Parseur d'articles scientifiques

Lucia Lebrun, lebrun.e2100463@etud.univ-ubs.fr
Baptiste Lelièvre, lelievre.e1903026@etud.univ-ubs.fr
Guilhem Mazoyer, mazoyer.e2002555@etud.univ-ubs.fr
Léa Schlaflang, schlaflang.e1900394@etud.univ-ubs.fr

Université Bretagne Sud, Vannes, France

15 mai 2022

GitHub Repository

Contents

Abstract

Nous proposons avec notre parseur, une nouvelle manière de traiter les articles scientifiques. Ainsi les informations récupérées sont classées et sauvegardées dans des fichiers XML offrant une possibilité infinie d'usage.

1 Introduction

Beaucoup d'articles scientifiques sont publiés et malgré leurs qualités, il est impossible de pouvoir tous les lire en long, en large et en travers. C'est pourquoi avoir un outil facilitant la lecture des parties qui nous intéresse sur une sélection d'article donné est un atout. C'est dans ce but que nous avons commencé ce projet, proposé par les chercheurs de l'IRISA.
Nous nous sommes concentré sur le traitement des fichiers PDF car les articles sont souvent publiés sous ce format et pour autant, il est loin d'être portable et loin d'être facile à analyser par les systèmes de Traitement Automatique de Langues.

2 Choix du language

2.1 Critères

Pour choisir le langage que nous utiliserions pour développer notre projet de parseur, nous avons confronté différents langages selon des critères précis :

- Le temps d'exécution sur la modification de texte, l'ouverture et la fermeture de fichier ainsi que sur le temps de calcul
- La disponibilité des ressources de nos confrères développeurs

Ces langages sont C, Java et Python.

2.1.1 Temps d'exécution

Baptiste Lelievre s'est occupé de créer les fichiers et les programmes de tests. Les résultats obtenus permettent de mettre en compétition les langages. Le temps de calcul est le temps mis par le programme pour effectuer un grand nombre de multiplications matricielles. Le temps pour la manipulation de texte et fichiers sont le résultat de l'ouverture puis du traitement de fichiers textes. Les calculs ont été fait à partir des mêmes données pour chaque programme.

	Calcul	Texte et fichier
C	2m54s	0m39s
Java	**0m44s**	0m22s
Python	45m32	**0m13**

Ces résultats montrent que pour nos besoins, étant l'accès à des fichiers et le traitement de texte, notre candidat prédestiné est **Python**. En tenant compte que pour ce qui est de la vitesse de calcul il est très lent. Cette information peut se révéler importante durant le développement si nous avons besoin d'effectuer une masse importante de calculs.

2.1.2 Bibliothèques

Nous avons ensuite fait des recherches sur la disponibilité des librairies permettant de récupérer du texte depuis un fichier PDF. Nous avons remarqué que de nombreuses librairies publiques étaient disponibles sous Python pour récupérer du texte depuis un fichier PDF ainsi que pour traiter le texte obtenu.
Notre choix s'est arrêté sur la librairie PyMuPDF que nous utilisons pour extraire le texte ainsi que les métadonnées des fichiers traités.

3 Conceptualisation

Le choix du langage et des librairies présentés précédemment nous a permis de commencer la conceptualisation de notre application. Le projet s'organisant sous forme de sprint. À la fin de chacun d'eux, notre client nous informait des nouvelles fonctionnalités attendues. Nous avons donc fait le choix de créer une structure simple et flexible à notre programme.
Pour cela nous avons suivi un fonctionnement simple, un fichier d'exécution permettant l'initialisation, si nécessaire, d'informations par l'utilisateur. Cette exécution envoie les références à traiter à notre cerveau répartissant les tâches à toutes les branches du programme. Ainsi à chaque nouvelle fonctionnalité demandée nous pouvions simplement ajouter une branche.

4 Récupération des données

4.1 Nom du fichier

La partie la plus facile du traitement des fichiers PDF fut de récupérer le nom du fichier. En effet, en important le module *os*, nous avons pu accéder aux commandes de l'OS au sein de notre programme.
OS.PATH.BASENAME(SELF.FOLDER + "/" + FILE)

4.2 Titre

Le titre d'un article est toujours la première chose écrite. En partant de ce postulat, nous avons fait le choix de récupérer la première ligne de texte de chaque document comme étant le titre de l'article. Pour cela nous avons utilisé une expression régulière.
RE.SEARCH(REGEX_TITLE, TEXT).GROUP(0)

4.3 Auteurs, emails et affiliations

Les auteurs, leurs adresses emails et leurs affiliations ont été la partie la plus compliquée tant ils peuvent être écrits n'importe où sur le document, bien que souvent sur la première page. Ils peuvent se présenter sous toutes les formes inimaginables.
Nous commençons par trouver les adresses emails présentent dans le texte. Ces adresses sont d'une part stockées et d'autre part utilisées pour obtenir le nom, le prénom ou un mot proche que nous utilisons pour reconnaitre les auteurs dans le texte. Afin d'augmenter la précision de la recherche, nous croisons les informations traitées depuis les adresses emails avec les métadonnées du document.
Enfin nous encapsulons ces deux informations, nom et email, dans une expression régulière pour retrouver les affiliations liées à chaque auteur.

4.4 Abstract, introduction, corps, discussions, références

La récupération de ces cinq parties fonctionne de manière similaires. Nous utilisons une expression régulière visant un mot-clé du type *"Introduction"* comme départ de la recherche jusqu'au mot-clé suivant. Les parties étant toujours organisées de la même manière, cela permet de récupérer facilement chacune des parties.
La seule différence avec le reste, est la manière dont nous parcourons les pages du document source pour rechercher ces motifs. Nous partons de la première page, puis parcourons chacune des pages jusqu'à trouver le premier mot-clé, ensuite nous

repartons de la même page et continuons notre recherche jusqu'au mot-clé suivant et ainsi de suite jusqu'à trouver tous les mots-clés ou atteindre la dernière page.

4.5 Informations invalides ou manquantes

Si la détection d'une des parties présentées est considérées comme invalide par le programme, la mention N/A est stipulée. Cette invalidité peut être due au manque d'information dans le document, par une mise en forme singulière des données ou par la limitation des capacités du programme.

5 Résultat

Afin de tester la précision de notre parseur nous avons comparé les fichiers XML extraits avec des fichiers de références. Un outil[1] mis en place par notre professeur, M. Kessler, nous a permis de faire automatiquement la comparaison des fichiers.

Nous avons obtenus une précision en pourcentage pour chaque partie de chaque fichier XML de test. Nous avons testé tous les documents et noté les valeurs retournées dans un tableau de calcul[2].

6 Conclusion

Ainsi se conclut cet article retraçant la création de notre parseur développé sous Python. Cette expérience nous aura appris la diversité des formats des articles scientifiques pourtant étant une création codifiée. Cela nous aura permis d'aborder le développement sous un angle que nous n'avions pas approfondi et qui, j'en suis certains, nous permettra de progresser dans tous les domaines informatiques que nous exercerons par la suite.

Ce fut un plaisir de faire ce programme pour les chercheurs de l'IRISA.

[1] http://inf1603.alwaysdata.net/ParserResultComparatorTest
[2] https://docs.google.com/spreadsheets/parseur