API with JSON responses
YAML	GitHub/GitLab Workflows

Example: XML

```
<movies>
  <movie>
    <title>Seven</title>
    <year>1995</year>
    <genre>crime</genre>
    <summary>
      A film about two homicide detectives' desperate
      hunt for a serial killer who justifies his crimes
      as absolution for the world's ignorance of
      the Seven Deadly Sins.
    </summary>
    <country>USA</country>
    <director id="31">
      <last_name>Fincher</last_name>
      <first_name>David</first_name>
      <birth_date>1962</birth_date>
    </director>
  </movie>
</movies>
```

Example: JSON

```
{
  "movies": {
    "movie": {
      "title": "Seven",
      "year": "1995",
      "genre": "crime",
      "summary":
        "A film about two homicide detectives' desperate\n      hunt fo
      "country": "USA",
      "director": {
        "+@id": "31",
        "last_name": "Fincher",
        "first_name": "David",
        "birth_date": "1962"
      }
    }
  }
}
```

Example: YAML

```yaml
movies:
  movie:
    title: Seven
    year: "1995"
    genre: crime
    summary: |-
      A film about two homicide detectives' desperate
            hunt for a serial killer who justifies his crimes
            as absolution for the world's ignorance of
            the Seven Deadly Sins.
    country: USA
    director:
      +@id: "31"
      last_name: Fincher
      first_name: David
      birth_date: "1962"
```

Memory vs Stream-Based Processing

- Semi-structured documents are sometimes very heavy and may not fit in main memory
- Some processing can be done by a linear browsing of the document, do not require the document do be loaded in main memory
- By default: most parsers load the document in main memory
- But there exists streaming parsers (sometimes called online parsers) that process the document as needed
- In XML: DOM parsers vs SAX parsers
- Alternative: disk-based processing (e.g., XML databases such as eXist)

Declarative Querying: XML

- XPath, standard language for expressing complex navigation patterns
- XQuery, standard language for expressing complex queries in XML documents (Turing-complete)
- XSLT, standard language for expressing transformation of XML documents (Turing-complete)
- Often possible to use either XQuery or XSLT, XQuery more easy to start with
- XSLT 3 supports streaming processing (but few implementations!)

Declarative Querying: JSON

- Technology and language environment around JSON is not as rich (and not as standardized) as around XML (and even worse for YAML)
- Competing tools/languages for declarative querying of JSON documents:

  JSONPath — one of the earliest, many implementations but a bit limited; streaming processors exist

  JMESPath — becoming the most commonly used, formally specified with many implementations, used as filters in some APIs (e.g., AWS CLI), better designed than JSONPath, not as powerful as XPath (only child axis available)

  jq — full-featured, Turing-complete, JSON processor command line tool (see yq for a YAML-compatible wrapper on jq); streaming parsing available

- All are inspired by XPath and share a (superficially) similar syntax; JMESPath and jq have pipelines similar to Unix's

RDF Data

- Subject–Predicate–Object RDF facts, for the Semantic Web and the Linked Open Data
- Rich collection of datasets, notably over SPARQL endpoints
- Various RDF serializations (text-based, XML-based, or JSON-based)
- Can be processed in an ad-hoc manner depending on serialization format, or using specific libraries, such as RDFLib

N-Triples example

```
<https://pierre.senellart.com/>
  <http://xmlns.com/foaf/0.1/name> "Pierre Senellart" .
<https://pierre.senellart.com/>
  <http://xmlns.com/foaf/0.1/mbox> <mailto:pierre@senellart.com> .
<https://www.di.ens.fr/michael.thomazo/>
  <http://xmlns.com/foaf/0.1/name> "Micha\u00EBl Thomazo" .
```

Turtle example

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<https://pierre.senellart.com/>
  foaf:name "Pierre Senellart" ;
  foaf:mbox <mailto:pierre@senellart.com> .

<https://www.di.ens.fr/michael.thomazo/> foaf:name "Michaël Thomazo" .
```

JSON-LD example

```
[
  {
    "@id": "https://pierre.senellart.com/",
    "http://xmlns.com/foaf/0.1/name": [
      { "@value": "Pierre Senellart" }
    ],
    "http://xmlns.com/foaf/0.1/mbox": [
      { "@id": "mailto:pierre@senellart.com" }
    ]
  },
  {
    "@id": "https://www.di.ens.fr/michael.thomazo/",
    "http://xmlns.com/foaf/0.1/name": [
      { "@value": "Michaël Thomazo" }
    ]
  },
  { "@id": "mailto:pierre@senellart.com" }
]
```

RDF/XML example

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/">

  <rdf:Description rdf:about="https://pierre.senellart.com/">
    <foaf:name>Pierre Senellart</foaf:name>
    <foaf:mbox rdf:resource="mailto:pierre@senellart.com"/>
  </rdf:Description>

  <rdf:Description rdf:about="https://www.di.ens.fr/michael.thomazo/">
    <foaf:name>Michaël Thomazo</foaf:name>
  </rdf:Description>

</rdf:RDF>
```

Compressed and Archived Data

- Compression: using a compression algorithm to reduce the size of a file
- Archival: storing an entire file hierarchy within a single file
- Various compression (Gzip, BZip2, xz), archival (tar), or archival-and-compression (ZIP, 7z, RAR) formats
- Instead of un-compressing/archiving, especially for very large files, may be useful to deal with data in a streaming manner:

```
import gzip
with gzip.open('foobar.csv.gz', 'rb') as f:
    for line in f:
        pass # Process line
```

Application-Specific Formats

- Check for dedicated libraries
- If dealing with XML, JSON, YAML data, use standard XML, JSON, YAML tools, especially declarative ones
- In the worst case, write your own parser
- If needed make sure data is processed in a streaming fashion