ETH zürich

Working with (big) data II

Data Science for Public Policy

Christoph Goessmann
Law, Economics, and Data Science Group (Prof. Elliott Ash)

Course: 860-0033-00L Data Science for Public Policy: From Econometrics to AI, Spring 2024

14 March 2024

Repetition

What did we do last time?

- I/O bound vs. CPU bound processes
- Basic command line usage
- Python virtual environments (`pipenv`)
- Automatic logging with timestamps (`import logging`)
- Estimating memory requirements
- Timing/profiling (`time`, cProfile, SnakeViz)
- **Multi threading vs. single threading**
- **CPU → GPU (`cupy` as `numpy` GPU drop-in replacement → 24x faster)**

Picking up where we left

10 million documents

- Each document *i* has a 128-dim. feature vector $a_{i,*}$ (e.g., LDA topics).
- Want to calculate all cosine similarities between documents *i* and *j*:
  $$s_{i,j} = \frac{a_{i,*} \cdot a_{j,*}}{||a_{i,*}|| \, ||a_{j,*}||}$$
  (naively corresponds to matrix multiplication of 1e7 x 128 matrix with its transpose)

Feature matrix (1e7 x 128):

$$(a_{i,j}) = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,128} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,128} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,128} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{10^7,1} & a_{10^7,2} & a_{10^7,3} & \cdots & a_{10^7,128} \end{bmatrix}$$

Similarity matrix (1e7 x 1e7):

$$(s_{i,j}) = (a_{i,j})(a_{i,j})^T$$

Picking up where we left

Document vector (1 x 128):

$$a_{i,*} = [a_{i,1} \quad a_{i,2} \quad a_{i,3} \quad \cdots \quad a_{i,128}]$$

→ size = 1e7 * 8 byte = **80 MB**

Feature matrix (1e7 x 128):

$$(a_{i,j}) = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,128} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,128} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,128} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{10^7,1} & a_{10^7,2} & a_{10^7,3} & \cdots & a_{10^7,128} \end{bmatrix}$$

→ size = 1e7 * 128 * 8 byte = **10.2 GB**

Similarity matrix (1e7 x 1e7):

$$(s_{i,j}) = (a_{i,j})(a_{i,j})^T$$

→ size = 1e7 * 128 * 8 byte = 800 TB

Picking up where we left

Out of memory on your computer?

Possible solutions?

- Reduce **number of documents**
- Move to a **cluster/PC with more RAM**
    - Every ETH student has access to Euler
- Use **sparse vectors**/matrices

How about making things faster?

- Multithreading / multiprocessing
- GPUs (great for linear algebra)

We'll try a few of these things now.

Picking up where we left

Using a CPU vs GPU on Euler (GPU only available to shareholders)

CPU only

```
#!/bin/bash
#SBATCH -n 1
#SBATCH --cpus-per-task=1
#SBATCH --time=4:00:00
#SBATCH --mem-per-cpu=24576
#SBATCH --mail-type=ALL
python matrix_math_test.py
```

`$ sbatch …`

`$ myjobs / seff` ***`jobid`***

Runtime: 120s

GPU (numpy -> cupy):

```
#!/bin/bash
#SBATCH --gpus=1
#SBATCH --gres=gpumem:22G
#SBATCH --time=4:00:00
#SBATCH --mail-type=ALL
module load gcc/8.2.0 python_gpu && python
matrix_math_test_gpu.py
```

`$ sbatch …`

`$ myjobs / seff` ***`jobid`***

Runtime: 5s **(24 x faster)**

Tentative Outline

For the remaining lecture

- **Shapes and forms of data**
- **APIs (and scraping)**
  - APIs (HTTP GET and HTTP POST)
  - Proxies vs. VPN
  - Parsing
- **Hands-on**
  - (Multi-threaded) API requests
  - OLS regression

Data

Some examples

	UNCOMPRESSED	COMPRESSED
TEXT	*.xml *.css *.md *.json(l) *.sql *.txt *.csv *.html *.tsv *.py	General purpose compression formats *.gz *.zip
BINARY	*.pkl *.tif *.png *.wav	*.mp3 *.mp3 (lossy) *.flac

Data

Guideline: Stick to human-readable and open source formats

- Flat/tabular data: `*.csv`
  - the open source and universal standard
  - databases are optimized for ingestion
- Hierarchical data: `*.json/jsonl`
- Arbitrary data structures: `*.pkl`
  - only use as last resort / if there is no other option

Use the integrated modules (`import json`, `import csv`) to read and write files. Do not escape/delimit manually … you'll almost certainly mess up.

Character encoding: use **UTF-8** whenever possible, it is the de-facto standard and works well internationally.

Data

CSV Sample [1]

```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""","",4900.00
1999,Chevy,"Venture ""Extended Edition, Very Large""","",5000.00
1996,Jeep,Grand Cherokee,"MUST SELL!
air, moon roof, loaded",4799.00
2000,Škoda,Fabia,"Good condition, very economic.”,1300.00
```

Linters, e.g., Rainbow CSV in Visual Studio Code often are confused by newlines within a cell.

Editors/viewers capable of dealing with larger files:

- Easy CSV Editor (macOS, commercial, ~10 CHF, trashy name but powerful)
- Tad Viewer (cross-platform, open source, can pivot)
- Modern CSV (cross-platform, freemium)

[1] adapted from https://en.wikipedia.org/wiki/Comma-separated_values, accessed 13 March 2024

Data

Sample JSON [1]

```
{
  "first_name": “Ade",
  "last_name": "Smith",
  "is_alive": true,
  "age": 27,
  "address": {
    "street_address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postal_code": "10021-3100"
  },
  "phone_numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ]
}
```

[1] https://en.wikipedia.org/wiki/JSON, accessed 13 March 2024

- Optimal for hierarchical data
- Heavily used for APIs
- Similar to a python dictionary
- Always uses double quotes for strings
- \\n for newline in string
  - First \ is for escaping
- Can introduce bloat to large corpora due to keyword repetition

APIs

Formalized way of exchanging data → Automation :)

Mostly HTTP GET and POST APIs:

HTTP GET

- query parameters in **URL**
- **header** for additional information (e.g., authentication for non-public APIs)
- length of URL is limited to 2048 characters (limiting when there are many query parameters)
- can only be use to **request** data

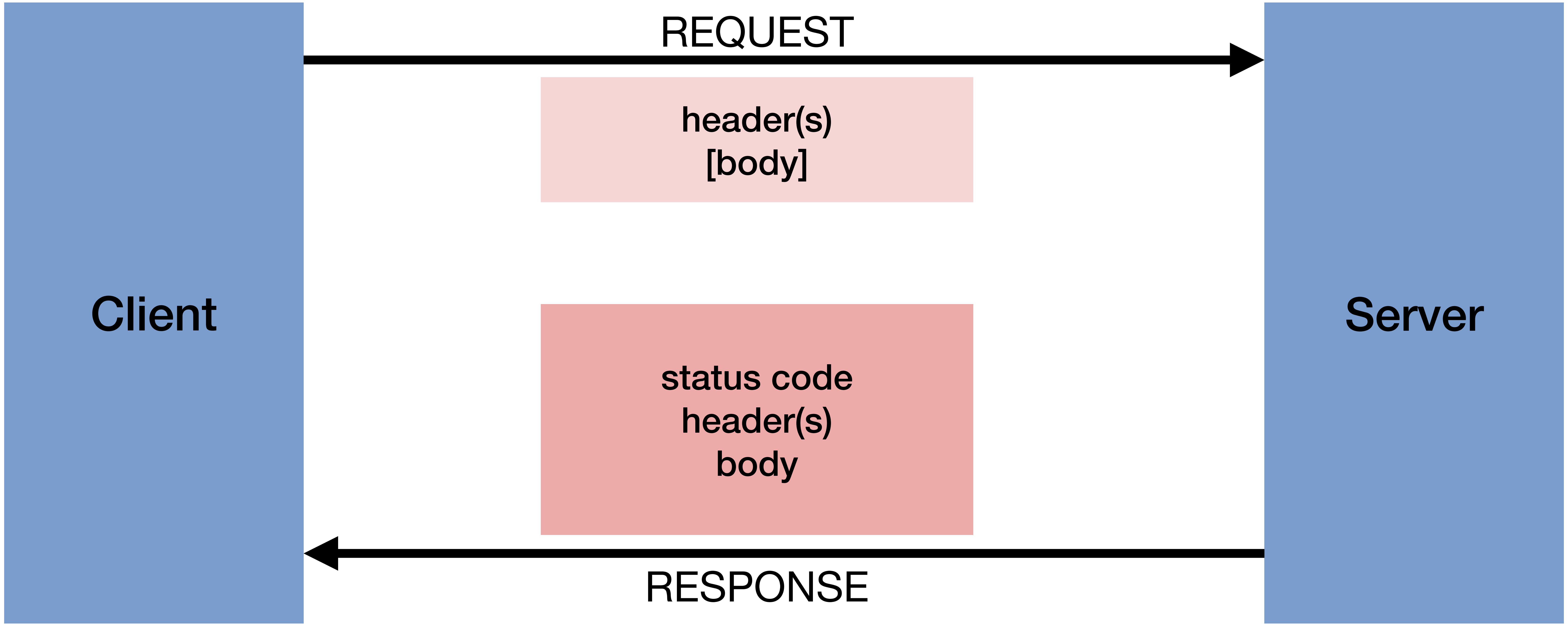
HTTP POST

- query parameters in **body**/payload
- **header** for additional information (e.g., authentication for non-public APIs)
- secure if HTTPS/TLS is used
- can be used to **request** and **send** data

Both methods return a HTTP status code, header(s), and normally also body.

APIs

Formalized way of exchanging data → Automation :)



APIs and Scraping

Appropriate tools

Command line: `curl`

Python: `import requests`

Web scraping is not inherently different from using APIs. Your browser is effectively doing the same as you in the command line. But: JavaScript and other interactive elements might require you to go beyond simple requests and opt for something like **selenium**.

Proxies and VPNs can mask your real IP address; mostly used to change client geolocation or rotate through IPs. Proxies normally better suited for scraping (less overhead, faster rotation).

For **parsing**, `BeautifulSoup` mostly is sufficient. For more complicated cases, you can go for `xml.etree.ElementTree`.

Hands-on

1. Create a virtual environment

```
$ pip install pipenv
$ mkdir lecture_04
$ cd lecture_04
$ pipenv install requests
  ipykernel matplotlib
  statsmodels tenacity
```

2. Open this lecture's jupyter notebook so that we can:

- execute an API request and examine its response
- perform an OLS regression in python
- optimize an I/O bound process (vs CPU bound in last session)
- increase scraping robustness via `@retry` decorator

End-of-Lecture Survey

ETH Edu App

Web app

https://eduapp-app1.ethz.ch/

iOS

Android