WEM : Web Mining
Laboratoire n°1
Crawling, indexation et recherche de pages Web

03.03.2023

1. Introduction

Ce laboratoire a pour but d'implémenter un web crawler parcourant une collection de pages web de votre choix, puis d'utiliser *ElasticSearch*[1] pour l'indexation et la recherche.

Les points étudiés dans ce laboratoire seront :

- Crawling de pages web en Python (en utilisant la librairie *Scrapy*[2])
- Construction d'un index avec *ElasticSearch*
- Implémentation d'une fonction de recherche

1.1 Organisation

- Ce laboratoire doit être réalisé par groupe de 3 étudiants au maximum.

1.2 Rendu

Rendre un zip contenant :

- Le code source de votre implémentation.
- Un mini-rapport rapport dans lequel vous discuterez du fonctionnement de votre programme, de vos choix d'implémentation et répondrez aux questions posées dans la partie *2.4 Questions théoriques*.

1.3 Date de rendu

- Le dimanche **19.03.2022** à **23h59** au plus tard.

1.4 Environnement de travail

- Vous devez avoir Python 3.9 ou plus récent d'installé.
- Il peut vous être utile de créer un environnement virtuel.

1.5 Fichiers fournis

- *requirements.txt* : contient la liste des dépendances python. Pour les installer :
  ```
  > pip install -r "requirements.txt"
  ```

- *docker-compose.yml* : Permet de lancer un conteneur ElasticSearch et Kibana (pour la visualisation, accessible sur http://localhost:5601/)

[1] https://www.elastic.co/fr/elasticsearch/
[2] https://scrapy.org/

2. Travail demandé

Le laboratoire à réaliser en *Python* se déroulera en quatre parties. Dans un premier temps, vous crawlerez un site web de votre choix à l'aide de la librairie *Scrapy*. Dans la deuxième partie, on va indexer nos données à l'aide d'*ElasticSearch*. Ensuite on utilisera l'index créé dans la seconde partie pour mettre en place une fonction de recherche. La dernière partie consiste à répondre quelques questions théoriques.

2.1 Crawler

Dans cette première partie, on vous demande de choisir un site web de votre choix et d'en extraire les données qui vous semblent pertinentes, dans l'idée d'indexer ultérieurement les champs associés pour tous les éléments principaux composant une page web. Par exemple, comme le montre la Fig. 3, sur une page HTML certains éléments peuvent être mis en avant.

Figure 1 – Sur certaines pages web, une catégorisation du contenu est proposée

La grande majorité des pages HTML fournissent également des données structurées afin de mieux comprendre le contenu de la page. On vous demande d'adapter votre crawler afin qui récupère ces données structurées (normes schema.org ou Open Graph à choix).

```
<script type="application/ld+json">
{
    "@context":"https://schema.org",
    "@type":"Article",
    "name":"Haute Ecole d'ingénierie et de gestion du canton de Vaud",
    "description":"La Haute École d'Ingénierie et de Gestion du Canton de Vaud a été créée  à la suite du regroupement
    de l'École d'ingénieurs du canton de Vaud (EIVD) et de la Haute école de gestion du canton de Vaud (HEG-VD), le
    1er août 2004. Elle propose différentes formations intégrées au système de Bologne : Bachelors of Science, Masters
    of Science et plusieurs programmes exécutifs en formation continue.",
    "author":{
        "@type":"Organization",
        "name":"Contributeurs aux projets de Wikimédia"
    }
    "datePublished":"2006-07-27T00:09:58Z",
    "dateModified":"2020-01-19T22:52:19Z",
    "image":"https://upload.wikimedia.org/wikipedia/commons/f/fe/Locator_Map_Kanton_Waadt.png"
}
</script>
```

Figure 2 –Exemple de données structurées au format JSON-LD

Veuillez préciser dans votre rapport quel(s) élément(s) de la page visitée vous avez décidé d'indexer et comment l'avez-vous réalisé.

L'utilisation de la librairie *Scrapy* repose principalement sur la classe *Spider* dont doit hériter votre crawler, vous devrez en particulier définir les attributs et méthodes suivants :

- name: identifie le Spider (doit être unique dans votre projet).
- start_urls: liste d'url que notre crawler va utiliser comme point de départ.
- parse(): permet de traiter et d'analyser la page que l'on visite en extrayant les données et de trouver également de nouvelles URL à suivre à partir de cette page. Ces nouvelles URL sont primordiales pour que notre Crawler puisse se « balader » sur le site à indexer.

```
import scrapy

class QuotesSpider(scrapy.Spider):
   name = "quotes"

   start_urls = [
      'https://quotes.toscrape.com/page/1/',
      'https://quotes.toscrape.com/page/2/',
   ]

   def parse(self, response):
      print(response.css('title::text').get())
```

Quelques remarques/précisions supplémentaires

- Veuillez noter l'existence des Selectors[3] de la librairie Scrapy qui vous aideront certainement à récupérer du contenu spécifique dans une page au format HTML.
- Attention à ne pas vous faire « blacklister »
- Regardez la page Scrapy Tutorial pour un exemple de fonctionnement.

2.2 Indexation

Dans cette deuxième partie, on souhaite à présent indexer nos pages visitées par le crawler de la première partie en utilisant *ElasticSearch*.

On va donc créer un seul index sur *ElasticSearch* qui contiendra tous nos documents.

Il vous faudra configurer précisément les champs (ainsi que leur type) que vous souhaitez indexer et également adapter votre programme afin d'indexer les éléments importants que vous avez décidé de stocker dans la première partie.

Quelques remarques/précisions supplémentaires

- Veuillez noter l'existence de la librairie Python *Elasticsearch DSL*[4] qui vous aidera certainement à écrire et à exécuter des requêtes *Elasticsearch*. Elle est construite par-dessus le client officiel d'*ElasticSearch* pour Python *elasticsearch-py*[5].

[3] https://docs.scrapy.org/en/latest/topics/selectors.html
[4] https://github.com/elastic/elasticsearch-dsl-py
[5] https://github.com/elastic/elasticsearch-py

2.3 Recherche

On souhaite à présent ajouter une fonctionnalité de recherche dans notre index. Il vous faudra créer un second programme permettant d'effectuer une recherche dans votre index et d'afficher les résultats obtenus ainsi que leur score (pertinence).

Veuillez mettre en place une recherche qui privilégiera la recherche dans le titre d'une page et dans les champs « mis en avant » (cf. Fig. 1 et 2) par rapport à tout le contenu de la page. Certains champs importants devront donc être « boostés » par rapport à d'autres champs moins importants.

Dans votre rapport, détaillez la syntaxe utilisée par ElasticSearch pour les faire des querys simples. Comment fait-on pour rechercher uniquement dans certains champs ? Comment fait-on pour « booster » certains champs par rapport à d'autres.

2.4 Questions théoriques

2.4.1 Veuillez expliquer quelle(s) stratégie(s) il faut adopter pour indexer des pages dans plusieurs langues (chaque page est composée d'une seule langue, mais le corpus comporte des pages dans plusieurs langues). A quoi faut-il faire particulièrement attention, quels sont les avantages et inconvénients de celle(s)-ci ? Veuillez expliquer la(les) démarche(s) que vous proposez.

2.4.2 *ElasticSearch* permet par défaut de faire de la recherche floue (fuzzy query). Veuillez expliquer de quoi il s'agit et comment *ElasticSearch* l'a implémenté. Certains prénoms peuvent avoir beaucoup de variation orthographiques (par exemple Caitlin : Caitilin, Caitlen, Caitlinn, Caitlyn, Caitlyne, Caitlynn, Cateline, Catelinn, Catelyn, Catelynn, Catlain, Catlin, Catline, Catlyn, Catlynn, Kaitlin, Kaitlinn, Kaitlyn, Kaitlynn, Katelin, Katelyn, Katelynn, etc). Est-il possible d'utiliser, tout en gardant une bonne performance, la recherche floue mise à disposition par *ElasticSearch* pour faire une recherche prenant en compte de telles variations ? Sinon quelle(s) alternative(s) voyez-vous, veuillez justifier votre réponse.