Parseur d'articles scientifiques

Antoine ADAM, Sylvain BUCHE, Guillaume COBAT

Abstract

Les chercheurs de l'IRISA ont demandé la création d'un outil qui leur permettrait de lire des aperçus d'articles scientifiques. En effet, ils n'ont pas le temps de tout lire, et un logiciel de ce genre leur permet de gagner un temps précieux. Les articles scientifiques sont très souvent sous format PDF, un format peu exploitable pour les systèmes de Traitement Automatique de Langues (TAL). Nous avons donc développé un parseur d'articles scientifiques qui extrait des informations des PDF puis génère des fichiers textes (.txt) ou XML, qui permettent aux systèmes TAL de travailler correctement.

1 Méthode

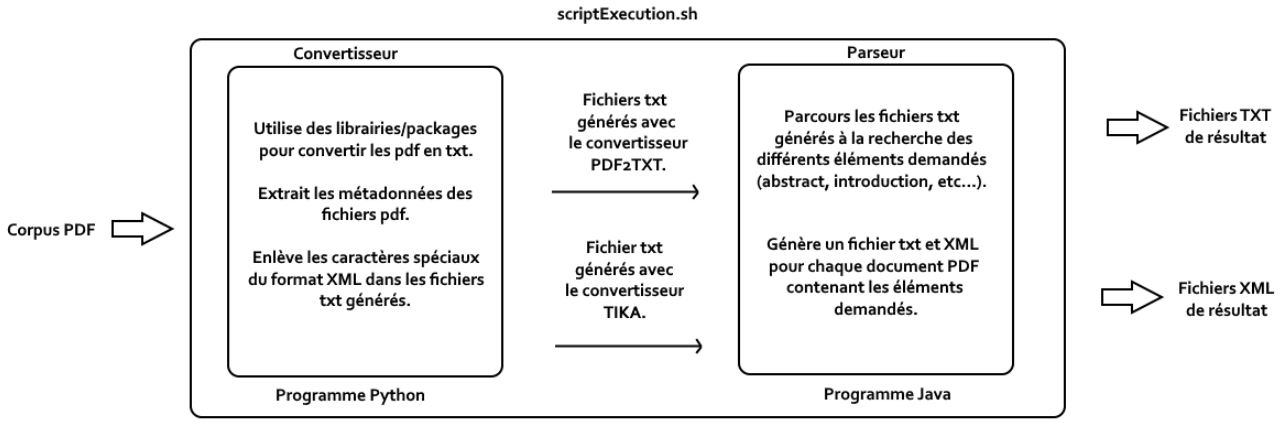
1.1 Introduction

Ce projet de parseur d'articles scientifiques a été mené de janvier à mai 2022. Il a été réalisé en suivant la méthode Scrum, c'est-à-dire que les fonctionnalités ont été ajoutées au fur et à mesure des sprints, des périodes de 2 ou 3 semaines lors desquelles l'équipe de développement ajoutait une ou plusieurs fonctionnalités bien précises. Après chaque sprint, l'équipe se retrouvait lors de la rétrospective de sprint pour discuter de la manière dont le sprint a été réalisé et comment améliorer le suivant. Ces moments d'échange nous ont permis d'identifier des problèmes surtout lors des premiers sprints, le temps que le projet soit correctement lancé.

1.2 Fonctionnement global

Lors du premier sprint, nous avons dû identifier les outils que nous allions utiliser pour convertir un PDF en format texte. Il faut en effet savoir que le parseur prend en entrée un ou plusieurs fichiers PDF, et le but et d'en extraire les données et de les exporter en format texte ou XML. Notre solution fonctionne donc comme suit :

Figure 1: Schéma de fonctionnement du programme



Le choix que nous avons fait pendant le premier sprint s'est porté sur un convertisseur nommé PDF2TXT. Cependant, malgré les bons résultats des tests réalisés pendant la phase de sélection, nous avons décidé d'ajouter un autre convertisseur nommé TIKA lors du développement. En effet, les fichiers générés par PDF2TXT lors du développement manquaient cruellement de précision.

Nous avons fait le choix d'utiliser 2 langages différents pour l'utilisation des convertisseurs et le parseur. Les librairies python dédiées à la conversion de PDF en texte sont puissantes et facilement accessibles. Cependant notre équipe de développement n'était pas suffisamment familiarisée avec ce langage en début de projet pour donner des résultats concluants de façon rapide. Nous avons donc décidé de réaliser le parseur en Java, un langage avec lequel l'ensemble de l'équipe est habitué à travailler. Ce découpage entre les 2 langages est particulièrement efficace, car il se situe au moment de l'exécution où des fichiers texte sont générés par le premier script puis sont repris par le second.

1.3 Récupération des éléments

Nous avions pour mission de récupérer les éléments suivant à l'intérieur des fichiers PDF :

- Nom du fichier (preambule).
- Titre de l'article.
- Nom des auteurs, mail et affiliation de ces derniers.
- Résumé (abstract).
- Introduction.
- Corps du document.
- Discussion.
- Conclusion.
- Bibliographie.

Nous avons décidé de construire notre parseur de manière modulaire. C'est-à-dire que chaque élément recherché fait l'objet d'une classe à part, ce qui nous a permis de travailler et d'optimiser indépendamment chaque partie. Le fonctionnement se déroule comme suit : la classe Main appelle 2 instances de Parseur qui prennent en entrée chacune les fichiers TXT provenant des 2 convertisseurs. Ensuite les classes de recherches (Find...) sont appelées par les parseurs pour chercher les éléments dans le texte. La recherche se fait à partir de chaines de caractères récurrentes dans les articles scientifiques. Par exemple pour rechercher l'introduction, la classe FindIntroduction va chercher une occurrence du mot Introduction dans plusieurs formats (ex : Introduction, introduction, 1. Introduction...). Une fois le début de l'introduction trouvée, FindIntroduction va chercher un élément qui indique que le paragraphe d'introduction est terminé (ex : une ligne vide suivie d'un '2. ' ou '2 '). Lorsque la fin est trouvée, la classe retourne l'introduction sous forme de String.

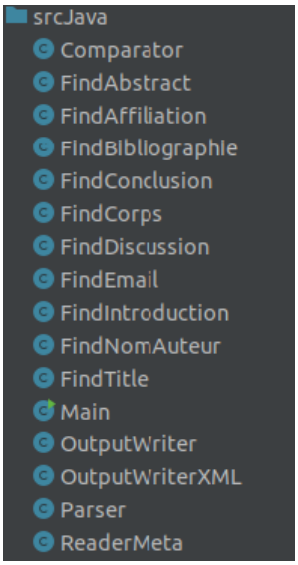


Figure 2: Arborscence du parseur

Quand chaque élément est trouvé (ou non, dans ce cas-là, le String est nul), les résultats obtenus à l'aide des 2 sources différentes (les 2 convertisseurs) sont comparés pour affiner les résultats. Il en est de même avec les métadonnées collectées par les convertisseurs, le but étant d'augmenter la précision des résultats. Cette étape est extrêmement importante, elle nous a permis d'augmenter la précision globale du parseur de plus de 35%.

Enfin, les classes OutputWriter et OutputWriterXML sont appelées pour générer le résultat, respectivement au format texte et XML.

2 Résultats

Figure 3: Compilation des résultats pour tous les fichiers du corpus de test

	Preamble	Titre	Auteurs	Introduction	Abstract	Discussion	Conclusion	Bibliographie	Total
BLESS	100	98,99	58,19	96,24	98,32	N/A	82,24	97,58	**90,22**
C14-1212	100	98,99	12	81,82	99,1	N/A	96,81	95,91	**83,52**
Guy	100	99,47	48,33	53,39	94,09	N/A	92,95	49,43	**76,81**
IPM1481	100	23,71	52,79	55,56	96,77	N/A	78,79	42,52	**64,31**
L18-1504	100	100	15	98,22	99,15	N/A	62,06	26,97	**71,63**
On_the_Morality_of_Artificial_Intelliger	100	96,39	2,67	6,45	99,42	N/A	91,73	86,01	**68,95**
acl2012	100	99,07	60,1	87,02	60,29	N/A	91,9	94,87	**84,75**
b0e5c43edf116ce2909ae009cc27a15	100	99,29	52,41	95,96	92,43	N/A	98,38	98,11	**90,94**
infoEmbeddings	100	66,67	53,92	37,91	89,86	N/A	93,53	88,03	**75,70**
surveyTermExtraction	100	98,21	20,89	17,58	68,01	N/A	75,42	85,3	**66,49**
Total	**100,00**	**88,08**	**37,63**	**63,02**	**89,74**	N/A	**86,38**	**76,47**	**77,33**

Les résultats obtenus par notre parseur sont de 77,33%. Sur les 8 items recherchés, 4 dépassent les 80% de précision. À noter, la précision réelle de l'item "preamble" n'est pas de 100%, car sur les fichiers présents sur le site de calcul de précision, un espace a été ajouté devant le nom du fichier dans l'item "preamble" (ex. : " BLESS.xml" au lieu de "BLESS.xml"). Lorsque nous ajoutons un espace devant le "preamble" dans notre fichier XML généré, la précision est de 100%. C'est pourquoi nous avons considéré que la colonne "preamble" atteignait 100% de précision, l'erreur venant du site de calcul de résultat et non de notre parseur.

L'item qui a été le plus difficile à extraire a été sans aucun doute les auteurs. Cela s'explique, car il y avait de nombreux éléments à chercher (plusieurs auteurs, avec pour chacun, leur nom, prénom, mail et affiliation), et aussi, car nous avons eu beaucoup plus de mal à trouver un pattern de recherche, une structure présente dans tous les articles.

3 Conclusion

3.1 Commentaire des résultats

Suite au développement du parseur, nous avons obtenu une précision de 77,33% en moyenne sur le corpus de test. Ce résultat est très positif, nous le considérons comme bon, compte tenu du fait que les résultats que nous avions une semaine auparavant ne dépassaient pas les 45% de précision. L'utilisation d'un 2e convertisseur plus précis nous a permis de gagner nettement en précision. Tout au long du projet, nous avons su garder un code très modulable, ce qui nous a permis d'augmenter facilement la précision de chaque élément recherché pour le dernier sprint.

3.2 Améliorations possibles

Nous avons pris beaucoup de temps à choisir les technologies à utiliser en début de projet. Par conséquent, nous avons accumulé un certain retard sur les premiers sprints.

4 Annexe

Les sources du projet sont disponibles sur GitHub : https://github.com/Nefaryos/PDF_Parser/tree/sprint4

PS : La branche la plus récente est celle du sprint 4. Nous n'avons pas jugé nécessaire de créer une nouvelle branche pour le sprint 5 car il n'y avait pas de nouvelles fonctionnalités à ajouter.