Rapport de stage de L3 : Extraction de concepts par AFC dans des corpus

Valentin Taillandier, Inalco

24 août 2018

Référence : Stage effectué du lundi 21 mai au vendredi 20 juillet 2018 à la maison de la recherche de l'Inalco (laboratoire Er-Tim) sous la direction de Damien Nouvel (maître de conférence, Inalco).

Résumé

L'AFC (Wille, 1982) est une méthode d'extraction de connaissances à partir de données, qui s'appuie sur le lien entre les objets et leurs attributs. Celle-ci extrait des « concepts », organisés dans un treillis par une relation d'ordre partiel. Le sujet du travail porte sur le traitement automatique des langues (TAL) par utilisation d'analyse formelle de concepts (AFC). Plusieurs travaux ont déjà abordé cette question, mais peu d'entre eux l'interrogent à la fois sur ses aspects formels (algèbre des concepts à extraire) et appliqués (passage à l'échelle sur de gros volumes de données). Nous expérimentons cette méthode pour la classification supervisée de documents, et la comparons aux méthodes traditionnelles (TF.IDF et LSA)

Mots-clés : Traitement automatique des langues, Analyse formelle de concepts, classification.

Table des matières

Remerciements

Je remercie chaleureusement Damien Nouvel pour son accueil au sein de son équipe, en temps que maître de stage. Sa bienveillance et son soutien m'ont permis de réaliser ce stage dans les meilleures conditions. Au delà de cela, Damien Nouvel m'a permis de découvrir le monde de la recherche et m'a donné accès à diverses conférences dans le domaine du Traitement Automatique du Langage Naturel.

Introduction

La classification de documents est une discipline du Traitement Automatique du Langage Naturel (TALN ou plus simplement TAL) qui consiste à classer automatiquement un ensemble de documents rassemblés sous la forme d'un corpus. Cette classification peut être faite selon une multitude de critères comme le genre, le thème ou encore la langue d'écriture. Cette tâche est réalisable grâce à des algorithmes de traitement de l'information, les algorithmes étant choisis selon la nature de la classification et celle des documents.

L'*analyse formelle de concepts* (AFC) a été introduite par Rudolf Wille en 1982 [8] et étudie les concepts d'un point de vue formel. Elle permet de rassembler les éléments d'un ensemble partageant les mêmes attributs dans des concepts.

L'objet de notre travail est d'explorer les concepts formels comme source d'information pour la classification de documents. Notre but est de déterminer si l'utilisation de concepts formels permet d'obtenir des résultats comparables à ceux des méthodes traditionnellement utilisées, mais aussi d'étudier son coût en terme de temps de calcul. Nous étudierons les programmes d'extraction de concepts existants via leur rapidité d'exécution. En d'autres termes, nous n'étudierons pas les algorithmes sous-jacents.

1 Contexte du stage

Ce stage a été effectué dans le cadre de la première année du magistère d'informatique de l'École Normale Supérieure de Rennes. Cette section présente l'équipe et l'organisation de travail.

1.1 L'équipe

Le stage a été effectué au sein de l'équipe Er-Tim [1] localisée à la maison de la recherche de l'Inalco [2] à Paris. Cette équipe se concentre sur le traitement automatique des langues et l'ingénierie multilingue. Elle a pour objets principaux la recherche en sémantique des textes pour les applications, le développement de méthodologie pour l'ingénierie des textes, des documents numériques multilingues et la production de ressources multilingues.

1.2 Le déroulement

Ce stage m'a permis de découvrir le travail de recherche et la vie au sein d'une équipe. Durant mon stage, j'ai été encadré par Damien Nouvel, maître de conférence à l'Inalco. Dans un premier temps, j'ai effectué un travail bibliographique visant à comprendre les concepts formels. Pour cela, j'ai entrepris la lecture du livre *Formal Concept Analysis : Mathematical Foundations* [2] qui se concentre sur la définition de l'AFC et des ses nombreuses propriétés mathématiques héritées de la théorie des treillis de Galois. Le livre contient aussi quelques algorithmes permettant d'explorer les concepts d'un contexte. Cette première tâche aura pris deux semaines. J'ai ensuite étudié les algorithmes d'extraction de concepts. Il a fallu trouver des algorithmes rapides permettant d'extraire les concepts des corpus que nous allons utiliser. Les corpus étant

1. Équipe de recherche textes, informatique, multilinguisme : `http://www.er-tim.fr/`.
2. L'Institut national des langues et civilisations orientales. Anciennement *Langues O'*.

parfois volumineux, l'utilisation de tels algorithmes se devait de passer à l'échelle. J'ai donc réalisé des *benchmarks* [3] permettant d'évaluer si la fouille de concepts dans les corpus serait réalisable en un temps raisonnable. Cette tâche a duré une semaine. Après avoir sélectionné les algorithmes que nous allions utiliser, j'ai réalisé un environnement de test en Python permettant d'appliquer les algorithmes sur les corpus et évaluer leur efficacité.

Étant à Paris pendant la durée de mon stage, j'ai participé à quelques conférences et colloques notamment, sur le sujet du traitement des langues (TAL) et la linguistique.

2 Vocabulaire et outils pour le TAL

2.1 Définitions

Corpus Un corpus est un recueil de documents usuellement similaires (genre, longueur, langue, etc). Par exemple, un corpus peut être composé de romans français.

Lemme et Lexème Un lemme est une unité autonome constituant le lexique d'une langue, le lexème étant la forme que prend un lemme dans un texte. Le lemme peut être considéré comme la forme du dictionnaire pour un lexème donné. Ainsi, pour le lexème «mangent », on associe le lemme `manger`. On parle de lemmatisation lorsque l'on cherche à remplacer les lexèmes d'un texte par les lemmes correspondants.

Token On parle d'analyse lexicale ou tokenisation lorsque on cherche à segmenter un texte en une séquence de symboles appelés tokens. Parmi ces tokens, on retrouve les lexèmes mais aussi d'autres unités comme la ponctuation. Cette tâche est difficile car la grammaire des langages naturels est ambiguë. Pour tokeniser les textes et lemmatiser les tokens, nous utiliserons le logiciel *TreeTagger* [4][7]. Ce logiciel transforme une chaîne de caractère en liste de lexèmes (la ponctuation étant supprimée).

Vocabulaire Le vocabulaire associé à un corpus est l'ensemble des différents lemmes que l'on retrouve dans les documents du corpus. La construction du vocabulaire nécessite la lemmatisation des documents du corpus.

2.2 Les outils du TAL

Nous allons présenter ici les différents outils utilisés en TAL pour la classification de documents. Sur ce sujet, le cahier d'Elsa Negre intitulé *Comparaison de textes : quelques approches...* [6] propose une bonne introduction.

Classification

Les corpus seront représentés à l'aide d'un modèle appelé *Vector Space Model* [5]. Cela signifie que le corpus sera représenté par une matrice dont chaque ligne sera la représentation vectorielle d'un document. Une représentation vectorielle devra donc être calculée pour chaque document. Il existe plusieurs manières de procéder à la vectorisation d'un document mais la

3. Bancs d'essais. Un benchmark consiste à lancer plusieurs fois un logiciel et élaborer des statistiques sur la durée des calculs afin d'en estimer les performances.

4. TreeTagger est un tagger développé par l'université Louis-et-Maximilien de Munich. `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`.

5. `https://en.wikipedia.org/wiki/Vector_space_model`

plus évidente est de construire des vecteurs ayant la taille du vocabulaire. Chaque composante des vecteurs renseignent la fréquence des lemmes dans les documents.

La matrice constituée peut alors subir des transformations afin d'en déduire de nouvelles représentions vectorielles.

Pour estimer la proximité entre deux documents, il suffit de calculer la distance entre les deux vecteurs associés. Ainsi, les documents $\vec{u}$ et $\vec{v}$ sont d'autant plus proches que la distance les séparant est faible. En TAL, il est commun d'utiliser la similarité cosinus définie par la formule suivante :

$$\text{simcos}(\vec{u}, \vec{v}) = 1 - \frac{\vec{u}.\vec{v}}{\|\vec{u}\|\|\vec{v}\|}$$

La similarité cosinus n'est pas une véritable distance car elle n'évalue que l'angle entre les deux vecteurs. Ainsi, deux vecteurs ayant la même direction et le même sens mais possédant des normes différentes seront considérés comme identiques. L'axiome de séparation n'est pas respecté. Il s'agit d'une pseudo-distance.

Une catégorie de document peut être représentée comme le vecteur moyen des représentations vectorielles des documents qui composent cette catégorie. Supposons que nous avons à disposition un corpus C composé de n catégories $C_1, C_2 ... C_n$, formant une partition du corpus ; on définit la représentation vectorielle i d'une catégorie par la formule suivante :

$$\vec{c_i} = \frac{\Sigma_{\vec{u} \in C_i} \vec{u}}{|C_i|}$$

Pour estimer la catégorie à laquelle devrait appartenir un nouveau document non classé, nous allons comparer sa distance à chacune des catégories et choisir celle qui minimise la distance suivante :

$$\text{argmin}_{i \in [\![1,n]\!]} \{\text{simcos}(\vec{u}, \vec{c_i})\}$$

Lorsque les vecteurs sont normalisés, la formule de décision précédente se réduit à la formule suivante :

$$\text{argmax}_{i \in [\![1,n]\!]} \{\vec{u}.\vec{c_i}\}$$

Cette méthode possède la particularité d'être rapide car elle se réduit à des calculs de produits scalaires. Ces derniers sont optimisés à l'aide de la librairie *Numpy*.

Représentation vectorielle

Afin de comparer nos documents, il est primordial d'adopter une représentation vectorielle efficace. L'efficacité d'une représentation vectorielle dépend de plusieurs facteurs. Dans un premier temps, la représentation vectorielle d'un document doit être expressive et pertinente. Elle doit non seulement contenir le plus d'informations sur le document mais aussi être adaptée à la distance que nous avons choisie. En effet, on s'attend à ce que deux documents parlant d'un même sujet aient une distance faible. Dans un second temps, la représentation vectorielle choisie ne doit pas contenir trop de dimensions. En effet, une faible dimensionnalité permet non seulement d'éviter la malédiction de la dimension [6] mais aussi de réduire le temps de

6. La malédiction de la dimension désigne une famille de phénomènes contre productifs qui apparaissent uniquement lorsque l'on traite des données dans des espaces de grande dimension. Wikipédia en propose une brève introduction : `https://en.wikipedia.org/wiki/Curse_of_dimensionality`

calcul des distances. Enfin, la représentation vectorielle des documents d'un corpus doit être calculée en un temps raisonnable.

Après lemmatisation du corpus, ce qui réduit considérablement la taille du vocabulaire, nous sommes capables de savoir quels lemmes apparaissent dans un document donné. Dans une première approche, On peut attribuer à chaque document, un vecteur dont la dimension est celle du vocabulaire. La $i^{ième}$ composante du vecteur peut contenir le nombre de fois que le lemme i apparaît dans le document, c'est l'approche fréquentielle. Il est aussi possible de construire un vecteur booléen, La $i^{ième}$ composante du vecteur contient vrai si le mot apparaît au moins une fois dans le document, c'est l'approche booléenne. Ce sont des approches «Bag of words »[7]. Cette approche est relativement efficace dans le cas où la taille du vocabulaire est raisonnable.

Il est possible de représenter le corpus tout entier par la matrice composée des vecteurs lignes représentant les documents. Si on utilise les représentations précédentes pour construire la matrice du corpus, on parlera de *matrice des fréquences* ou *matrice booléenne*. Dans la suite, la matrice booléenne sera déduite de la matrice des fréquences (une valeur non nulle de la matrice des fréquences sera considéré comme le booléen *vrai*).

TFIDF

L'algorithme TFIDF *(Term frequency-Inverse document frequency)* propose d'améliorer la matrice des fréquences en apportant un poids plus grand aux termes les plus discriminants. Initialement, la matrice des fréquences est définie de la façon suivante :

$$M = (\mathrm{tf}_{i,j})$$

La quantité $\mathrm{tf}_{i,j}$ pour *term frequency* désigne le nombre d'apparitions du lemme j dans le document i. On définit la quantité positive idf_j pour *inverse document frequency* par la formule suivante :

$$\mathrm{idf}_j = -\log \frac{|\{i \in C | \mathrm{tf}_{i,j} \neq 0\}|}{|C|}$$

Le numérateur $|\{i \in C | \mathrm{tf}_{i,j} \neq 0\}|$ représente le nombre de documents dans lesquels le lemme j apparaît. Le dénominateur $|C|$ est le cardinal du corpus, c'est à dire le nombre de documents. Un lemme apparaissant dans un unique document aura un idf_j élevé tandis qu'un lemme j apparaissant dans tous les documents aura un idf_j nul. L'idf est donc un nombre que l'on accorde à un lemme permettant d'augmenter son poids s'il est discriminant et de le réduire dans le cas contraire. L'algorithme *TFIDF* retourne la matrice $\mathrm{TFIDF}(M) = (\mathrm{tf}_{i,j} \cdot \mathrm{idf}_j)$.

Analyse sémantique latente

L'algorithme LSA *(Latent Semantic Analysis)* est utilisé en traitement des langages pour réduire la dimension des matrices. La méthode s'appuie sur la décomposition en valeurs singulières qui peut être vue comme la diagonalisation de matrices non carrées. Considérons M, la matrice des fréquences. Le théorème de décomposition en valeurs singulières assure l'existence

7. Sac de mots. Le document est représenté par une liste non-ordonnée de mots. On perd alors l'information relative à la position des mots au sein des documents

d'une décomposition de la forme $M = U\Sigma V^T$ où U et V sont des matrices carrées orthogonales et Σ est une matrice de même forme que M nulle partout sauf sur sa diagonale comportant des valeurs positives non nulles appelées valeurs singulières. Le nombre de coefficients non-nuls sur la diagonale de Σ est en fait égal au rang de M. Σ peut donc être vue comme une diagonalisation de M. On peut supposer, par permutation des lignes de U et des colonnes de V^T que la diagonale de Σ est triée par ordre décroissant. Si l'on sélectionne les k plus grandes valeurs singulières ainsi que les vecteurs associés dans U et V, on obtient une approximation de rang k de la matrice M :

$$M_k = U_k \Sigma_k V_k^T$$

En considérant la matrice $U_k\Sigma_k$, on obtient une nouvelle matrice de taille $|C| \times k$ représentant les $|C|$ documents sous la forme de vecteurs lignes de dimension k. La matrice des fréquences est alors approchée en une matrice plus petite. La pertinence de cette approximation résulte du fait d'avoir sélectionné les valeurs singulières les plus grandes pour nos k composantes. On dit que nous avons réduit la matrice des fréquences à celle des k concepts [8] les plus importants.

3 Analyse formelle de concepts

L'analyse formelle de concepts (AFC) a été introduite par Rudolf Wille en 1982 [8]. Un contexte est un triplet (G, M, I) où G et M sont des ensembles et $I \subseteq G \times M$:

— G est l'ensemble des objets.
— M est l'ensemble des attributs.
— Soit $(g, m) \in G \times M$, la propriété $(g, m) \in I$ indique que l'objet g possède l'attribut m. L'ensemble I peut donc être vu comme une relation. On note alors $gIm \Leftrightarrow (g, m) \in I$.

On définit deux opérateurs de dérivation pour $A \subseteq G$ et $B \subseteq M$ par $A' = \{m \in M | \forall g \in A, gIm\}$ et $B' = \{g \in G | \forall m \in B, gIm\}$.

Un concept du contexte (G, M, I) est alors un couple (A, B) où $A \subseteq G$ et $B \subseteq M$ vérifient $A' = B$ et $B' = A$. A est dénommé l'extension car il contient les objets qui représentent physiquement ce concept. B est dénommé l'intension car il contient les attributs qui représentent le concept [9]. Nous avons représenté en figure 1 le contexte des nombres entiers de 1 à 10 pour les attributs *composé*, *pair*, *impair*, *premier*, *carré*. Dans ce contexte, le couple $(\{3, 5, 7\}, \{\text{impair}, \text{premier}\})$ forme un concept, celui des nombres premiers de l'intervalle $[\![1, 10]\!]$.

Dans notre étude, nous allons considérer un contexte formel formé à partir d'un corpus. L'ensemble G serait composé des documents du corpus tandis que l'ensemble M serait composé du vocabulaire du corpus. On note gIm lorsque le mot m se trouve dans le document g.

La méthode traditionnelle est d'utiliser la matrice des fréquences ou la matrice booléenne comme source d'information pour la classification. Dans cette méthode les lignes sont des documents et les colonnes des lemmes. Nous allons plutôt étudier les utilisations possibles de la *matrice des concepts* où les colonnes sont des concepts. Cela signifie que nous allons représenter les documents par des vecteurs de concepts. Chaque composante étant à vrai si le document contient le concept correspondant.

8. Attention, la notion de concept dans l'algorithme LSA n'est pas la même que celle des concepts formels. Ici, un concept est une combinaison linéaire et stochastique de lemmes.

9. En logique l'utilisation d'intension et d'extension est courante pour décrire un concept : `https://fr.wikipedia.org/wiki/Intension_et_extension`.

	composé	pair	impair	premier	carré
1			x		x
2		x		x	
3			x	x	
4	x	x			x
5			x	x	
6	x	x			
7			x	x	
8	x	x			
9	x		x		x
10	x	x			

FIGURE 1 – Un contexte formel pour des nombres entiers. Un des concepts est mis en évidence.

4 Méthodologie et évaluation

Pour évaluer la performance d'une classification automatique, nous utiliserons la *f-mesure*.

Après avoir construit une représentation vectorielle quelconque pour les documents, nous obtenons une matrice M de taille $|C| \times k$ représentant les $|C|$ documents du corpus sous la forme de vecteurs lignes de taille k.

Pour tester l'efficacité de notre représentation vectorielle, nous allons procéder comme suit. Premièrement, nous séparons le corpus aléatoirement en deux parties distinctes. L'une $C_{80\%}$ comportera 80% des documents et l'autre $C_{20\%}$ les 20% restants. Nous utiliserons les 80% pour réaliser les vecteurs représentants les catégories :

$$\forall i \in [\![1, n]\!] : \qquad \vec{c_i} = \frac{\Sigma_{\vec{u} \in C_i \cap C_{80\%}} \vec{u}}{|C_i \cap C_{80\%}|}$$

C'est grâce à ces représentant que l'on déduit une fonction de décision calculant la catégorie minimisant les pseudo-distances cosinus.

L'exercice pour l'ordinateur est alors d'utiliser ces représentants afin de classer les 20% restants. Pour chaque catégorie i, on définit la précision et le rappel comme suit :

$$\text{précision}_i = \frac{\text{nombre de documents correctement attribués à la classe } i}{\text{nombre de documents attribués à la classe } i}$$

$$\text{rappel}_i = \frac{\text{nombre de documents correctement attribués à la classe } i}{\text{nombre de documents appartenant à la classe } i}$$

La précision globale et le rappel global sont obtenus en calculant les moyennes. La f-mesure est définit comme étant la moyenne harmonique de la précision et du rappel qui est un nombre compris entre 0 et 1.

$$F = 2 \cdot \frac{\text{précision} \cdot \text{rappel}}{\text{précision} + \text{rappel}}$$

Notre classification est d'autant meilleure que le nombre F est proche de 1. Nous réitérerons l'expérience afin d'évaluer la f-mesure sur d'autres séparations aléatoires du corpus. Nous calculons la moyenne et l'écart-type des f-mesures.

L'efficacité du modèle est évaluée à partir de documents qui n'ont pas été utilisés pour l'entraînement du modèle. Cette méthode de séparation du corpus en deux parties permet donc d'isoler la vérification de l'entraînement et constitue un procédé fiable pour l'évaluation d'un modèle.

Le schéma globale est représenté figure 2.

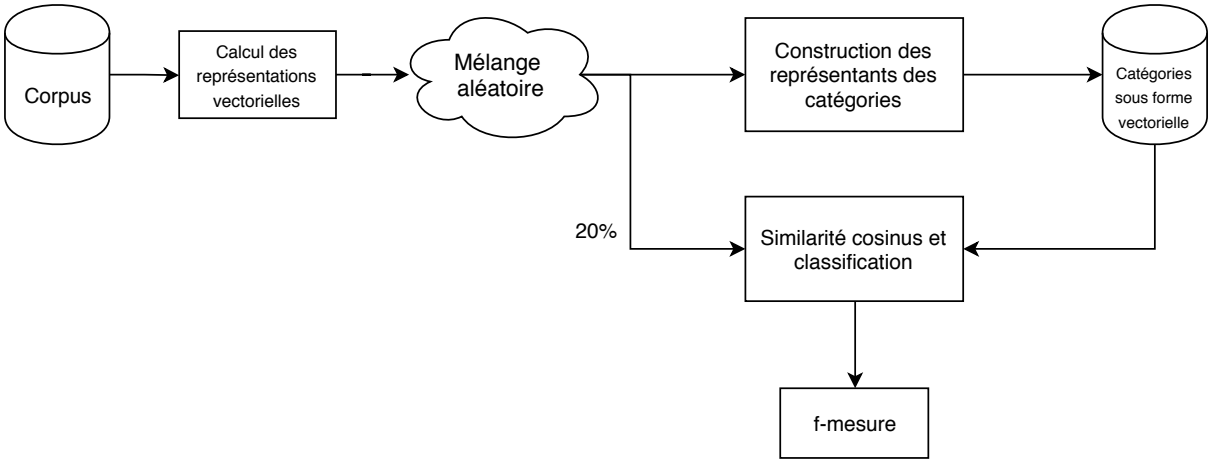

FIGURE 2 – Schéma de l'évaluation d'un classificateur.

5 Travail effectué

5.1 Recherche bibliographique

La première étape a été une recherche bibliographique afin de comprendre les fondements mathématiques de l'AFC. Le livre *Formal Concept Analysis : Mathematical Foundations* [2] propose une approche mathématique rigoureuse et complète des concepts formels. l'AFC hérite de la théorie des treillis de Gallois, ce qui lui offre une multitude de propriétés intéressantes pour le traitement des langages. La lecture de ce livre permet une compréhension des concepts formels et apporte rapidement les intuitions afin de construire un premier algorithme naïf d'extraction de concepts dans les contextes. De nombreuses recherches ont permis de découvrir de nouveaux algorithmes plus performants.

Pour notre étude nous avons sélectionné le programme *PCBO (Parallel Cbo)* [4]. Il s'agit de l'implémentation en langage *C* de l'algorithme du même nom. Cet algorithme est récursif et massivement parallèle ce qui lui offre une exécution rapide sur les ordinateurs possédant un grand nombre de processeurs. Le programme reçoit une matrice représentant un contexte (ici il s'agira de la *matrice booléenne* représentant un corpus) et retourne la matrice des concepts et des intensions.

Cette dernière est une matrice booléenne de dimension $k \times |V|$ où k est le nombres de concepts extraits et $|V|$ est le cardinal du vocabulaire. Si l'on note cette matrice $M = (m_{i,j})$, $m_{i,j}$ vaut 1 si et seulement si le lemme j fait partie du concept i. Nous ne détaillerons pas ici le fonctionnement de cet algorithme car notre étude porte sur l'efficacité de l'utilisation des concepts formels dans la classification de documents. Ce qui nous intéresse ici est son temps de calcul.

5.2 Corpus choisi

Les documents que nous avons choisis pour notre étude sont les *abstracts* en langue française des articles de l'encyclopédie en ligne Wikipédia. Nous pouvons les obtenir classés par ontologies [10] en requêtant la base de donnée *DbPedia* via le langage *SPARQL*[3]. Nous allons nous restreindre à l'ontologie *animaux* qui contient les articles Wikipédia traitant des animaux. Cette dernière est composée de plusieurs sous-ontologies que nous utiliserons comme catégories de documents lors de nos essais de classification [11]. Nous allons mettre notre méthode à l'épreuve sur le problème de la classification d'articles animaliers.

Dans toutes les utilisations de ces documents, nous supposons avoir lemmatisé le texte à l'aide de Tree-Tagger et supprimé les hapax [12]. Cette transformation offre non seulement une réduction considérable de l'étendue du vocabulaire du corpus mais aussi une réduction de la taille des documents. En effet, chaque abstract de l'ontologie *animaux* est constitué d'une moyenne de trente mots (après lemmatisation et suppression des hapax).

Pour les essais de classification, nous ne conserverons pas les ontologies des *arachnides*, des *crustacés* et des *mollusques* car l'effectif de leur documents n'est pas suffisant pour effectuer des statistiques. En ne conservant que les documents ayant plus de trois-cents caractères, nous obtenons un corpus de quatre-mille abstracts répartis en six catégories d'animaux et pour un vocabulaire de quatre-mille cinq cents lemmes.

5.3 Efficacité des logiciels

5.3.1 Contextes aléatoires

Nous ne pouvons pas prévoir la vitesse d'exécution de l'algorithme choisi car le nombre de concepts à extraire dépend fortement du contexte formel étudié. Dans un premier temps, nous allons générer des contextes aléatoires. Pour cela, nous définissons un nombre d'objets et un nombre d'attributs. Ensuite, l'évènement «l'objet i possède l'attribut j »suivra une loi uniforme avec la probabilité un demi.

Nous allons exécuter le logiciel PCBO sur plusieurs contextes aléatoires en utilisant une machine équipée de 32go de mémoire vive et 16 processeurs Intel©Xeon©E5-2690 v3 candencés à 2.60GHz. Cette exécution sera comparée à une exécution de PCBO restreinte à un unique processeur afin d'évaluer le gain apporté par l'exécution parallèle. Nous faisons varier le nombre de documents du contexte pour une taille de vocabulaire fixée (figure 3). Ensuite, nous faisons varier la taille du vocabulaire pour un nombre de documents fixé (figure 4).

Nous observons non seulement le gain apporté par la parallélisation mais aussi l'évolution rapide du temps de calcul nécessaire lorsque la taille du contexte augmente. Cependant l'approche que nous avons utilisée n'est pas représentative de la réalité.

10. Une ontologie peut être vue comme une catégorie d'article[1]. La liste des ontologies est lisible à l'adresse suivante : `http://mappings.dbpedia.org/server/ontology/classes/`.

11. Les sous-ontologies sont : amphibiens, arachnides, oiseaux, crustacés, poissons, insectes, mammifères, mollusques et reptiles.

12. Les hapax sont des mots qui n'apparaissent que dans un seul document du corpus. Ils ne permettent pas de relier des documents ensembles et sont donc inutiles dans les problèmes de classification.

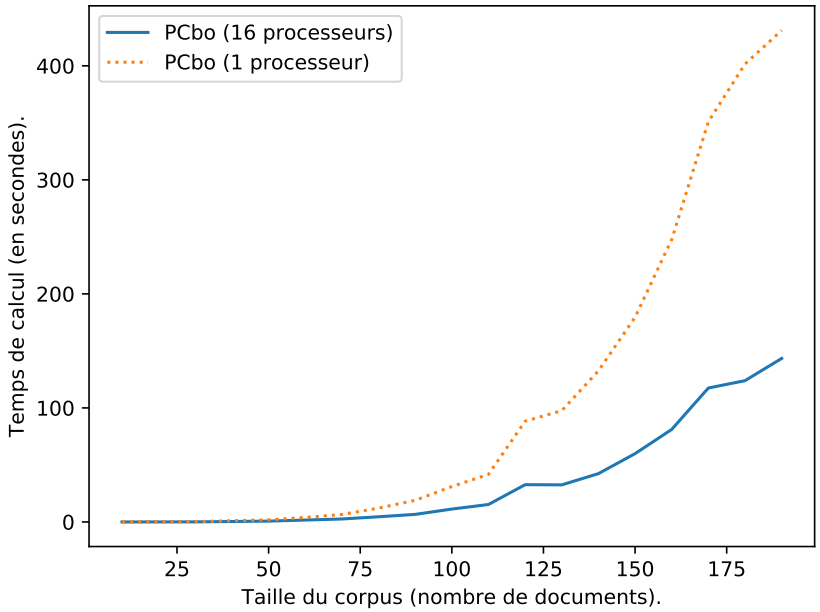

FIGURE 3 – Évolution du temps de calcul en fonction du nombre de documents (taille du vocabulaire fixée à 200 lemmes).

5.3.2 Contextes basés sur un corpus réel

Dans la réalité, les documents écrits en langues naturelles suivent une distribution statistique très particulière qui a été étudiée par George Kingsley Zipf[5]. La loi de Zipf stipule que, dans un texte, la fréquence d'apparition du j$^{\text{ième}}$ mot le plus fréquent est de la forme $\frac{K}{j}$ où K est une constante de normalisation commune à tout les mots. Dans notre cas où le cardinal du vocabulaire est m, la relation $\sum_{j=1}^{m} \frac{K}{j} = 1$ impose $K = H_m^{-1}$ où H_m est le nombre harmonique de rang m. En conséquence, la matrice booléenne représentant le corpus est très creuse, il y a donc moins d'informations à traiter qu'à l'étude précédente.

Nous allons réaliser nos calculs de performance sur les abstracts de l'ontologie *Animaux* que nous utiliserons dans nos essais de classification. Pour cela, nous allons lancer le logiciel PCBO sur les premiers documents de l'ontologie puis nous allons augmenter graduellement la taille de l'échantillon. Nous allons ainsi faire apparaître l'évolution de certains paramètres en fonction du nombre de documents.

Nous constatons que, malgré un vocabulaire plus riche, l'algorithme PCBO s'exécute plus rapidement sur des documents réels (figure 5). Nous pouvons traiter un corpus de cinq mille documents en deux minutes, ce qui est convenable à notre échelle. Nous pouvons aussi constater que le nombre de concepts extraits est très grand (figure 6). Il semble maladroit de vouloir utiliser la similarité cosinus sur les vecteurs documents exprimés dans l'espace des concepts formels. Il sera primordial d'adopter des méthodes pour réduire le nombre de concepts et sélectionner les plus intéressants.

L'évolution de la richesse du vocabulaire en fonction du nombre de documents est représenté en figure 7.

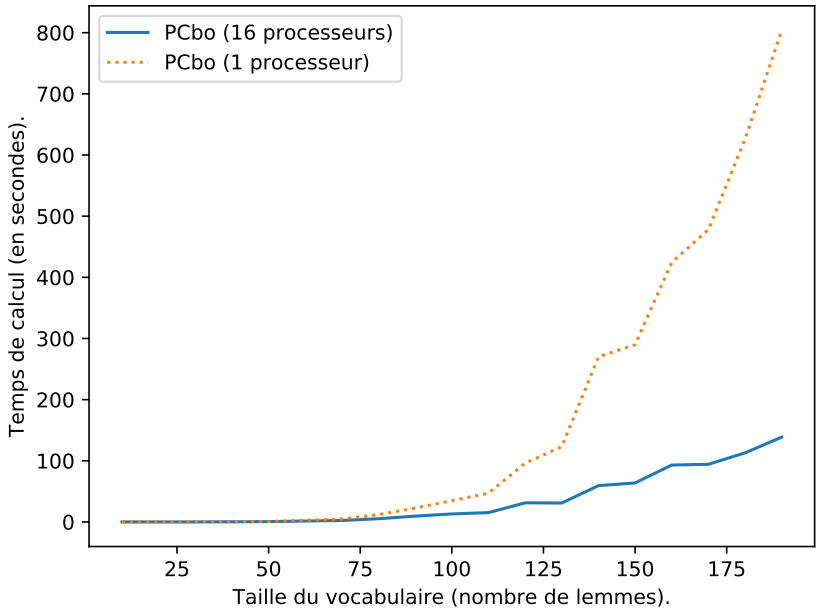

FIGURE 4 – Évolution du temps de calcul en fonction de la taille du vocabulaire (nombre de documents fixé à 200).

5.4 Implémentation d'un environnement de test

Pour tester l'efficacité de l'utilisation des concepts formels dans la classification des documents, nous avons réalisé un environnement de test en *Python*. Cette implémentation se divise en deux classes. La classe `Corpus` permet de créer un objet contenant toutes les informations nécessaires et exploitables d'un corpus. La classe `Classify` prend un objet de type `Corpus` et construit une interface permettant de tester et comparer les différentes méthodes de classification sur le corpus donné.

La classe `Corpus`

La classe `Corpus` doit permettre la construction et la sauvegarde d'un corpus. Elle doit aussi permettre l'exploitation de l'information contenue dans un corpus afin de tester les algorithmes de classification sur un corpus. Un objet `Corpus` possède les attributs suivants :

- Deux listes de longueur $|C|$ contenant le titre de l'article ainsi que l'abstract de chaque article Wikipédia du corpus.
- Une liste de longueur n contenant les noms des catégories auxquelles peuvent appartenir les documents.
- Une matrice booléenne de taille $|C| \times n$ permettant de savoir à quelles catégories appartient un document. Dans notre cas, le corpus est partitionné et un document ne peut faire partie que d'une seule catégorie.
- Une liste de longueur $|\mathcal{V}|$ contenant le vocabulaire contenu dans le corpus. Les éléments du vocabulaire sont des lemmes.
- Une matrice des fréquences de taille $|C| \times |\mathcal{V}|$ qui donne le nombre d'occurrences de chaque lemme dans chaque document.

Les deux derniers attributs sont calculés à partir des attributs précédents. L'extraction des

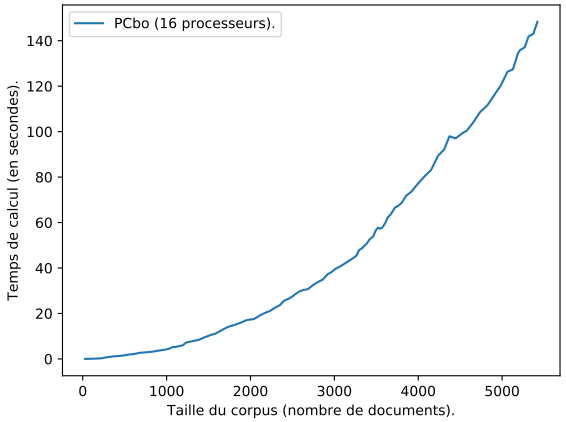

FIGURE 5 – Évolution du temps de calcul en fonction du nombre de documents choisis dans l'ontologie *Animal*.

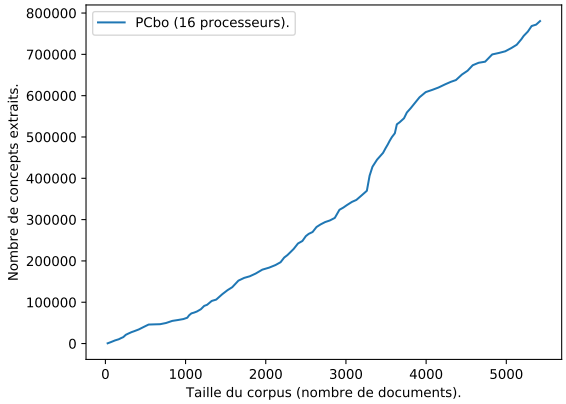

FIGURE 6 – Évolution du nombre de concepts extraits fonction du nombre de documents choisis dans l'ontologie *Animal*.

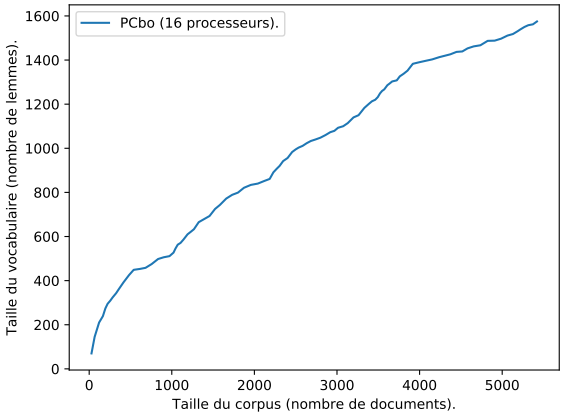

FIGURE 7 – Évolution de la taille du vocabulaire en fonction du nombre de documents choisis dans l'ontologie *Animal*.

lemmes contenus dans un document se fait via l'outil *TreeTagger*. Les matrices sont creuses et sont donc représentées en mémoire par une structure proche de celle des listes d'adjacences utilisées pour les graphes. [13]

La construction d'un objet `Corpus` peut se faire selon trois méthodes. La première méthode est l'ouverture d'un corpus précédemment sauvegardé. La seconde méthode est la construction d'un corpus via trois listes et une matrice qui sont les quatre premiers attributs que nous avons présentées. Enfin, il est possible de construire un corpus via une liste d'ontologies. Les abstracts de chaque ontologie seront téléchargés. Chaque ontologie sera considérée comme une catégorie de documents.

Plusieurs méthodes de la classe `Corpus` permettent d'accéder à des informations sur un corpus. Ces méthodes permettent à la classe `Classify` de traiter le corpus et lancer les algorithmes de classification.

La classe `Corpus` propose la sauvegarde d'un corpus sous le format d'une archive *tar*. Ce

13. Pour la représentation des matrices creuses nous utiliserons le module `Sparse matrix` de la librairie `SciPy`.

format permet d'enregistrer toutes les informations d'un corpus en un seul fichier, ce qui est commode pour archiver nos corpus et les réutiliser plus tard. Il est possible d'écrire un texte *Readme* qui sera stocké dans l'archive tar afin de conserver quelques informations sur le corpus comme la source des documents, le contenu ainsi que la date de création.

La classe `Classify`

La classe `Classify` permet de créer un classificateur à partir d'un objet de type `Corpus`. Le classificateur possède des méthodes permettant d'appliquer les algorithmes matriciels (tels que TFIDF, LSA et PCBO) sur la matrice des fréquences. Il est possible d'appliquer ces algorithmes en chaîne, les uns après les autres, afin de transformer la matrice des fréquences et obtenir la représentation vectorielle souhaitée pour les documents. Une fois ces algorithmes appliqués, il est possible de calculer les représentants des catégories. Pour cela, le classificateur choisit 80% des documents de façon complètement aléatoire qui serviront à la construction des représentants. Enfin, le classificateur tente, en calculant les distances aux catégories, de classer les 20% restants. La matrice de confusion [14] ainsi que la f-mesure associée sont calculées et affichées dans un rapport de traitement au format pdf.

Devant le très grand nombre de concepts extraits, nous avons du mettre en place des méthodes permettant d'en réduire le nombre. Dans un premier temps, il est envisageable d'appliquer l'algorithme LSA à la matrice issue de l'algorithme PCBO. Cependant, cette méthode n'est pas satisfaisante dans la mesure où appliquer l'algorithme LSA nous fait quitter le domaine des concepts formels. En effet, nous avons comme intuition que certains concepts formels se rapprochent des catégories dans lesquelles on cherche à classer nos documents. Par exemple, on peut s'attendre à ce qu'il existe un concept formel très proche du concept des mammifères.

L'idée est donc de sélectionner les concepts susceptibles de représenter une catégorie. De la même manière qu'un hapax, un concept qui apparaît dans un unique document n'est pas intéressant. Enfin, un concept qui apparaît dans la plupart des documents n'est pas discriminant et n'apporte donc pas d'information. Nous avons étendu la classe `Classify` pour qu'elle puisse supprimer ces concepts.

6 Procédés et résultats

6.1 La méthode classique

Une méthode classique et couramment utilisée en Traitement Automatique du Langage afin d'obtenir une bonne représentation vectorielle pour les documents est d'appliquer une fois l'algorithme TFIDF puis l'algorithme LSA en choisissant 300 composantes. Les calculs associés à ces transformations sont très rapides, mille essais ont été réalisés en vingt-deux minutes. Sur mille essais, nous obtenons une f-mesure moyenne de 0,94 (pour un écart-type de 0,018) ce qui démontre que le classement est satisfaisant. Lorsque TFIDF est utilisé seul, sans application de LSA, la f-mesure est de 0.6. Ce résultat justifie l'utilisation de l'algorithme LSA. L'obtention d'une f-mesure différente pour chaque essai provient de la séparation aléatoire du corpus en 80% et 20%. La matrice de confusion nous montre les erreurs commises par la classification

14. La matrice de confusion permet d'avoir connaissance des erreurs commises par la classification. Un exemple se trouve plus loin en figure 8.

(figure 8). Enfin l'histogramme en figure 9 nous montre la répartition des f-mesures sur mille essais.

Réel \ Estimé	**Amphibien**	**Oiseau**	**Poisson**	**Insecte**	**Mammifère**	**Reptile**
Amphibien	115	1	0	0	0	2
Oiseau	0	312	3	7	10	1
Poisson	0	1	55	2	4	1
Insecte	0	1	0	73	1	2
Mammifère	0	0	1	3	72	2
Reptile	0	0	0	0	0	82

FIGURE 8 – Matrice de confusion lors d'un essai avec TFIDF et LSA.

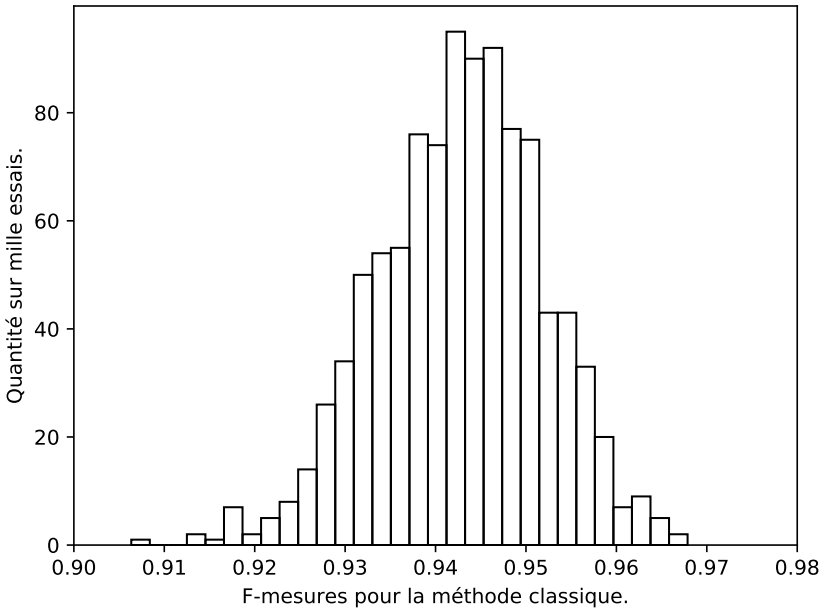

FIGURE 9 – Histogramme de la répartition des f-mesures pour mille essais avec TFIDF et LSA.

La méthode de l'application de TFIDF et LSA est simple, très rapide, et donne des résultats satisfaisants. Nous comparerons donc les résultats de cette méthode à ceux issus de l'utilisation des concepts formels.

6.2 L'analyse formelle de concepts

Pour mettre en place l'utilisation des concepts formels, nous avons procédé comme suit. Premièrement, nous extrayons les concepts formels du corpus à l'aide de l'algorithme PCBO et calculons ainsi la matrice des vecteurs documents représentés dans l'espace des concepts formels. Ensuite, nous calculons les représentants des catégories de documents dans l'espace des concepts formels en moyennant les vecteurs documents pour chaque catégorie. Ces vecteurs sont très grands car les concepts sont très nombreux (comme nous l'avions vu en figure 6).

Pour réduire la taille des vecteurs, nous supprimons tous les concepts partagés par plusieurs catégories de documents (dans notre étude, vingt documents étaient le seuil apportant les meilleurs résultats). Nous supprimons aussi les concepts partagés par trop peu de documents au sein d'une catégorie. Les concepts subsistants sont exclusifs à une catégorie et représentés par beaucoup de documents de cette dernière. Avec cette heuristique, nous obtenons quatre-mille concepts. Les calculs associés à cette technique sont longs, le programme a fonctionné pendant cinq heures pour effectuer mille essais. La f-mesure moyenne associée à ces essais est de 0,89 (avec un écart type de 0,014).

La matrice de confusion nous montre les erreurs commises par la classification (figure 10). Enfin l'histogramme en figure 11 nous montre la répartition des f-mesures sur mille essais.

Réel \ Estimé	**Amphibien**	**Oiseau**	**Poisson**	**Insecte**	**Mammifère**	**Reptile**
Amphibien	114	0	0	0	0	5
Oiseau	0	303	3	3	7	1
Poisson	0	0	50	0	1	0
Insecte	0	0	1	52	5	1
Mammifère	0	5	3	2	52	3
Reptile	0	0	2	1	0	69

FIGURE 10 – Matrice de confusion lors d'un essai avec les concepts formels.

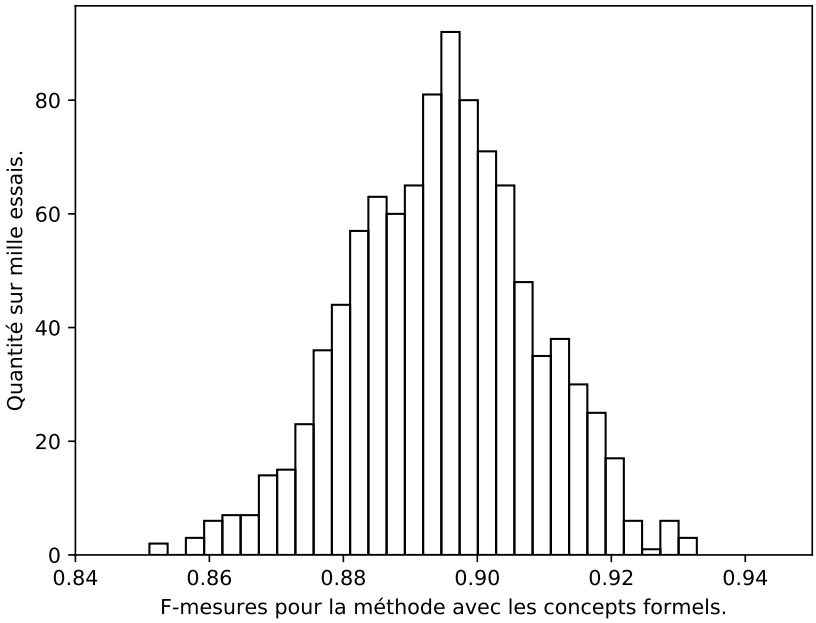

FIGURE 11 – Histogramme de la répartition des f-mesures pour mille essais avec les concepts formels.

6.3 Observations

La classification que nous avons mis en œuvre avec les concepts formels est plus lente que la méthode classique et apporte de moins bons résultats. Le temps de calcul nécessaire à notre méthode provient de l'extraction d'un très grand nombre de concepts.

7 Conclusion

La f-mesure calculée avec notre méthode des concepts formels n'est pas mauvaise, comparée à celle obtenue par utilisation de TFIDF seul. Elle montre que l'extraction de concepts formels peut apporter des informations utiles et discriminantes sur les documents. Nous avons expérimenté un schéma particulier mais d'autres méthodes peuvent, peut-être, être mises en place, afin d'exploiter plus efficacement les concepts formels.

Notre méthode utilise des vecteurs afin de sélectionner les catégories qui minimisent des distances. Les concepts formels étant booléens (un document contient ou non un concept), nous avons pensé à aborder une approche de type logique. En assimilant les concepts à des variables de la logique des prédicats, nous pourrions associer une formule logique à chaque catégorie. Les documents seraient assimilés à des valuations pouvant satisfaire les formules précédentes. Cette méthode nous semblait intéressante mais nous n'avons pu l'expérimenter.

Enfin, nous avons appliqué notre méthode sur un corpus relativement petit. En effet, la longueur d'un abstract d'un article de Wikipédia est faible. Notre méthode devient inutilisable lorsque l'on souhaite l'appliquer à un corpus plus grand et plus riche. Beaucoup de concepts sont calculés mais la plupart sont supprimés après analyse des représentants des catégories. Nous pensons que le temps de calcul associé à l'extraction des concepts formels peut être réduit avec une heuristique qui permettrait de calculer uniquement les concepts intéressants.

Références

[1] Martin Brümmer, Milan Dojchinovski, and Sebastian Hellmann. Dbpedia abstracts : A large-scale, open, multilingual nlp training corpus. In *LREC*, 2016.

[2] Bernhard Ganter and Rudolf Wille. *Formal concept analysis : mathematical foundations*. Springer Science & Business Media, 2012.

[3] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. Sparql 1.1 query language. *W3C recommendation*, 21(10), 2013.

[4] Petr Krajca, Jan Outrata, and Vilem Vychodil. Parallel recursive algorithm for fca. In *CLA*, volume 2008, pages 71–82. Citeseer, 2008.

[5] Benoît Mandelbrot. Etude de la loi d'estoup et de zipf : fréquences des mots dans le discours. *Logique, langage et théorie de l'information*, 1957.

[6] Elsa Negre. Comparaison de textes : quelques approches... working paper or preprint, April 2013. URL `https://hal.archives-ouvertes.fr/hal-00874280`.

[7] Helmut Schmid. Treetagger| a language independent part-of-speech tagger. *Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart*, 43 :28, 1995.

[8] Rudolf Wille. Restructuring lattice theory : An approach based on hierarchies of concepts. In Ivan Rival, editor, *Ordered Sets*, pages 445–470, Dordrecht, 1982. Springer Netherlands. ISBN 978-94-009-7798-3.