Dylan PARANT - Jonathan DEROUET - Julien Mastio

Web Scrapping

Introduction

L'objectif de ce TD était de scrapper des données et d'y effectuer différents traitements. Pour ce faire, plusieurs bibliothèques python ont été fournies. Ces bibliothèques sont au nombre de 3 et sont :

- BeautifulSoup
- JusText
- Boilerpipe Celles-ci ont un fonctionnement différent les unes par rapport aux autres.

Récupération des données

Dans ce TD, nous avons travaillé avec deux bibliothèques et non trois, car nous avons eu des soucis d'installation sur Boilerpipe.

Sous BeautifulText

Dans le cas de beautifulSoup, la récupération s'est faite à partir de toutes les balises '

'. Nous avons récupéré tout le contenu textuelles à l'interieur de celles-ci.

Ci-dessous un extrait du code permettant la récupération des lignes du fichier ainsi que la création d'un fichier contenant les éléments récupérés.

```
# opens file
currentFile =
    open(filename, 'r', encoding='UTF-8', errors="ignore").read()
soup = BeautifulSoup(currentFile, "html.parser")
# Gets number of line
lines = soup.find_all('p')
lineNumber = len(lines) - 1
# Counts number of char
carNumber = 0
newFileName = "Corpus_detourage/BS/" + f + ".txt"
newFile = open(newFileName, "w", encoding="UTF-8")
for line in lines:
    currentLine = '<p>'+str(line.getText())+'</p>'
    print(currentLine)
    carNumber += len(currentLine)
    newFile.write(currentLine)
newFile.close()
```

Sous JusText

Dans le cas de JusText, la récupération du contenu textuel de la page est automatique. Nous avons défini la langue par défaut à Anglais car nous étions forcés d'en définir une. Ci-dessous un extrait du code permettant la récupération des lignes du fichier ainsi que la création d'un fichier contenant les éléments récupérés.

```
response = open(filename, "r", encoding="UTF-8").read()
paragraphs = justext.justext(response, justext.get_stoplist("English"))
for paragraph in paragraphs:
if not paragraph.is_boilerplate:
   newFileName = "Corpus_detourage/JT/" + f + ".txt"
   newFile = open(newFileName, "a", encoding="UTF-8")
   newFile.write("<p>" + paragraph.text + "</p>\n")
   newFile.close()
```

Calcul effectués

Suite à la récupération des données, il nous fallait calculer différentes métriques: comme notamment le nombre de ligne et le nombre de caractères nous allons les voir en détails :

Moyenne des lignes et des caractère

Il s'agit de la moyenne des lignes et des caractères pour chaque fichier. Il n'y a pas de raison de plus détailler ce calcul.

Ecart Type des lignes et des fichiers

L'écart type correspond a la moyenne des écarts à la moyenne. Ci-dessous un exemple de calcul pour ce projet :

```
    # calculates the standart deviation of lines
    sum = 0
    for res in resultsArray:
        sum += (res[1] - moyLines)**2
    sum /= len(resultsArray)
    standartDeviationLines = sqrt(sum)
```

Tous les résultats calculés sont présents dans le dossier 'generatedDatas'.

Reconnaissance de langue

Avec l'utilisation de jusText, nous sommes forcés lors de notre scrapping d'attribuer une langue à notre fichier, dans l'exercice 1 nous avons choisi de mettre la langue anglais pour tous les fichiers.

Pour déterminer la langue nous avons utilisé la bibliothèque langid. Nous pouvons classifier un fichier en fonction de sa langue grace à la fonction classify de la bibliothèque.

```
lang = langid.classify(response)
paragraphs = justext.justext(response,
```

```
justext.get_stoplist(convertISO(lang[0])))
```

Il suffit alors de convertir le code ISO639-1 en langage naturel grâce à la fonction suivante :

```
def convertISO(ISOcode):
    switcher = {
        "fr": 'French',
        "en": 'English',
        "el": 'Greek',
        "pl": 'Polish',
        "ru": 'Russian',
        "zh": 'English'
    }
    return switcher.get(ISOcode, "Invalid ISO code")
```

Enfin, nous avons utilisé le fichier doc_lg.json pour avoir les vraies langues des fichiers html et nous avons reproduit la même démarche pour scrapper les données utiles.

Évaluation intrinsèque

Pour réaliser l'évaluation intrinsèque de nos résultats on évalue d'abord F-mesure, Rappel et précision pour tout les fichiers du répertoire qui nous intéresse (JT, BS, JT_langid, JT_trueLg, ...) :

```
    def compute_intrinsic(folder_path):
    rs = []
    file_list = os.listdir(folder_path)
    for f in file_list:
        file_name = folder_path + "/" + f
        golden_file_name = "../Corpus_detourage/clean/" + f.strip(".txt")
        result = clt.evaluate_file(file_name, golden_file_name)

        element = {
            "filename": f,
            "f-score": result["f-score"],
            "precision": result["precision"],
            "recall": result["recall"],
            "lang": json[f.strip(".txt")]
        }
        rs.append(element)
    return rs
```

Les résultats sont stockés dans une liste d'éléments que l'on pourra ensuite regrouper par langue ou par source. Une fois regroupé il ne reste plus qu'à faire la moyenne des mesures, le code n'est pas présenté dans ce rapport, si cela vous intéresse je vous invite à directement aller voir le dossier 'Exercice3' disponible sur Github.

Unfluff

Pour réaliser l'exercice 4 nous avons choisi d'utiliser uniquement la librairie Unfluff car plus simple à mettre en place.

Il suffit de créer un projet node et d'installer la dépendance :

```
npm install unfluff
```

Pour scrapper l'ensemble de nos fichiers on parcours le dossier html et on utilise la fonction extractor de la librairie sur l'ensemble des données de nos fichiers d'origines. Il ne reste plus qu'à sauvegarder le résultat du scrapping dans un nouveau dossier UF.

```
const testFolder = 'Corpus_detourage\\html';
const fs = require('fs');
const extractor = require('unfluff');

fs.readdir(testFolder, (err, files) => {
    files.forEach(file => {
        let data = fs.readFileSync(testFolder + '/' + file);
        let content = extractor(data);
        fs.writeFileSync('./uf/' + file, content.text, encoding="utf-8");
    });
});
```

Enfin, pour évaluer la pertinence de l'outil on réutilise le code python déjà présenté dans ce rapport.