Regards Citoyens

Formation Data Scraping et Dataviz

Brice Person

OpenData Burkina Faso – Ouagadougou - 24/26 novembre 2015

Préambule

Lorsque une ligne est précédée de « $ », il s'agit généralement d'une commande à exécuter dans votre terminal.

Rappel : Utilisation basique de la ligne de commande

Quelques commandes de base :

cd, ls, mkdir, cat, less, grep, wc, rm, find, ...

cd ~/ le tilde indique le home de l'utilisateur courant.

La touche « tabulation » du clavier permet d'auto-compléter les termes saisis.

Pour connaître l'utilisation d'une commande en particulier, il suffit de taper :

```
$ man nom_de_la_commande
```

Ex :

```
$ man ls
```

Travaux Pratiques : Scraping de sites web

Cela coule un peu de source mais la première chose à vérifier lorsque l'on souhaite récupérer des données d'un site web est que leur ré-utilisation soit autorisée et cela dans des termes compatibles avec notre projet.

Ensuite si les données ne sont pas disponibles en téléchargement direct, il faudra prévoir un mécanisme permettant de ne pas perturber le site d'origine lors de leur extraction.

Les Jurisprudences Burkinabées du site Juricaf.org

Le site Juricaf.org est une base de données de jurisprudences francophones couplée à un moteur de recherche performant mis à disposition par l'AHJUCAF.

Cette association permet la ré-utilisation de la plupart des jurisprudences des pays africains dans les termes de la licence ODbL V1, consultable à cette adresse :
http://www.juricaf.org/documentation/licence_odbl.php

C'est de l'Open Data. Dès lors que l'on respecte les termes de cette licence nous sommes couverts au niveau juridique. Malheureusement, il n'est pas fourni de moyen de télécharger directement un corpus de données, nous allons donc y remédier.

Nous ré-utiliserons notre environnement python créé lors du 1er TP :

```
$ source ~/pythenv/scraping/bin/activate
```

Création de notre nouveau projet que nous intitulerons « juriscrapper » :

```
$ mkdir -p ~/dev/juriscrapper
$ cd ~/dev/juriscrapper
```

On observe que sur la page d'accueil du site (http://www.juricaf.org) sont fournis des liens vers les collections de jurisprudences des différents pays ou institutions. Un clic sur l'un d'eux nous amène dans le moteur de recherche qui nous énumère les résultats correspondants avec une pagination.

Pour le Burkina Faso, au moment où j'écris ces lignes, on obtient 283 résultats dont 240 concernent la Cour de Cassation et 43 la Cour Constitutionnelle.

http://www.juricaf.org/recherche/+/facet_pays%3ABurkina_Faso (page que nous appellerons page de résultats initiale)

Ce qui nous intéresse dans un premier temps c'est d'obtenir tous les liens vers ces décisions. Il y en a 10 par page et passer d'une page à l'autre modifie notre URL avec un paramètre « page » différent.

http://www.juricaf.org/recherche/+/facet_pays%3ABurkina_Faso?page=2

Un clic sur « fin >> » en bas de page nous apprend qu'il y a 29 pages

http://www.juricaf.org/recherche/+/facet_pays%3ABurkina_Faso?page=29

Parser la page de résultats initiale est donc notre première étape et il nous suffit de récupérer la valeur du paramètre « page » de l'URL du lien intitulé « fin >> » pour pouvoir générer les autres liens contenant les résultats puisque c'est de l'incrémental.

Notre objectif est que notre parseur final soit réutilisable quelle que soit la page de résultats initiale. Cela pourra servir à d'autres pour récupérer des collections placées également sous licence ODbL.

Nous utiliserons le programme « wget » pour les téléchargements et python avec BeautifulSoup pour le parsing.

Commençons par télécharger notre page de résultats initiale :

```
$ wget http://www.juricaf.org/recherche/+/facet_pays%3ABurkina_Faso -O init.html -q
```

Le paramètre -O nous permet d'enregistrer cette page dans un fichier dont on aura choisi le nom, ici « init.html ». Le paramètre -q est utilisé pour rendre wget muet.

Créez un fichier python intitulé « getResults.py » avec ce contenu :

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
import sys, urlparse
from bs4 import BeautifulSoup

soup = BeautifulSoup(open(sys.argv[1]), "html5lib")

for link in soup.find_all('a'):
 if link.get_text() == 'Fin >>':
  path = urlparse.urlparse(link.get('href'))
  params = urlparse.parse_qs(urlparse.urlparse(link.get('href')).query)
  nbpages = int(params['page'][0])

prefix = "http://www.juricaf.org"+path.path

for i in range(nbpages):
 print(prefix+'?page='+str(i+1))
```

Vous remarquerez que pour utiliser BeautifulSoup, cette fois nous utilisons le parseur « html5lib » plutôt que « lxml-xml » car dans notre cas particulier celui-ci ne fonctionne pas comme attendu. En effet, celui-ci lorsqu'il rencontre les « < » et « > » contenus dans le texte des liens qui nous intéressent les traite comme s'il avait affaire à du HTML mal

formé et essaye donc de réparer cela, ce qui conduit à la suppression de l'information qui nous intéresse.

Pour pallier ce type de problème pensez donc toujours à vérifier si le parseur choisi pour utiliser BeautifulSoup correspond à vos attentes avec un « print(soup.prettify()) » avant toute autre manipulation (voir documentation http://www.crummy.com/software/BeautifulSoup/bs4/doc/).

```
$ python getResults.py init.html
```

Nous avons notre liste de liens

On l'enregistre dans un fichier nommé « results.txt »

```
$ python getResults.py init.html > results.txt
```

Notre deuxième étape consiste à récupérer les liens vers les décisions. Il nous faut donc télécharger et stocker les pages dont les liens sont contenus dans notre fichier « results.txt ».

```
$ mkdir -p ./results
$ cd ./results
```

Ne lancez pas encore cette commande :

```
$ wget -i ../results.txt --wait 1
```

Le paramètre « -i » nous permet d'indiquer un fichier texte contenant une liste de liens.

Nous avons parlé tout à l'heure de prévoir un mécanisme permettant de ne pas perturber le site duquel on souhaite extraire des données. Le paramètre « --wait 1 » nous permet d'indiquer que nous souhaitons espacer d'une seconde le téléchargement de chacun de nos liens.

Vous rencontrerez parfois certains sites qui, bien qu'autorisant la ré-utilisation de leurs données, sont protégés contre les robots comme le nôtre, aussi voici quelques astuces :

```
$ wget -i ../results.txt --wait 1 --random-wait
```

« --random-wait » permet d'indiquer que nous souhaitons espacer nos téléchargements de manière aléatoire. Concrètement, wget multipliera la valeur de notre paramètre « --wait » par 2 en tant qu'intervalle maximale et téléchargera les différents liens dans un délai aléatoire compris entre 0 et « --wait » * 2. Donc dans le cas présent entre 0 et 2 secondes.

Enfin, il est possible de faire croire au site qu'il a affaire à un utilisateur normal en ajoutant quelques headers :

```
$ wget -i ../results.txt --wait 1 --random-wait -q -U "Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:40.0) Gecko/20100101 Firefox/40.0" --header="Accept-Language: en-us,en;q=0.5" --header="Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" --header="Connection: keep-alive" –referer="/"
```

Ici nous nous faisons passer pour Firefox en version 40.0 sur une distribution GNU Linux Fedora. Vous pouvez lancer cette commande.

En général, on utilise systématiquement ces artifices car sans cela nous risquerions un blacklistage de notre adresse IP, ce qui compliquerait notre projet.

Notre dossier « results » contient désormais toutes les pages de résultats. Notre troisième étape consiste à extraire de ces pages les liens vers les pages des décisions.

Observons donc à nouveau le contenu de notre fichier « init.html » qui contient notre page de résultats initiale dont la structure est identique à celles de notre dossier « results ». Il nous faut trouver un point commun pour distinguer les liens qui nous intéressent. On constate que les liens vers les décisions commencent tous par « /arret ». Ceci est suffisant, mais on aurait aussi pu collecter tous les liens dans la « div » de classe « resultatcols » qui est unique.

```
$ cd ../
```

Créez un fichier python intitulé « getLinks.py » avec ce contenu :

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
import sys, csv, re
from bs4 import BeautifulSoup

soup = BeautifulSoup(open(sys.argv[1]), "html5lib")

prefix = 'http://www.juricaf.org'

for link in soup.find_all('a'):
 if link.get('href').startswith('/arret'):
  print(prefix+link.get('href'))
```

Tester le script :

```
$ python getLinks.py init.html
```

Il ne nous reste plus qu'à appliquer ce script sur nos pages de résultats pour obtenir la liste des liens vers les décisions.

```
$ touch files.txt
```

Nous utiliserons une simple boucle « bash » avec « find » et orienterons la sortie vers un fichier nommé « files.txt » :

```
$ for file in $(find ./results/ -name "*page*") ; do python getLinks.py "$file" >> files.txt ; done
```

Notez l'utilisation de « >> » qui permet d'ajouter le résultat des itérations à la fin du fichier « files.txt ». Faites donc attention à ce que ce fichier soit bien vide avant de lancer cette commande.

Notre fichier « files.txt » contient à présent l'intégralité des liens vers les décisions. Nous allons pouvoir les télécharger pour les stocker dans un dossier « files ».

```
$ mkdir -p ./files
$ cd ./files
```

```
$ wget -i ../files.txt --wait 1 --random-wait -q -U "Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:40.0) Gecko/20100101 Firefox/40.0" --header="Accept-Language: en-us,en;q=0.5" --header="Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" --header="Connection: keep-alive" --referer="/"
```

```
$ cd ../
```

On observe maintenant les codes sources des décisions contenues dans notre dossier « files » pour déterminer les variables extractibles. Puis nous allons les tester pour avoir une vue d'ensemble de leur présence ou non sur toutes les décisions.

- Le titre est contenu dans la balise « title »
- Le numéro d'affaire est contenu dans le titre. Ex : 2003 cc 1 (JB)
- Le texte est contenu dans une balise « span » ayant une propriété « itemprop » avec pour valeur « articleBody »
- Il y a des balises meta à évaluer

Les balises « meta » les plus intéressantes sont préfixées DC pour Dublin Core, les autres étant une redite pour les réseaux sociaux.

Nous allons créer un parseur qui traitera les décisions une par une. Celui-ci générera une ligne au format CSV par décision. Il est donc crucial de connaître à l'avance quelles seront les colonnes présentes dans le jeu de données final pour prévoir un comportement adapté lorsqu'une information sera manquante.

Nous pourrions nous contenter de recenser à la main les variables que nous souhaitons récupérer pour faire un dictionnaire python en initialisant des valeurs vides qui seraient renseignées ou non lors du parsing individuel d'une décision. Ce serait en effet suffisant dans beaucoup de projets.

Cependant, dans le cas présent, il nous faut déterminer l'étendue des métas Dublin Core. Celles-ci sont-elles dynamiques ? Ou sont-elles systématiquement présentes ?

Maintenant que nous avons l'ensemble des décisions sur notre disque dur, nous allons pouvoir le déterminer :

Créez un fichier python intitulé « checkVars.py » avec ce contenu :

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
import sys, io, csv, re
from bs4 import BeautifulSoup

soup = BeautifulSoup(open(sys.argv[1]), "lxml-xml")

data = {}

# On teste les variables DC
for meta in soup.find_all('meta'):
 if str(meta.get('name')).startswith("DC"):
  data[meta.get('name')] = meta.get('name')

# Formatage CSV
output = io.BytesIO()
writer = csv.DictWriter(output, fieldnames=sorted(data.keys()), quoting=csv.QUOTE_NONNUMERIC)
writer.writerow({k:v.encode('utf8') for k,v in data.items()})
print(output.getvalue().strip())
```

On le lance ensuite sur toutes nos décisions :

```
$ for file in $(find ./files/ -name "*-*") ; do python checkVars.py "$file" >> check.csv ; done
```

En examinant le fichier « check.csv » résultant, on peut s'apercevoir que la plupart des metas DC sont disponibles sur chacune des décisions à l'exception de la meta « DC.references ». Nos métas DC sont donc dynamiques.

Comme nous souhaitons que notre parseur soit réutilisable pour d'autres collections que celle du Burkina, en conséquence nous allons devoir générer un header dynamique.

Nous extrairons tous les noms de colonnes possibles de « check.csv » avec un autre petit script par la suite.

Toujours dans notre « checkVars.py », nous ajoutons manuellement à notre dictionnaire « data » les clés des autres variables repérées précédemment en les renseignant par leurs noms.

```
data = {}
```

deviens :

```
data = {
 'title': 'title',
 'caseref': 'caseref',
 'content': 'content'
}
```

On vide le fichier « check.csv » :

```
$ >  check.csv
```

On relance le même processus :

```
$ for file in $(find ./files/ -name "*-*") ; do python checkVars.py "$file" >> check.csv ; done
```

Créez un fichier python intitulé « extractHeader.py » avec ce contenu :

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
import sys, io, csv

reader = csv.reader(open(sys.argv[1], 'r'))

data = {}

for row in reader:
 for val in row:
  data[val] = val

# Formatage CSV
output = io.BytesIO()
writer = csv.DictWriter(output, fieldnames=sorted(data.keys()), quoting=csv.QUOTE_NONNUMERIC)
writer.writerow({k:v.encode('utf8') for k,v in data.items()})
print(output.getvalue().strip())
```

On le lance sur « check.csv »

```
$ python extractHeader.py check.csv
```

Nous avons notre header, sauvegardons-le dans un fichier « header.csv »

```
$ python extractHeader.py check.csv > header.csv
```

Il ne reste plus qu'à créer le parseur de décisions.

Créez un fichier python intitulé «parser.py » avec ce contenu :

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
import sys, io, csv, re
from bs4 import BeautifulSoup

soup = BeautifulSoup(open(sys.argv[1]), "lxml-xml")

# Initialiser un dictionnaire basé sur header.csv pour recueillir les données
reader = csv.reader(open('header.csv', 'r'))
data = {}

for row in reader:
  for val in row:
    data[val] = ''

# Métas Dublin Core
for meta in soup.find_all('meta'):
  if str(meta.get('name')).startswith("DC"):
    data[meta.get('name')] = meta.get('content')

# Titre
data['title'] = soup.title.string

# Numéro d'affaire
data['caseref'] = data['title'].split(',')[-1].strip()

# Texte de la décision
texte = soup.find_all("span", itemprop="articleBody")[0].get_text()

# Suppression des espaces inutiles
texte = re.sub(' {2,}',' ', texte)
```

```
# Aérons un peu le texte
texte = re.sub('\n','\n\n', texte)

data['content'] = texte

# Formatage CSV
output = io.BytesIO()
writer = csv.DictWriter(output, fieldnames=sorted(data.keys()), quoting=csv.QUOTE_NONNUMERIC)
writer.writerow({k:v.encode('utf8') for k,v in data.items()})
print(output.getvalue().strip())
```

On le lance de la même manière que « checkVars.py » :

```
$ for file in $(find ./files/ -name "*-*") ; do python parser.py "$file" >> data.csv ; done
```

Et voilà, nous avons les données !

Pour que notre parseur soit réutilisable, nous n'avons plus qu'à récapituler les différentes commande exécutées dans un script shell en ajoutant une gestion par dossiers afin de pouvoir conserver les résultats de nos différentes requêtes.

Le seul script à modifier pour que ça fonctionne correctement est « parser.py » car sans cela celui-ci ne trouvera pas le fichier « header.csv »

Remplacer la ligne :

```
reader = csv.reader(open('header.csv', 'r'))
```

Par :

```
directory = re.sub(r"/.+", "", sys.argv[1])
reader = csv.reader(open(directory+'/header.csv', 'r'))
```

Créez un fichier bash intitulé «main.sh » avec ce contenu :

```
#!/bin/bash

## TELECHARGEMENT

# Faire un slug de l'url pour nommer le dossier qui contiendra son résultat
directory="$(echo -n $1 | sed -e 's/[^[:alnum:]]/-/g' | tr -s '-' | tr A-Z a-z)"

if [ -d "./$directory" ]; then
 rm -r "./$directory" ;
fi

mkdir -p "./$directory/results"
mkdir -p "./$directory/files"

# Télécharger la page de résultats initiale
wget $1 -O "./$directory/init.html" -q -U "Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:40.0)
Gecko/20100101 Firefox/40.0" --header="Accept-Language: en-us,en;q=0.5" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" --header="Connection: keep-
alive" --referer="/"

# Obtenir la liste des urls des pages de résultats
python getResults.py "./$directory/init.html" > "./$directory/results/results.txt"
```

```
# Télécharger les pages de résultats et les stocker dans le dossier "results"
cd "./$directory/results"
wget -i ./results.txt --wait 1 --random-wait -q -U "Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:40.0)
Gecko/20100101 Firefox/40.0" --header="Accept-Language: en-us,en;q=0.5" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" --header="Connection: keep-
alive" --referer="/"

# Obtenir la liste des urls des fichiers à télécharger
cd ../../
> "./$directory/files/files.txt"
for file in $(find "$directory/results/" -name "*page*") ; do python getLinks.py "$file" >> "./
$directory/files/files.txt" ; done

# Télécharger les fichiers pour les stocker dans le dossier "files"
cd "./$directory/files"
wget -i ./files.txt --wait 1 --random-wait -q -U "Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:40.0)
Gecko/20100101 Firefox/40.0" --header="Accept-Language: en-us,en;q=0.5" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" --header="Connection: keep-
alive" --referer="/"

## EXTRACTION DES DONNÉES

# Test des variables
cd ../../
> "./$directory/check.csv"
for file in $(find "$directory/files/" -name "*-*") ; do python checkVars.py "$file" >> "./
$directory/check.csv" ; done

# Détermination des colonnes
python extractHeader.py "./$directory/check.csv" > "./$directory/header.csv"
```

```
# Parser les fichiers et stocker les résultats dans data.csv
cat "./$directory/header.csv" > "./$directory/data.csv"
for file in $(find "$directory/files/" -name "*-*") ; do python parser.py "$file" >> "./
$directory/data.csv" ; done
```

Rendre « main.sh » exécutable :

```
$ chmod +x main.sh
```

Ensuite pour utiliser notre parseur, il nous suffira de lui donner en argument une URL de résultat du site Juricaf.org tel que :

```
$ ./main.sh http://www.juricaf.org/recherche/+/facet_pays%3AB%C3%A9nin
```

pour obtenir toutes les juriprudences du Bénin.

Une petite parenthèse, sachez également que toutes les décisions du site sont disponibles en XML en ajoutant simplement « /xml » à la fin de leur URL.

Exemple : http://www.juricaf.org/arret/BURKINAFASO-COURDECASSATION-20070601-2007CASS16JB/xml

Notez toutefois que le XML ainsi obtenu est moins complet que les pages elles-mêmes.