Visual Analytics – Assignment 2 – Part 1

Claudio Maggioni

Indexing

The first step of indexing is to convert the given CSV dataset (stored in `data/restaurants.csv`) into a JSON-lines file which can be directly used as the HTTP request body of Elasticsearch document insertion requests.

The conversion is performed by the script `./convert.sh`. The converted file is stored in `data/restaurants.jsonl`.

The gist of the conversion script is the following invocation of the *jq* tool:

```
jq -s --raw-input --raw-output \
  'split("\n") | .[1:-1] | map(split(",")) |
    map({
      "id": .[0],
      "name": .[1],
      "city": .[2],
      "location": {
        "lon": .[8] | sub("^\"\\["; "") | sub("\\s*"; "") | tonumber,
        "lat": .[9] | sub("\\]\"$"; "") | sub("\\s*"; "") | tonumber,
      },
      "averageCostForTwo": .[3],
      "aggregateRating": .[4],
      "ratingText": .[5],
      "votes": .[6],
      "date": .[7]
    })' "$input"
```

Here the CSV file is read as raw text, splitted into lines, has its first and last line discarded (as they are respectively the CSV header and a terminating blank line), splitted into columns by the `,` (comma) delimiter character, and each line is converted into a JSON object by *jq*. Note that *jq* is invoked in `slurp` mode so that the output is elaborated in one go.

Since location coordinates strings, represented in the csv as:

`"[{longitude}, {latitude}]"`

(with `{longitude}` and `{latitude}` being two JSON formatted `float`s), the comma split performed by *jq* divides the line in two pieces. I exploit this side effect by simply removing the spurious non-numeric characters (like `[]"` and space), converting the obtained strings into `float`s and storing them in the `lon` and `lat` properties of `location`.

After the conversion, the JSON-lines dataset is uploaded as an *Elasticsearch* index named `restaurants` by the script `upload.sh`. The script assumes *Elasticsearch* is deployed locally, uses HTTPS authentication and has HTTP basic authentication turned on. Installation parameters for my machine are hardcoded in variables at the start of the script and may be adapted to the local installation to run it.

The upload script, in order:

- Tries to `DELETE` (ignoring failures, e.g. if the index does not exist) and `POST`s the `/restaurants` index, which will be used to store the documents.

- Field mappings are `POST`ed at the URI `/restaurants/_mappings/`. Mappings are defined in the `mappings.json` file.
- The lines of the dataset are read one-by-one, and then the correspoding document is `POST`ed at the URI `/restaurants/_doc/{id}` where `{id}` is the value of the `id` field for the document/line.

The mappings map the `id` field to type `long`, all other numeric fields to type `float`, the `location` field to type `geo_point`, and the `date` field to type `date` by using non-strict ISO 8601 with optional time as a parsing format. All string fields are stored as type `text`, while also defining a `.keyword` alias for each to allow exact match queries on each field.