PROJET MOTEURS/SEO

Projet 1 - Cartographgie du contenu dupliqué

BOSMA François (bosma_f)
EUTROPE Alexis (eutrop_a)
PERON Gwendal (peron_g)
PERARD Alexandre (perard_b)

Juillet 2017

Sommaire

Chapitre 1

Inputs et génération des k-grams

Bibliothèques utilisées : BeautifulSoup4, NLTK

1.1 Extraction du texte d'une page web

- Suppression des commentaires avec [Comment] de BeautifulSoup4
- Extraction des textes 'lisibles' avec BeautifulSoup4
- Suppression des balises <style> <script> <head> <title> et de leur contenu avec un filtre
- Suppression des séparateurs (espaces, \n, \r ...) avec un split puis join
- Suppression de la ponctuation ainsi que de certains caractères spéciaux (i.e. non-breakable spaces) avec un filtre
- Extraction des k-grams avec *NLTK*

1.2 Utilisation du script

- Il est demandé de rentrer la taille des shingles désirée afin d'ajuster la précision
- Il est ensuite demandé d'entrer une à une des urls, sous la forme http://example.com (http:// est nécessaire)
- Tada!

Chapitre 2

Calcul des taux de duplication

Une fois les liste des shingles correspondant aux urls en poche, il ne reste plus qu'à les comparer. Au préalable cependant nous devons définir les "sketchs", i.e sous-ensemble aléatoire d'une liste de shingles, pour des raisons de performances. Nous avons donc défini, de manière arbitraire à 200 et 0.25, deux variables globales nous permettant de définir la taille d'un sketch en fonction de la taille d'une liste de shingles. Avec ces valeurs, la taille du sketch de A sera:

- si la taille de A <= 200: la taille de A
- sinon si la taille de A * 0.25 < 200: 200
- sinon: A * 0.25

Enfin, nous comparons les listes de shingles et définissons les données de la matrice avec une valeur de similarité comprise entre 0 et 1. Pour ce faire nous calculons nombre_de_shingles_en_commun / nombre_total_de_shingle.

Chapitre 3

Génération des outputs

3.1 CSV

Pour le CSV, nous avons utilisé la lib *csv* de python qui permet de rapidement transcrire dans un fichier au format csv toute donnée compatible, qui était donc la méthode la plus adaptée dans notre cas car nous représentions notre matrice avec un tableau à deux dimensions.

3.2 PDF

Pour le PDF, nous avons utilisé une bibliothèque python open source nommée *ReportLab*, qui contient de très nombreuses fonctions utiles à la génération dynamique de PDF à partir de données directement en python. Cette bibliothèque permet la génération de documents paramétrables (marges etc) à partir d'une liste d'éléments, qui eux mêmes ont différentes propriétés de styling (alignement etc). Les éléments sont des classes dédiées à la représentation d'un certain type de donnée, nous avons en l'occurrence utilisé la classe Table qui permet de représenter un tableau à deux dimensions. Il existe d'autres classes pour générer des graphiques, charts, ou encore des illustrations vectorielles définies par du code. Nous avons donc représenté nos données sous forme d'une table, et colorié en rouge les taux de similarité $>= 0.8$.

Nous avons également choisi d'afficher au dessus de la matrice la liste des urls entrées par l'utilisateur en indiquant à quel numéro correspondent-elles. Par soucis de lecture nous avons tronqué les urls trop longues (> 50 caractères).

Chapitre 4

Packaging

Pour effectuer le rendu du projet nous avons utilisé *cxFreeze* (http://cx-freeze.readthedocs.io/en/latest/). Cet outil nous a permis de "packager" notre projet en incluant python et les bibliothèques utlisées ainsi que de créer un binaire.