Università
della
Svizzera
italiana

Faculty
of Informatics

Student: Edoardo Riggio

Hotel Search

Final Report

Contents

1. Abstract

In this project I have to create a search engine for hotels. These hotels must be taken from at least two different countries and from several different websites – i.e. more than a couple. After having scraped the hotels using Scrapy, I have to index them using Solr and display them to the user via a graphical user interface. Moreover, the top 20 results must be visualized onto a map.

The project is structured in four main parts: Crawling, Indexing, Visualizing and User Evaluation. For each of the parts – except User Evaluation – I have created a repository on GitHub.

2. Crawling

The fist part of the project consisted in crawling several websites in order to get data on hotels. In order to do so I've only used Scrapy. I've crawled hotels in Switzerland and Italy from three different websites. These websites are:

1. Tripadvisor
2. Lonely Planet
3. MySwitzerland

The crawled data was then structured as follows:

Element	**Type**	**Description**
Source	String	Where is the hotel from – tripadvisor | lonely | myswitzerland
Name	String	The name of the hotel
URL	String	The url of the hotel detail page
Address	String	The physical address of the hotel
Coordinates	String	The coordinates of the hotel in the format *lat,lon*
Phone_Number	String	The phone number of the hotel
Description	String	The description of the hotel
Rating.Score	String	The rating of the hotel out of 4
Rating.N_Ratings	String	The number of people who rated the hotel

The coordinates where the only element that was not taken from the websites – being maps rendered dynamically. I have used a free and open-source API called *nominatim* in order to convert all of the addresses into coordinates. The only problem with this method is that not all addresses are interpreted correctly by the API, thus some hotels will have *null* values for the coordinates.

The reason I didn't choose Booking.com – as it was suggested in the project pdf – was because I had a weird bug with the spider where it generated multiple copies of already scraped hotels. For this reason I decided to use MySwitzerland instead.

All of the data obtained from the spiders has been saved in three different JSON files – one for each spider – inside of the crawler repository. From these three websites I've obtained 10'020 hotels. Of them, 6'529 hotel are in Switzerland, and 3'491 are in Italy.

2.1. Tripadvisor

In order to scrape hotels from Tripadvisor, I started to scrape from

```
https://www.tripadvisor.com/Hotels-g187768-Italy-Hotels.html
```

And

https://www.tripadvisor.com/Hotels-g188045-Switzerland-Hotels.html

Rather than from https://www.tripadvisor.com. This is because those two links directly have a list of the hotels in the respective country, thus there is no need for further requests in order to navigate to those pages.

This website was simple to scrape, and the data was displayed in a structured way. The only problem I had with this website was how to move to the next page. To do so I had to send a POST request to Tripadvisor servers which had to contain a user-agent – without which I couldn't get the actual next page, and a parameter called offset. This parameter needed to be set to a multiple of 25 – 25 for page 2, 50 for page 3...

2.2. Lonely Planet

In order to scrape this website – like in the case of Tripadvisor – I had to start from

https://www.lonelyplanet.com/italy/hotels?page=1&subtypes=Hotel

And

https://www.lonelyplanet.com/switzerland/hotels?page=1&subtypes=Hotel

Rather than from https://www.lonelyplanet.com. This website was easy to scrape and well structured, thus no problems were encountered.

2.3. MySwitzerland

In order to scrape this website I started from

https://www.myswitzerland.com/en-ch/accommodations/hotel-search/

Rather than https://www.myswitzerland.com. As for the scraping of this website, I didn't encounter any problem, thus no particular workaround or hack was needed.

3. Indexing

In order to index the data, I've used Solr. In Solr the first thing I did was to modify the default *managed-schema* contained inside of the *config* folder of the *_default* schema. This was done in order to let Solr know how to interpret the data passed to it. The data is divided by Solr in the following fields:

Element	**Type**
Source	string
Name	string
URL	string
Address	string
Coordinates	location
Phone_Number	string
Description	string
Rating.Score	pdouble
Rating.N_Ratings	string
text	text_general

The coordinates of each hotel are saved as a *location*. By doing so, I am able to make requests to Solr based on the distance between the place the user has searched for and the hotels.

Moreover, as I've said when talking about crawling, not all locations will have coordinates – since I'm relying on an external API. For this reason I've decided to add a field called *_text_*, which matches text from both the *url* field, as well as the *address* field. In this way, in the case that a hotel does not have coordinates, it will be matched based on whether the query is found inside either the *url* or *address* fields.

The reason why I've chosen to match the query to the *url* field, is that the urls taken from all three sources – Tripadvisor, Lonely Planet and MySwitzerland – most of the time contain the place where the hotel is located in.

3.1. Query

In order to retrieve the most relevant data from the Solr database, I've used the following query – which uses Solr spacial search:

```
d=5&fq=%7B!geofilt%7D&pt={lat}%2C{lon}&q={query}&rows=25&sfield=coordinates
 &sort=geodist()%20asc
```

Breaking down this long query, we can see the following parameters being passed to Solr:

- **fq=%7B!geofilt%7D → fq={!geofilt}**

  This sets the filtering method of the query to be the Solr geofiltering function – which takes *sfield*, *d* and *pt* as parameters.

- **d=5**

  This sets the distance of the hotel coordinates to be in a 5km radius from the center – which is defined later in the query.

- **pt={lat}%2C{lon} → pt={lat},{lon}**

  This sets the center point for the filtering to be the one defined by the coordinates *lat* and *lon*.

- **sfield=coordinates**

  This tells Solr that the field of type *location* is the *coordinates* field.

- **q={query}**

  This sets the query to be the name of the place being searched for. This becomes very useful in the case that the coordinates of a hotel are not present.

- **sort=geodist()%20asc → sort=geodist() asc**

  This sets the sorting function to be *geodist()* – which will sort all of the data based on their distance to *pt*. The value *asc* indicates that the sorted data needs to be in ascending order of distance.