Parseur de PDF

Antoine Jamelot, Université de Bretagne Sud
jamelot.e1500523@etud.univ-ubs.fr

Sofiane Ben Massaoud, Université de Bretagne Sud
ben-massaoud.e1803113@etud.univ-ubs.fr

Baptiste Colas, Université de Bretagne Sud
colas.e1800392@etud.univ-ubs.fr

Théau Huteau, Université de Bretagne Sud
huteau.e1803128@etud.univ-ubs.fr

4 mai 2021

1 Définition

Parser : (Anglicisme informatique) (Programmation) Parcourir le contenu d'un texte ou d'un fichier en l'analysant pour vérifier sa syntaxe ou en extraire des éléments (définition du Wictionnaire).

2 Abstract

Cet article a pour but de présenter notre production dans l'Unité d'Enseignement *Projet de développement*. Nous vous présenterons les méthodes utilisés et les raisons de ces choix, ainsi que nos résultats finaux sur un corpus de référence et une discussion sur ses résultats. Nous avons donc parsé les articles du corpus tout d'abord en convertisant l'article au format `txt` ou `xml` selon la commande rentrée en ligne de commande. Ensuite on normalise le texte, c'est-à-dire qu'on le transforme en un texte d'une seule colonne et on procède à un parsage par section en cherchant des chaînes de caractères représentant le titre de la section recherché. Une fois les articles parsés, on compare un corpus d'articles parsés à la main à un corpus parsé avec l'outil que nous avons produit à l'aide de la fonction `précision.py`. Par la suite nous observons les résultats obtenu.

3 Méthode

3.1 Conversion des PDF au format `txt`

Pour parser ce corpus de référence, l'une des premières étapes est de convertir les articles initialement au format `pdf` au format `txt`. Pour cela nous avons pris le temps de choisir entre plusieurs modules. Notre recherche nous a amenené à choisir `pdftotext` car il permettait de conserver le format des articles comportants plusieurs colonnes et au vu des différents tests que nous avons effectués avec différents modules du genre pdftotext semblait le plus prometteur (voir rapport sur GitHub). Mais `pdftotext` avait ses désavantages, par exemple pour certains titres de section en petites capitales, une espace venait s'insérer après la majuscule, ainsi REFERENCES devenait `R EFERENCES`, ce qui pouvait poser des difficultés par la suite pour la délimitation des différentes sections. À noter que nous avons aussi utilisé PyPDF2 car il présente des fonctionnalités intéressantes sur la manipulation des métadonnées des articles comme les auteurs, le titre, etc.

3.2 Conversion des PDF au format `xml`

Par la suite, il nous a été demandé de convertir les PDF en XML, ce qui consistait assez simplement à présenter d'une nouvelle manière les données extraite de l'article parsé.

3.3 Choix du langage

Pour réaliser ce projet nous avons choisi le langage python. Nous avons fait ce choix pour plusieurs raisons que je vais détailler ici.

Python a l'avantage d'être facile à comprendre et à apprendre. Il est aussi un langage à usage général. Ce langage comporte l'un des gestionnaire de paquets les plus matures, et qui comprend `pdftotext`. Mais surtout dans notre situation, Python avait l'avantage d'être connu et maîtrisé dans son ensemble par tous les membres du groupes. De plus, c'est un langage bien documenté.

3.4 Normalisation du texte

Les articles du corpus sur lesquels nous avons travaillé tout le long du projet étaient souvent des articles comportant plusieurs colonnes et cela peut poser problème. En effet, lorsque vous lisez une ligne de manière classique (avec les fonctions *readlines()* ou *read()* en python) le texte n'est pas lu une colonne après l'autre mais chaque ligne de chaque colonne en même temps, donc il faudrait soit lire colonne par colonne, soit normaliser les textes comportants plusieurs colonnes. Étant donné que certains textes ne contiennent pas plusieurs colonnes, nous avions à l'esprit d'opter pour la deuxième option car nous voulions aussi enlever du parsage les en-têtes et pieds de page qui pouvaient poser problème par la suite.

Antoine Jamelot a donc implémenté une fonction qui avait pour but de normaliser les textes dans la classe page.py.

3.5 Parsage par section

Après avoir normalisé l'ensemble des articles, nous nous sommes penchés sur le parsages des articles section par section.

Pour parser une section, au départ nous cherchions dans le texte une chaîne de caractères (`str` en Python) qui correspondait au titre de la section recherchée. Lorsque la chaîne était trouvée, nous avions donc reperé ce qui était supposé être le début. Ensuite nous cherchions le début de la section suivante et nous avions notre deuxième « flag », repère. Alors à ce moment-là, en mémorisant le numéro de ligne et de page du début et de la fin de la section, nous retournions un champ avec le numéro de page de ligne du début et de la fin. Ensuite, nous écrivions ce texte dans un `txt` ou `xml` selon le paramètre passé en ligne de commande.

Les titres comportaient des fautes dues à la conversion. Donc, si nous cherchions une chaîne de caractères, cela pouvait donner lieu à des faux négatifs. Il fallait donc faire une recherche de plusieurs chaînes de caractères et chercher un mot seul sur sa ligne — car après un titre de section, on trouve toujours un retour à la ligne.

3.6 Exploitation des métadonnées

Les métadonnées, lorsqu'elles étaient présentes, ont permis d'améliorer la précision du module, car elles ont permis de déterminer avec exactitude le titre, par exemple, ou certains auteurs (mais non leurs affiliations).

3.7 Menu graphique tkinter

Il nous a été aussi demandé de permettre de permettre le parsage par un « menu graphique ». Ce que j'ai donc permis, en lancant comme décrit dans la section du README : « Lancement de l'outil »– « Parsage », paragraphe « Menu tkinter ».

4 Résultats

Nous allons maintenant nous intéresser aux résultats que nous avons avec l'outil dans sa dernière version, c'est à dire sa version dans le sprint 4. Nous verrons comment fonctionne l'outil sur un corpus de référence composé de 10 articles et nous nous intéresserons à sa précision. Cette précision est calculée en comparant un article parsé « à la main »et un article avec l'outil. Ensuite on compare les deux parsages sections par sections, c'est-à-dire le titre, les auteurs un à un, l'abstract, l'introduction, la conclusion, les résultats et les références.

Cette précision est calculée comme ceci :

$$\text{Précision} = \frac{\text{Nombre de sections correctes}}{\text{Nombre de sections attendues}}$$

Et on obtient un nombre inférieur entre 0 et 1 que l'on peut convertir en pourcentage. Ce chiffre correspond donc à la précision du parsage de notre système sur le corpus de référence. Nous allons observer ces résultats.

4.1 Présentation résultats

Article	Précision stricte	Précision souple
BLESS,pdf	0,5	0,875
C14-1212,pdf	0,375	0,75
IPM1481,pdf	0,125	0,375
b0e5c43edf116ce2909ae009cc27a1546f09,pdf	0,375	0,5
infoEmbeddings,pdf	0,375	0,375
surveyTermExtraction,pdf	0,25	0,5
On_the_Morality_of_Artificial_Intelligence,pdf	0,334	0,334
L18-1504,pdf	0,625	0,625
Guy,pdf	0,858	0,858
acl2012,pdf	0,125	0,625
Moyenne	**0,394**	**0,582**

TABLE 1 – Résultats du parsage des 10 articles du corpus de référence.

4.2 Discussion des résultats

On peut voir que les résultats sont loins d'être parfaits. Cependant, l'ensemble de l'équipe s'y attendait, car nous avions remarqué que les auteurs et leurs affiliations n'étaient jamais écrits d'une même manière entre les différents articles, ce qui a rendu le parsage des auteurs et de leurs affiliations assez difficile. Si un seul auteur n'était pas correctement rendu, le score de précision pour l'article pouvait diminuer jusqu'à $\frac{1}{6}$, chaque article ayant au moins six sections à extraire. Mais pourquoi ces difficultés avec les auteurs et affiliations ? Cela est dû au manque d'uniformité de l'écriture des noms des auteurs, de leurs adresses et leurs affiliations à travers les différents articles du corpus, d'où les résultats de globalement 48,8 %.

5 Conclusion

Nous vous avons donc présenté notre outil de parsage. Il est certes imparfait, nous aurions pu produire un outil qui donnait une précision de 100 % sur certains articles mais ce n'était pas l'objectif du projet. L'objectif du projet était de faire un outil parsant des articles scientifiques même si leurs structures différait. C'est donc ce que nous avons essayé de faire au mieux.

6 Références

Python 3.9.4 documentation : Lien vers documentation