Fouille de données, fouille de textes : les titres de HAL

Introduction

Dans ce travail, nous étudions des titres de publications stockées sur l'archive ouverte HAL. 53 426 titres ont été fournis avec, pour chacun, des informations supplémentaires liées à la publication titrée : domaine (Lettres, Linguistique ou Informatique), type (article, communication ou chapitre d'ouvrage), année de publication et nombre d'auteurs. Nous parlerons de **variables** (appelées aussi traits, attributs, descripteurs ou *features*). Dans une première partie, nous nous attachons à rechercher des relations entre les variables non textuelles, les numériques et les énumérations (*Nominal* pour **Weka**, *categorical* pour **Scikit-learn**). Une relation permet, à partir d'une ou plusieurs **variables**, de prédire la valeur d'une autre, appelée **variable cible** (nous ne parlons pas du cas de plusieurs variables cibles). Dans la deuxième partie, nous prenons comme variable cible le **domaine** et essayons d'établir des **variables dérivées du titre** pour ensuite construire des modèles de classification automatique dans la troisième partie.

1. Analyse des champs non textuels

Dans un premier temps, nous utilisons simplement Python et **pandas** pour recueillir des informations générales sur les champs non textuels à partir du CSV encodé en UTF-8 :

- **Domaine** Informatique : 18 694 titres (35 %), Linguistique : 17 582 (33 %), Lettres : 17 150 (32 %)
- **Support** Communication : 24 693 (46 %), Article : 17 273 (32 %), Chapitre : 11 460 (22 %)

On compte entre 1 et 38 auteurs par titres, avec une moyenne de 1,83. 62 % des titres ont été écrits par un auteur, 14 % par deux auteurs, 12 % par trois et 6 % par 4 soit 94 % des titres ont été écrit par moins de 5 auteurs. Si on écarte certaines incohérences, les années s'étendent de 1771 à 2018. Les titres de 2000 à 2018 comptent pour 93 % des titres.

Après avoir étudié dans les grandes lignes nos données non-textuelles, nous utilisons **Weka** pour analyser les relations entre celles-ci. Nous typons nos variables, *Nominal* pour **domaine** et **support**, *Numeric* pour **années** et **auteurs**, et supprimons ensuite la variable **titre**. Pour voir les relations qu'entretiennent les variables entre elles, nous utilisons un classifieur de type arbre de décision basé sur l'algorithme J48 avec ses paramètres par défauts car il gère les variables des types *Numeric* et *Nominal*. Nous ne pouvons construire d'arbre qu'en ciblant les variables de type *Nominal* néanmoins, nous en construisons donc deux : l'un ciblant le **domaine**, l'autre le **support**. En observant les arbres, on peut estimer l'importance des variables pour déterminer la cible. Pour la cible **domaine**, rien n'apparaît clairement à l'étude de l'arbre de Weka. Pour la cible **support**, le premier choix de l'arbre, et donc le critère le plus déterminant, est la variable **domaine**, mais il n'est pas le seul : vient ensuite l'**année** et plus tard le **nombre d'auteurs**. Avec Python, on dresse ces trois tableaux en considérant 2 variables :

Sup-port	Domaine		
	Info.	Lett.	Ling.
Article	**3 898**	6 617	6 758
Comm.	**13 629**	4 498	6 566
Chapitre	1 167	6 035	4 258

Sup-port[1]	Domaine		
	Info.	Lett.	Ling.
ART	23 %	38 %	39 %
COMM	**55 %**	18 %	27 %
COUV	**10 %**	53 %	37 %

Sup-port	Domaine[2]		
	Info.	Lett.	Ling.
ART	21 %	39 %	38 %
COMM	**73 %**	26 %	37 %
COUV	**6 %**	35 %	24 %

À gauche, le comptage brut. Au milieu, on prend **domaine** pour cible et **support** comme variable, les pourcentages sont par rapport au nombre total de titres pour un support donné. À droite, on intervertit en prenant **support** pour cible et **domaine** comme variable, les pourcentages sont donnés par rapport au nombre total de titres pour un domaine donné. Sur nos données (1), un titre de chapitre a seulement 10 % de chance d'être dans le domaine informatique alors que pour une communication ce pourcentage s'élève à 55 %. (2) Un titre dans le domaine informatique a 73 % de chance d'être celui d'une communication et seulement 6 % celui d'un chapitre. Rien d'aussi probant ne se dégage des autres statistiques qui ont une répartition plus équitable. **On peut donc affirmer que les domaines ne sont pas représentés de la même façon entre les supports et que les supports ne sont pas représentés de la même façon entre les domaines**.

Nous utilisons à présent une autre fonction de Weka : la visualisation. Nous testons plusieurs combinaisons de variables en abscisse et en ordonnée, ainsi que d'éparpillement (*jitter*). Nous supprimons pour utiliser cette fonction les trois titres ayant comme année 195, 217 et 218 sinon les années sont irreprésentables. Le résultat est décevant : le fait que la représentation soit écrasée sur les bordures ne nous permet pas

d'identifier clairement les tendances, même celle évoquée du **domaine** Informatique vers le **support** Communication, alors qu'elle est forte. Nous détectons cependant dans la configuration **support** en abscisse et **nombre d'auteurs** en ordonnée des formes de points différentes. Nous construisons un script Python pour établir des statistiques précises en prenant pour cible le **nombre d'auteurs** par rapport aux **supports** et obtenons les 4 premières lignes de résultats suivantes :

Nombre d'auteurs	ART			COMM			COUV		
	Nb	%	Cumul	Nb	%	Cumul	Nb	%	Cumul
1	12 485	72 %	**72 %**[2]	11 213	45 %	**45 %**[3]	9 490	**83 %**[1]	83 %
2	2 409	14 %	86 %	4 052	16 %	61 %	1 270	11 %	**94 %**[4]
3	1 204	7 %	**93 %**[5]	4 804	20 %	81 %	418	4 %	98 %
4	6 25	4 %	97 %	2 733	**11 %**[8]	**92 %**[6]	160	1 %	99 %

Ce tableau confirme l'intuition visuelle : les différents **supports** n'ont pas le même profil concernant le **nombre d'auteurs**. (1) Si le support est un chapitre d'ouvrage, il y a 83 % de chance qu'il soit écrit par un seul auteur, cette proportion est à 72 % pour un article (2) et elle tombe à 45 % pour une communication (3). L'accélération du cumul n'est pas la même : si le support est un chapitre d'ouvrage, il y a 94 % de chance qu'il soit écrit par 1 ou 2 auteurs (4). Pour dépasser les 90 %, il faut considérer les titres écrits par de 1 à 3 auteurs pour les articles (5), et les titres écrits par de 1 à 4 auteurs pour les communications (6). **Le nombre d'auteurs varie donc bien en fonction du support de publication.** On peut également inverser la réflexion et prendre pour cible le **support** par rapport au **nombre d'auteurs**, en affichant seulement de 1 à 4 auteurs :

Support	1		2		3		4	
	Nb	%	Nb	%	Nb	%	Nb	%
ART	12 485	38 %	2 409	31 %	1 204	19 %	6 25	18 %
COMM	11 213	34 %	4 052	52 %	4 804	75 %	2 733	**78 %**[7]
COUV	9 490	29 %	1 270	16 %	418	7 %	160	5 %

La règle *« Si un titre a 4 auteurs alors il a pour support une communication »* pour trouver toutes les communications a une précision de 78 % (7), en d'autres termes, sa valeur prédictive positive est de 78 %, mais elle a un rappel de 11 % seulement (8). Notre mesure de compromis, ou f-mesure, est de 0.19. On peut comparer cette mesure à une autre reprenant le critère le plus déterminant choisi par l'algorithme J48 pour trouver le **support**, le **domaine** : *« Si un titre est en informatique, c'est une communication »*. Cette règle a une précision de 73 % et un rappel de 55 %, soit une f-mesure de 0.63 ce qui est bien meilleur. Nous pouvons maintenant passer à l'extraction de variables depuis du texte.

2. Analyse du titre en fonction du domaine

Les modèles d'apprentissage automatique ne peuvent pas travailler directement à partir de texte. Il faut extraire des variables depuis celui-ci. Classiquement, pour chaque mot présent dans le texte, on va créer une variable numérique, technique appelée *one hot encoding*. Nous créons à partir de nos données un nouveau fichier Weka (.arff) et nous ne gardons que la variable **titre** et **domaine** comme cible. Nous transformons la colonne titre en 1000 colonnes numériques indiquant la présence d'un seul mot avec la fonction StringToWordVector(), en appliquant le Snowball stemmer et la mise en casse minuscule (pour que représentation, Représentations et représentations ne forment qu'une colonne), en ignorant les mots vides du français et une fréquence minimale de 5 mots. On lance un classificateur arbre J48 en cross-validation avec 3 replis pour avoir des résultats généralisables et non surentraînés sur nos données. Bloqué par le manque de mémoire, nous décidons d'aborder la question autrement.

Nous divisons nos 53 426 titres en deux parties, 2 tiers pour l'échantillon d'apprentissage, *train*, 1 tier pour l'échantillon *test*, avec la fonction train_test_split de **scikit-learn** en veillant à avoir à la même proportion de domaines dans les deux grâce à l'argument *stratify*. Nous analysons les longueurs moyennes des titres en caractères et en mots, on note la position médiane de la linguistique mais malheureusement elles sont trop proches pour être véritablement distinctives :

- **Nb de caractères** Linguistique = **81** Informatique = 80 Lettres = 76
- **Nb de mots** Linguistique = 12.08 Informatique = 11.70 Lettres = **12.34**

Nous envoyons nos titres à **Talismane** pour lemmatisation. Nous étudions tout d'abord la répartition des parties du discours (POS) dans chaque domaine et remarquons des différences (voir Annexe C). Les titres des Lettres utilisent bien plus de ponctuation (1) que l'informatique, et la linguistique occupe encore une position médiane. Pour les noms propres, les Lettres en utilisent plus de deux fois plus (2) que les autres domaines. L'Informatique utilise plus de ponctuations (3) et de noms communs (4). **Nous pourrions donc prendre comme variable la moyenne de nom propre ou celle de ponctuation** par rapport au nombre d'éléments du titre.

Nous construisons ensuite pour chaque titre une liste des lemmes employés, en supprimant les mots vides, les doublons et en prenant la forme en casse minuscule si le lemme est '_', que nous appelons « **silhouette** ». Nous regardons les mots les plus fréquents des silhouettes dans chaque domaine du corpus *train*. En linguistique, avec 1 316 occurrences, le mot « français » est le plus fréquent. Cela représente 11 % de la totalité des mots des silhouettes de ce domaine. Comme il n'y a plus de doublons, nous pouvons dire que 1 316 titres du domaine linguistique ont au moins une fois ce mot. La règle « Si le titre contient le mot "français", alors il est du domaine Linguistique » a une précision, ou valeur prédictive positive, égale à 1 316 divisé par le nombre total de titres avec ce mot, soit 1 316 en Linguistique, 103 en Informatique et 296 en Lettres : 77 %. Elle a un rappel égal à 1 316 divisé par le nombre total de titres dans le domaine Linguistique dans le corpus *train*, soit 11 722, pour un rappel de 11 % et une F-mesure de 20 %. C'est en combinant ces règles que l'on peut espérer obtenir un bon classificateur. **Si nous devions prendre 10 traits, nous choisirions les 10 mots présentant la meilleure F-mesure dans chaque domaine** (voir Annexe D pour un tableau récapitulatif).

Néanmoins, la couverture de ces 10 traits textuels est trop faible : si on regarde le pourcentage de titres dans un domaine donné ayant au moins un de ces 10 premiers mots, on est seulement à 40 % en Informatique, 30 % en Lettres et 42 % Linguistique. Nous voulons savoir combien de traits faut-il prendre pour chaque domaine pour atteindre une couverture donnée (voir Annexe E pour le tableau). Si on mesure la dérivée de la fonction reliant le nombre de traits à la couverture donnée, on constate que l'Informatique s'identifie bien plus vite et plus facilement car elle nécessite un faible nombre de traits pour atteindre les 90 % de couverture. De plus, il reste à résoudre la problématique des mots présents dans plusieurs domaines : comment catégoriser le titre dans ce cas, avec quelles pondérations ? Ce type de calculs est au centre des modèles de **Scikit-learn**.

3. Classification automatique d'un titre par domaine

Dans cette partie, nous comparons trois classificateurs pour trouver le **domaine**. Le premier est construit à partir de nos **silhouettes**. Celui-ci donne un score par domaine à chaque silhouette. Ce score est la somme des F-mesures des mots de la silhouette pour un domaine donné. L'algorithme considère le titre étant du domaine ayant le plus haut score. Pour calibrer le temps d'exécution et la consommation mémoire, nous pouvons nous limiter à reconnaître les X premiers mots de chaque domaine, les autres étant ignorés. Nous avons fait les calculs pour 100, 1 000, 10 000 et tous les traits. Du fait des imperfections de notre algorithme, il était intéressant de le tester aussi sur le corpus *train* ayant permis d'établir le modèle, pour mesurer la « perte » subit par la limitation, avant d'essayer de prédire *test*. Plus on ajoute de traits, plus l'exactitude (accuracy) du classificateur augmente. On obtient le même résultat avec 10 000 traits et tous (on en compte 12 832 en Informatique, 20 111 en Lettres et 12 918 en Linguistique). Avec tous les traits nous ne sommes qu'à 81 % d'exactitude sur le corpus *train* et 78 % sur le corpus *test*. Augmenter le nombre de traits diminue le nombre de silhouette évaluée à 0 : de 5 587 pour 100 traits, on tombe à 73 pour 10 000 et 0 avec tous. Aucun titre ne génère une égalité entre deux domaines, l'évaluation peut toujours trancher. Seulement 5 titres n'ont pas de silhouette, qui seront donc inclassables. Le temps de construction de notre modèle, le calcul de la F-mesure pour chaque mot des silhouettes, et le temps de prédiction sont inférieurs à 1 seconde pour tous les traits et tous les échantillons. Nous avons essayé un autre classificateur basé sur une seule POS et obtenu une mauvaise exactitude, 22 % en se basant sur NPP, 24 % sur PONCT, ainsi qu'en combinant ces deux variables avec notre modèle basé sur les silhouettes : 57 %.

Le deuxième classificateur, construit avec **scikit-learn**, est un arbre de décision DecisionTreeClassifier qui prend tous les mots du titre et pas seulement ceux de nos silhouettes. Nous utilisons les hyperparamètres *min_samples_split* à 20 et *random_state* à 99. Le troisième classificateur est un modèle Support vector machines (SVC) de **scikit-learn**. Ces deux derniers classificateurs ont été testés à la fois avec un entraînement sur un corpus *train* doté seulement de la variable **titre**, dont nous extrayons des variables avec un TfidfVectorizer, et un corpus *train2* avec la variable **titre** et les variables non textuelles **nombre d'auteurs**, **année** et **support**, pour prédire

respectivement *test* et *test2*. Sur *test,* l'arbre prédit en 0,79 seconde en moyenne avec une exactitude de 92 %. L'arbre comporte 16 311 nœuds et à une profondeur de 656. Nous remplaçons alors la variable **titre** par notre **silhouette**, en concaténant les mots qui la composent et en la soumettant à la même vectorisation. Nous obtenons à alors en 2 secondes en moyenne une prédiction avec une exactitude de 95 %, pour un arbre de 13 235 nœuds et presque trois fois plus profond avec une profondeur de 1 906. Rajouter les autres variables, en utilisant *train2* et en essayant de prédire *test2*, fait baisser l'exactitude à 75 %. Il semble qu'en rajoutant les informations contenues dans les autres variables, le modèle soit moins exact. Notons que le DecisionTreeClassifier est capable de fournir un arbre au format graphviz/dot pour visualiser ses nœuds. Malheureusement, avec autant d'échantillons et de variables, nous n'avons pas réussi à le produire.

Le troisième classificateur, SVC, avec *gamma* à 0.001 et *C* à 100, sur la variable **titre** seule vectorisée par TfidfVectorizer a une exactitude de 0,848 %. Sur la **silhouette** seule vectorisée, l'exactitude est de 85,2 %. L'amélioration due au passage du **titre** à la **silhouette** est donc beaucoup plus faible que pour le DecisionTreeClassifier. En rajoutant les données non textuelles à la silhouette, on tombe à 84,7 %. Là aussi, l'ajout de ces données diminue l'exactitude mais dans une ampleur bien moindre. Nous avons ensuite choisi de faire varier les hyperparamètres *kernel function* entre 'rbf' et 'linear', le *gamma* entre 0.001 et 0.0001 et *C* entre 10, 100 et 1000. **Scikit-learn** dispose d'un objet GridSearchCV pour implémenter un test automatique de ces paramètres et la sélection de la meilleure combinaison. Diminuer *C* par rapport à 100 entraîne une perte d'exactitude, l'augmenter à 1000 fait passer à 86,5 % (+1 %) mais le temps de construction du modèle explose (23 minutes). La diminution du *gamma* fait également baisser l'exactitude et le *kernel* 'linear', même avec *C* à 1000, stagne à 78 % d'exactitude, le 'rbf' reste meilleur.

On s'intéresse ensuite aux erreurs de classement lorsqu'on a seulement une variable textuelle. Dans le cas de notre modèle sur la variable **silhouette**, une erreur classique et la présence de trop de mots « étrangers » au domaine, c'est-à-dire ayant un très bon score dans un domaine différent du domaine du titre. Ainsi, le titre *« Mécanisme de synchronisation scalable à plusieurs lecteurs et un écrivain »*, avec pour silhouette *« mécanisme synchronisation scalable plusieurs lecteur écrivain »* a été classé avec tous les traits textuels de notre modèle en *Lettres alors qu'il est en Informatique. L'évaluation des scores pour chaque domaine donne 0.0093 pour Linguistique, 0.0155 pour Informatique et 0.0398 pour Lettres. Informatique est le seul domaine à attribuer des points pour chaque mot, Linguistique et Lettres n'en donnant pas pour *synchronisation* et *scalable*. Mais les poids des mots *écrivain* (f-mesure de 0.0221) et *lecteur* (0.0164) expliquent le classement en *Lettres. Ce titre est également incorrectement classé en *Lettres par un SVC avec *gamma* à 0.001 et *C* à 100. Le DecisionTreeClassifier à *min_samples_split* à 20 classe lui bien ce titre. Il commet néanmoins aussi des erreurs, comme avec ce titre *« Le cycle de reproduction des zoonymes »*, dont la silhouette est *« cycle reproduction zoonymes »*, classé en *Informatique au lieu de Linguistique. Ce titre est également mal classé avec le SVC, mais différemment : il s'agit pour lui d'un *Lettres. Mon modèle, bien qu'ayant une exactitude inférieure, catégorise bien ce titre. Sans pouvoir observer l'arbre du DecisionTreeClassifier pour des raisons de puissance technique, il est difficile de comprendre cette erreur. On peut néanmoins se rabattre sur mon modèle voir comment il a fait son choix en calculant le score pour chaque domaine : 0.0053 pour Linguistique, 0.0046 pour Informatique, 0.0038 pour Lettres. *cycle* est à 0.0040 en Informatique et 0.0039 en Linguistique, 0.0035 en Lettres. C'est lui qui compte pour l'essentiel du poids mais il ne permet pas de départager les trois domaines. *reproduction* est évalué à 0.0006 en Informatique, 0.0006 en Linguistique et 0.0003 en Lettres. Là encore, Informatique et Linguistique sont très proches. . Mais *zoonymes* n'est connu que du domaine Linguistique, et son score de 0.007 permet de trancher.

Conclusion

Nous tirons deux principaux enseignements de ce travail. En premier, le prétraitement des champs textuels permet d'améliorer l'exactitude du modèle DecisionTreeClassifier, qui gagne 3 % en remplaçant la variable textuelle **titre** par sa **silhouette**, composée des lemmes, ou des formes si le lemme est introuvable par **Talismane**, en ayant supprimé les doublons et les mots vides. Peut-être aurions-nous dû garder les doublons néanmoins pour renforcer la catégorisation et le gain n'est que de 0.004 % sur le SVC, soit moins que ce que l'on peut obtenir en augmentant l'hyperparamètre *C*. En second, l'ajout d'information n'est pas synonyme de gain d'exactitude. La seule configuration où cela se produit est avec un SVC avec *gamma* à 0.001 et *C* à 100 sur **titre** TfidfVectorizeré et les **variables non textuelles** où on gagne 0.002 %. C'est très faible et toutes les autres configurations indiquent une perte. Peut-être est-ce dû à notre union des variables qui n'est pas assez bien faite.

Annexe A : Résumés des modèles et de leurs performances

Paramètres	Exactitude
DecisionTreeClassifier *min_samples_split*=20 sur **titre** TfidfVectorizeré + features !TXT	70 %
DecisionTreeClassifier *min_samples_split*=20 sur **silhouette** TfidfVectorizeré + features !TXT	75 %
Mon modèle sur **silhouette**	78 %
SVC avec *gamma* à 0.001 et *C* à 100 sur **silhouette** TfidfVectorizeré + features !TXT	0.847 %
SVC avec *gamma* à 0.001 et *C* à 100 sur **titre** TfidfVectorizeré	0.848 %
SVC avec *gamma* à 0.001 et *C* à 100 sur **titre** TfidfVectorizeré + features !TXT	0.850 %
SVC avec *gamma* à 0.001 et *C* à 100 sur **silhouette** TfidfVectorizeré	0.852 %
SVC avec *gamma* à 0.001 et *C* à 100**0** sur **silhouette** TfidfVectorizeré	0.863 %
DecisionTreeClassifier, *min_samples_split* à 20 sur **titre** TfidfVectorizeré	92 %
DecisionTreeClassifier, *min_samples_split* à 20 sur **silhouette** TfidfVectorizeré	95 %

Annexe B : Dépôt Git du code

Tout le code est accessible à l'URL : https://github.com/Xitog/tal/tree/master/master2/exodata

Annexe C : Fréquence des POS dans les titres

POS	Informatique		Lettres		Linguistique	
NPP	10 564	7 %	26 234	16 %[2]	9 002	6 %
PONCT	9 141	6 %	24 333	15 %[1]	17 149	11 %
P	31 080	20 %[3]	21 320	13 %	23 084	15 %
NC	56 697	37 %[4]	42 156	25 %	50 848	32 %
ADJ	13 790	9 %	11 706	7 %	15 608	10 %

Le pourcentage se rapporte au nombre total de POS dans chaque domaine.

Annexe D : Les 10 traits les plus importants pour les silhouettes, par domaine

Info.	P	R	F	Lettres	P	R	F	Linguistique	P	R	F
système	87	8	14	siècle	75	6	11	français	77	11	20
application	94	6	11	littérature	90	5	10	langue	85	11	20
modèle	80	6	11	roman	82	4	8	linguistique	91	7	12
réseau	93	6	11	entre	41	4	7	discours	79	5	10
donnée	83	5	9	histoire	70	4	7	analyse	43	4	7
analyse	54	5	9	chez	56	4	7	entre	41	4	7
approche	62	5	8	littéraire	82	3	6	étude	50	4	7
Image	69	4	8	écriture	62	3	5	cas	57	4	7
modélisation	90	4	7	français	17	3	5	corpus	71	3	6
Multi	97	4	7	théâtre	88	2	4	construction	55	3	5

Annexe E : Couverture par nombre de traits pour notre modèle

Quelle couverture pour :	Info.	Lettres	Ling.
50 traits	75 %	58 %	70 %
150 traits	90 %	78 %	86 %
1 000 traits	99 %	95 %	97 %
10 000 traits	100 %	99 %	100 %