DÉSAMBIGUÏSATION LEXICALE SEMI-SUPERVISÉE

Document préparé par :

HO Chih-Yang, KOMADINA Santiago

Date :

23 juin 2020

1. INTRODUCTION

1.1. TERMINOLOGIE

1.1.1. L'*APPRENTISSAGE SUPERVISÉ*, *NON SUPERVISÉ* ET *SEMI-SUPERVISÉ*

On parle d'apprentissage supervisé quand les données sont étiquetées. Ce type d'apprentissage demande plus de données pour que l'on puisse généraliser le motif des données. Par exemple, cela peut permettre de créer un système pour distinguer les e-mails qui sont spam et ceux qui ne le sont pas à la base des données étiquetées. Contrairement à l'apprentissage supervisé, l'apprentissage non supervisé traite des données qui ne sont pas étiquetées. Ce type d'apprentissage permet de découvrir un motif à partir des données non étiquetées. En général, l'apprentissage non supervisé nous permet de faire des taches plus compliquées que l'apprentissage supervisé. Par exemple, il peut permettre de classifier les clients en différentes catégories en fonction de leur comportement d'achat. Une autre technique, qui combine les deux types d'apprentissages décrits ci-dessus, est ce que l'on appelle l'apprentissage semi-supervisé, c'est à dire qu'il utilise en grande partie des données non étiquetées, mais aussi un nombre limité de données étiquetées. L'intérêt d'utiliser cette méthode est que nous pouvons avoir un meilleur résultat tout en utilisant un petit nombre de données étiquetées.

1.1.2. *VECTORISATION*

Étant donné que la machine ne peut pas comprendre la langue naturelle, si l'on souhaite que la machine traite les données des langues naturelles, il faut commencer par transformer les traits en vecteurs. La vectorisation est une méthode qui permet à la machine d'analyser les langues naturelles car la machine ne traite que les chiffres.

3. *CLUSTERING*

Data Clustering est aussi ce que l'on appelle « partitionnement de données » en français. C'est une méthode qui permet d'analyser les données numériques et vectorisées. Elle a pour but de partitionner les données en un certain nombre de groupes par leur similarité. Le but est donc que les données avec des traits similaires soient regroupées dans le même groupe. En général, les données qui sont regroupées dans un même groupe sont proches (dans l'espace) l'une de l'autre. C'est une méthode d'apprentissage non-supervisé, c'est à dire que l'on ne connaît pas l'étiquette des données. Une fois que l'on a réalisé le clustering, on a besoin d'associer le sens de chaque groupe.

1.1.4. *PRÉCISION* (ACCURACY)

Accuracy est un mot en anglais pour évaluer la performance d'un système. L'accuracy se calcule en faisant la somme du nombre de cas de vrai positif (ce qui est positif et a été observé positif) et du nombre de cas de vrai négatif (ce qui est négatif et a été observée négatif), que l'on divise par le nombre total d'échantillons observés. L'accuracy nous permet d'avoir le pourcentage de cas où le système a correctement prédit.

1.1.5. *NGram*

Un NGram est une sous-séquence de N éléments construits par une séquence donnée. Par exemple, Uni-Gram de la phrase « Ses murailles ont été abattues en 1958. » est ['Ses', 'murailles', 'ont', 'été', 'abattues', 'en', '1958']. C'est à dire, Uni-Gram est bien tous les mots dans une liste. Si on prend Bi-Gram de la même phrase, on va avoir une liste ['Ses murailles', 'murailles ont', 'ont été', 'été abattues', 'abattues en', 'en 1958']. Il y a N-1 éléments deux mots de la phrase.

La vectorisation par NGram est ainsi une façon d'apprendre à la machine le sens d'un mot à partir de son contexte. Pour distinguer les différents sens d'un verbe on peut voir son contexte. Par exemple, « We need to book out ticket soon. » et « We need to read that book soon », « book » apparaît sur toutes les deux phrases mais il porte un sens totalement diffèrent.

1.2. CONTEXTE

La désambiguïsation lexicale consiste à choisir le sens d'une graphie dans un contexte donné parmi un ensemble de sens possibles (cet inventaire de sens étant prédéfini). On considère en général un texte lemmatisé, et on ne traite que des ambiguïtés de sens pour un même lemme (par ex. "voler" = "planer" ou bien "dérober"). La difficulté de cette tâche est d'une part que les indices permettant la désambiguïsation sont très variés, d'autre part qu'une approche purement supervisée suppose un trop grand nombre de données: on ne peut pas envisager disposer de beaucoup d'exemples désambiguïsés pour chaque mot ambigu du lexique.

On propose ici d'utiliser la technique semi-supervisée, qui s'appuie pour désambiguïser un lemme donné, sur quelques exemples annotés (=désambiguïsés) plus beaucoup d'exemples non annotés.

Il s'agira, pour un lemme ambigu donné, de:

- Représenter une occurrence sous forme de traits, d'après le contexte linéaire et/ou syntaxique de l'occurrence

- Réaliser une induction de sens (indépendamment des sens existants) en appliquant un clustering sur un ensemble d'occurrences à désambiguïser. Un clustering k-means ou hiérarchique seront testés.
- Puis investiguer comment contraindre le clustering pour utiliser les sens existants et les exemples annotés.

1.3. OBJECTIFS DU PROJET

Un verbe peut avoir plusieurs sens selon les différents contextes. Par exemple, pour le verbe « abattre », on peut avoir 5 sens, il peut avoir les sens suivants : « faire tomber », « tuer un être vivant », « causer un état mental », « accomplir » et « faire descendre sur quelque chose ». Pour nous, êtres humains, on peut distinguer facilement ce qui signifie le verbe dans un certain contexte. Cependant, ce n'est pas le cas pour la machine. La machine ne comprend que les nombres et se contente de réaliser des calculs. Donc il faut transformer les différents sens dans les différents contextes en une représentation qui ait du sens non pour nous mais pour la machine. Et sur cette base chercher à identifier les traits éventuellement pertinents dans la tache de désambiguïsation.

Ce que nous allons faire c'est permettre à la machine de prédire le sens d'un verbe dans un contexte (une phrase). L'objectif de ce projet est donc de désambiguïser les sens d'un verbe par les traits et des moyennes de clustering.

Le processus pour trouver la meilleure façon de désambiguïsation est décris ci-dessous :

1. Tout d'abord, on chercher les traits potentiellement pertinents et les transformant en vecteur (vectorisation)
2. On fait le clustering par les traits potentiellement pertinents
3. Ensuite, on compare les clusters prédits et les clusters réels
4. Enfin, on obtient un pourcentage d'accuracy

On répète le processus plusieurs fois pour obtenir le meilleur résultat. Autrement dit, on va tenter plusieurs traits potentiellement pertinents pour avoir le meilleur accuracy, c'est également ce qui nous permettra d'avoir une meilleure performance pour différentier les différents sens d'un verbe.

2. HYPOTHÈSE ET MÉTHODES

2.1 HYPOTHÈSES

Notre objectif est de réunir par groupe de sens les représentations vectorielles d'un même mot dans des contextes différents. Idéalement nous voudrions que les vecteurs des occurrences d'un même mot partageant un sens soient rapprochés dans leur espace vectoriel.

Choisir le “bon sens” d’une occurrence consisterait alors à comparer son vecteurs avec ceux de chacun des groupe de sens possible et sélectionner celui qui lui est le plus proche. Cependant, toute la difficulté de la tâche consiste à trouver une représentation vectorielle qui puisse permettre cette différence. Se pose alors une question fondamentale, ayant une phrase avec un verbe à désambiguïser, quelles informations présentes dans la phrase allons-nous chercher à représenter ? Nous avons décidé de nous baser sur l’hypothèse distributionnaliste : le sens d’un mot est indiqué par les mots qui l’accompagnent. Nous supposons alors que le contexte linéaire (mots avant et après notre verbe dans la phrase) et syntaxique (éléments qui ont une relation de dépendance syntaxique avec notre verbe) doivent garder des indications concernant le sens du verbe.

2.2 MÉTHODE

Nous avons testé plusieurs représentations vectorielles et algorithmes de clustering sur les différents exemples pour chacun de nos verbes et nous avons chercher les meilleurs combinaisons possibles. Notre procédure s’est divisé en trois grandes parties :

1) La création d’exemples à partir des données fournies.

2) La vectorisation des exemples

3) Le clustering + évaluation

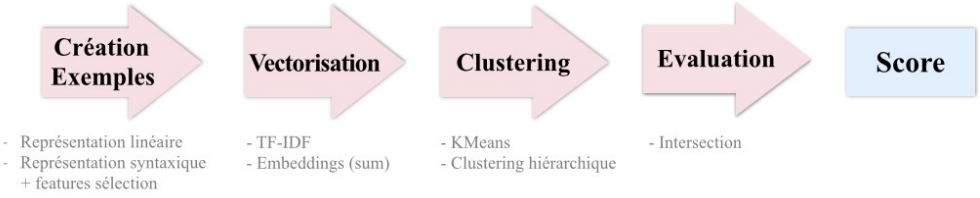

Figure 1: Pipeline représentant le projet

2.2.1 CRÉATION D’EXEMPLES

Deux types d’exemples ont été envisagés : les exemples linéaires (on crée un contexte linéaire de n mots/traits avant et après notre verbe à désambiguïser) et les exemples syntaxiques (on crée l’arbre de dépendance syntaxique associée à notre phrase en sélectionnant les traits qui nous intéressent). Parmi les traits exploitables lors des tests cinq possibilités existent pour chaque élément de la phrase à analyser : le mot-forme, le lemme, le

POS tagging normal, le POS tagging profond et la relation de dépendance syntaxique du mot avec l'élément qui le régit.

2.2.1.1 Représentation linéaire

Dans cette partie, le but est de générer des vecteurs correspondant aux traits des lemmes et des POS (Part of Speech). Le lemme est la forme non fléchie d'un mot, par exemple, le lemme de « abattues » est « abattre ». POS, également appelé étiquetage morphe-syntaxique, permet de donner aux mots une catégorie en fonction de leur fonction grammaticale. Par exemple, « abattre » est dans la catégorie de verbe, « année » est dans la catégorie de nom.

Pour commencer, on prend le mot ciblé « mot i » et sa demie-fenêtre de 2 mots qui le précèdent (« mot i-2, mot i-1 ») et qui le suivent (« mot i+1, mot i+2 »). On considère donc 2 mots avant le mot ciblé et 2 mots après le mot ciblé. Par exemple, pour la phrase *« Ses murailles ont été abattues en 1958.»*, on a « avoir être abattre en 1958 » en lemme et « V V V P N » en POS avec une demie-fenêtre de 2.

Exemple : *« Ses murailles ont été abattues en 1958.»*	
Lemme d'exemple : « son muraille avoir être abattre en 1958. » POS d'exemple : « D N V V V P N PONCT »	
Représentation linéaire avec le mot ciblé en demie-fenêtre de 2 : « abattre »	
En lemme	« avoir être abattre en 1958 »
En POS	« V V V P N »

Tableau 1 : Exemple de la représentation linéaire avec le mot ciblé « abattre »

2.2.1.2 REPRÉSENTATION syntaxique

Il s'agit cette fois de construire les arbres de dépendance syntaxique pour une phrase associée à un verbe donnée. Une fois l'arbre créé on choisit quel *feature* ont va tester. La figure XX montre l'arbre de dépendance syntaxique pour l'exemple donné plus haut, le *feature* sélectionnée est dans ce cas le mot fléchit. Ici, notre verbe conforme le noyau syntaxique de la phrase (la racine), ne possède donc pas de père mais a cinq noyaux fils directs. Dans nos exemples nous avons décider de prendre que le père et les enfants (éventuels) de notre mot à désambiguïser.

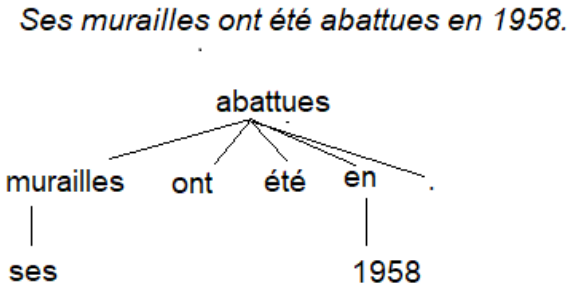

Figure 2: Arbre de dépendance syntaxique (feature=mot-fléchit)

2.2.1 VECTORISATION

Une fois nos exemples créés nous procédons à leur vectorisation. Nous nous sommes concentrés sur deux types de vectorisation :

1. La vectorisation par somme des *embeddings* pré-entraînés du contexte du mot à désambiguïser.
2. La vectorisation par fréquence de terme et inverse de fréquence de document (*tf-idf*).

2.2.2.1. Embeddings de mots.

CBOW (Continus bag of word) est un modèle de Word2Vec, qui a pour but de trouver un mot ciblé à partir de ses mots de contexte, autrement dit, les mots autour du mot ciblé (avec une demi-fenêtre de 2). Si l'on reprend le même exemple ci-dessus : « *Ses murailles ont été abattues en 1958.* », avec le mot ciblé qui est « abattre », les mots de contexte avec une demi-fenêtre de 2 sont [avoir, être, en, 1958] en lemme.

Pour le modèle de CBOW, l'entrée est cette liste de mots de contexte, et la sortie est la liste des sens possible pour le mot ciblé. Par exemple, selon l'inventaire de sens, le mot « abattre » a 4 sens possible différents, donc il y a 4 classes pour la sortie. Le but est donc d'entraîner le système (training) à déterminer quel est le sens correct du mot ciblé.

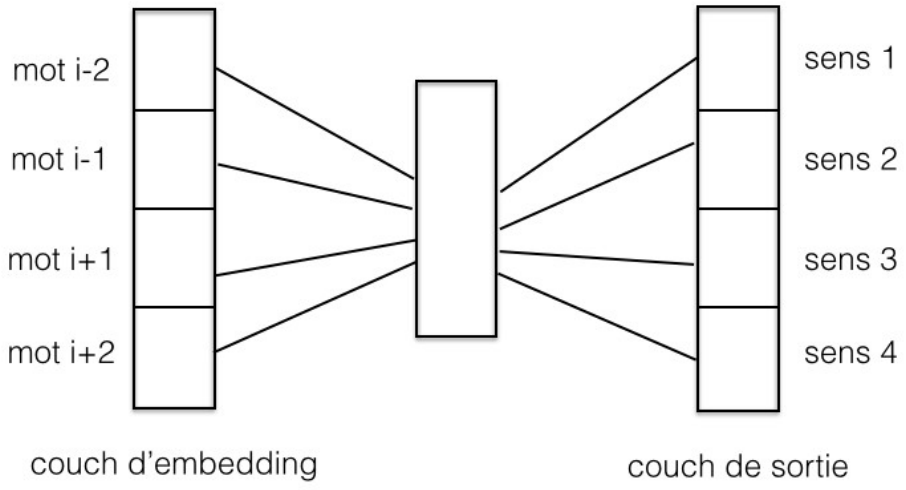

Figure 3 : modèle de CBOW

Une fois le training effectué, il est possible d'extraire les vecteurs de chaque mot d'entrée (mots de contexte). Ensuite on somme les vecteurs des mots de contexte pour obtenir le vecteur de l'exemple.

Pour notre projet, nous avons codé ce type d'architecture en utilisant les données fournies comme exemples d'apprentissage mais nous les avons aussi comparé avec des vecteurs déjà pré-entraînés sur des corpus beaucoup plus longs et mis à disposition librement. Nous avons notamment, récupéré et réadapté les vecteurs proposées par [2][1].

2.2.2.2. TF-IDF

Le TF-IDF (Term Frequency - Inverse Document Frequency) est une méthode qui permet de pondérer les mots d'un exemple dans le corpus (il y a 161 exemples, ou phrases, au total dans le corpus du verbe « abattre »). Le but de TF-IDF est donc d'évaluer quel mot est important dans un exemple, et quels mots sont peu utiles pour déterminer le sens d'un mot ciblé (tels que les articles, les auxiliaires,...).

Cela se détermine par le nombre d'occurrence des mots dans un exemple, et leur fréquence d'apparition dans le corpus. Par exemple, les mot tels que « le, un, être » apparaissent souvent, pas seulement dans un exemple mais tout les exemples du corpus. Ces mots n'ont donc pas la capacité de distinguer les traits d'un exemple. Par contre, pour les mots « manger, boire, ranger », ces mots n'apparaissent pas dans tout le corpus mais seulement quelques exemples, du coup ces mots nous permettent de distinguer les traits d'un exemple.

Le TF-IDF est un score que l'on assigne à chacun des mots d'un corpus (qui se calcule donc pour la totalité du corpus, et non pas exemple par exemple). Il consiste au produit de TF (Term Frequency) et de IDF (Inverse document frequency) :

1disponibles à l'addresse : *https://fasttext.cc/docs/en/crawl-vectors.html*

- *Le TF calcule combien de fois un mot apparaît dans un exemple, divisé par le nombre total de mots dans l'exemple. Par exemple, avec la phrase en lemme « son muraille avoir être abattre en 1958 », vu qu'il n'y a pas de mot répété, et qu'il y a 7 mots au total dans la phrase. Donc pour chaque mot de cette phrase, le taux de TF est tous 1/7. Donc le TF regarde la fréquence d'un mot dans un exemple (phrase).*
- *Le IDF calcule combien dans combien d'exemple le mot apparaît (cela correspond à l'occurrence). On divise le nombre total des exemples par le nombre d'exemples dans lesquels le mot apparaît. IDF correspond au log de ce chiffre. Donc plus le nombre d'exemples dans lequel le mot apparait est petit, et plus le score de IDF sera grand, et vice-versa. Si le nombre total des exemples est identique au nombre d'exemples dans lequel le mot apparait, le score de IDF est 0, et cela signifie que ce mot n'a pas d'importance.*

A la fin, le produit des scores TF et IDF nous permet d'obtenir la valeur de TF-IDF. Un exemple complet ci-dessous :

<table>
<tr><td colspan="3">Exemple : « ‘Ses murailles ont été abattues en 1958.’
‘Les creeks décident de résister, mais ils sont finalement abattus.’»</td></tr>
<tr><td colspan="3">Lemme d'exemple : « ‘son muraille avoir être abattre en 1958.’,
‘le creek décider de résister, mais il être finalement abattre.’ »
POS d'exemple : « ‘D N V V V P N PONCT.’,
‘D N V P V PONCT C CL V ADV V PONCT’ »</td></tr>
<tr><td colspan="3">TF-IDF en lemme en visant le mot « être »</td></tr>
<tr><td>TF</td><td rowspan="3">[‘avoir être abattre1[2] en 1958’,
‘être finalement abattre2 . *f1*[3]’]</td><td>0.2</td></tr>
<tr><td>IDF</td><td>log(2/2) = 0</td></tr>
<tr><td>TF-IDF</td><td>0.2 * 0</td></tr>
<tr><td colspan="3">TF-IDF en POS en visant le nom « N »</td></tr>
<tr><td>TF</td><td rowspan="2">[‘V V abattre1 P N’,</td><td>0.2</td></tr>
<tr><td>IDF</td><td>log(2/1) = log2</td></tr>
</table>

2. Abattre1 signifie que ce verbe abattre ici possède le premier sens selon notre l'inventaire. C'est pour distinguer abattre avec différents sens.

3. Au cas ou si la demie fenêtre dépasse la phrase, on ajoute les symboles comme ‘*d1*’, *d2* avant d'une phrase et ‘*f1*’, ‘*f2*’ après une phrase pour une demie fenêtre de 2, si la valeur de demie-fenêtre est plus élevée, il y aura plus de symboles ajoutés pour éviter le dépassement.

	‘V ADV abattre2 PONCT *f1*’]	
TF-IDF		0.2 * log2

Tableau 2 : exemple de calcul TF-IDF

2.2.3 CLUSTERING

Une fois nos exemples vectoriser, la partie de clustering peut avoir lieu. Nous utilisons deux algorithmes de clustering : *Kmeans* et *Hierarchique* que nous allons lancer sur les mêmes données vectoriser à fin de comparer leur performances selon différents hyper-paramettres.

2.2.3.1 Kmeans clustering

C’est un apprentissage non supervisée. Selon les données, on peut trouver ses « centroids » corresponds au nombre de clustering que nous avons défini. Avec N nombres de « centroids », on peut classifier les données à N nombres de groupes par le processus ci-dessous :

1. On place les données et on définit N nombres de clusters

2. La machine choisit N nombres des centroids aux hasard sur le plan

3. Les données les plus proches d’un centroid appartiennent à ce centroid

4. On met les centroids au centre des donnes qui appartiennent à ce centroid

5. On répète l’étage 3 et 4 pour avoir une convergence et on obtient nos clusters à la fin

Parfois, on ne connaît pas forcement le meilleur nombre de groupes pour les données. Pour résoudre ce problème, on peut appliquer la « méthode de coude » (Elbow method) qui nous permet de visualiser quel nombre de cluster est le meilleur.

2.2.3.2 Clustering hiérarchique

Cette manière de faire clustering est également un apprentissage non supervisé comme KMeans. Par contre, l’algorithme n’est pas pareil. Pour faire le clustering hiérarchique, on prend l’idée de dendrogramme comme ci-dessous :

1. *On met d’abord 2 données les plus proches dans un cluster*
2. *On met les données au cluster le plus proche pour former un nouveau cluster.*
3. *On répète l’étape 2 jusqu'il nous reste qu’un cluster*

À différence de Kmeans, cet algorithme a besoin d’un ensemble d’hyper-parametres qui doivent être choisit arbitrairement et qui peuvent avoir un impact considérable sur le résultat. Le deux plus importants et ceux que nous avons testés sont :

- La mesure de distance pour entre les vecteurs peut être réalisée par la *distance euclidienne, la distance de manhattan* ou le *cosinus.*
- Le critère d'unification des clusters (comment savoir quels clusters sont les rapprochés) sont donnés par *ward (*méthode reposant sur le calcule de la variance au sein d'un cluster*), single (*on compare la distance entre les deux points le plus rapprochés des deux clusters*) , complete(*on compare la distance entre les deux points le plus éloignés des deux clusters*)* et *distance moyenne(*on compare la distance moyenne entre les deux clusters*).*

2.2.3.3 Évaluation

Pour évaluer la précision des prédictions de nos systèmes nous avons utilisés la méthode de l'intersection : on fait intersection de tous les clusters avec tous les éléments de toutes les classes. On prend le nombre des éléments l'intersection le plus grand. Puis on calcule la précision sur ces listes de prédictions. On a aussi calculé la matrice de confusion associée.

3. RÉSULTATS, ANALYSE ET INTERPRÉTATION

3.1 RÉSULTATS

À continuation nous présentons les meilleurs résultats obtenus par verbe ainsi que le type d'exemple utilisé, la représentation vectorielle et l'algorithme de clustering (avec ses paramètres) associés.

Table 1: Meilleurs scores obtenus pour chaque verbe

Verbe	Accuracy	Création exemple	Vectorisation	Clustering
Abattre	0.68	k^*=4 linéaire lemme window=3	Embeds	**KMeans**
Aborder	0.84	k^*=4 linéaire lemme window=3	TF-IDF	**Hierarchical** Mesure de distance : euclidean Linkage : single
Affecter	0.98	k^*=2 linéaire	Embeds	**Hierarchical** Mesure de

		POS(dep) window=3 :		distance : cosine Linkage : single
Comprendre	1.0	k*=2 linéaire POS window=2	TF-IDF	**KMeans**
Compter	0.9	k*=2 linéaire lemme window=2	TF-IDF	**KMeans**

(*) *Le paramètre k indique le nombre de clusters à chercher*

3.2 ANALYSE

Une première chose à noter en voyant ces résultats est l'inégalité entre les verbes. Cette inégalité peut se comprendre principalement selon deux facteurs. Premièrement, les verbes les plus difficiles à classer ("abattre" et "aborder") sont les deux verbes qui possèdent le plus de sens différents (entre quatre et cinq définitions possibles contre deux pour "comprendre"). En second lieu, le nombre de clusters que nous avons chercher à obtenir avec les algorithmes de clustering. Il est évident que classer un élément dans deux possible groupes est plus facile que le faire avec quatre ou cinq.

Notons ensuite que le type d'exemple qui a le mieux marché sont ceux de type linéaires en utilisant principalement les lemmes des éléments dans la phrase et une fenêtre contextuelle allant de 2 à 3. Concernant la vectorisation, aussi bien la vectorisation par *embedding*s que par *tf-idf* semblent marcher. Il en va de même pour les algorithmes de clustering. Selon le verbe, soit *Kmeans* soit *Hierarchique* marchent bien. En ce qui concerne les paramètres de *Hierarchique* on constate que le linkage préféré est point le plus proche (*single*) mais ne se dégage pas une mesure de distance particulière.

3.3 INTERPRÉTATION

Dans les lignes qui suivent nous allons essayer de proposer une explication de l'inégalité des résultats.

Partons du verbe *comprendre.* C'est le seul qui a obtenu un score parfait, c'est aussi le seul verbe qui ne présente que deux sens lexicaux ce qui facilite considérablement la partie de classification. La méthode la plus naïve de classification aurait théoriquement 50% de chances de bien classer un exemple. Une possible explication viendrait du fait que les deux sens de ce lexème étant bien différents, les contextes d'apparition du verbe serait donc eux aussi bien distincts ce qui entraînerait une représentation vectorielle différente selon le cas. En effet, les

deux sens possibles de *comprendre* (“faire partie de” et “acquisition d’un concept/idée”) sont radicalement différents, et dans cette perspective, classer les deux représentations devient une tâche plutôt aisée.

Contrairement, *abattre*, qui a obtenu le pire résultat (68%), est un verbe qui présente cinq possibles définitions qui en plus sont semblables les unes des autres au point que même parmi les locuteurs français on ne se met pas souvent d’accord sur leur nombre. On s’attend alors à que le verbe apparaisse dans des contextes très similaires et donc leur représentation vectorielle serait très rapprochées ce qui rendrait le clustering bien plus difficile.

4. DISCUSSION

4.1 DIFFICULTÉS RENCONTRÉES : ASSOCIATION DES CLUSTERS

Quand on fait clustering par soit le KMeans soit hiérarchique, il n’est pas facile de décider quels clusters correspondants aux clusters gold class. Ce que nous faisons pour le moment, c’est d’utiliser la fonction « intersection » pour chercher le nombre le plus important entre les clusters et puis on trouve les clusters associés plus optimales où on obtient le meilleur taux d’accuracy.

4.2 LES AUTRES IDÉES DE DÉSAMBIGUÏSATION

4.2.1 La possibilité de la structure des phrases

On peut résumer la structure des phrases pour une même étiquette. Ensuite, on peut chercher à prédire la structure la plus probable d’une phrase pour une étiquette donnée. Donc quand on rencontre des nouvelles phrases et que nous analysons leur structure on peut éventuellement avoir des bons estimateurs pour trouver l’étiquette correspondante la plus probable.

4.2.2 L’algorithme de Lesk & clustering

Une autre possibilité intéressante serait de combiner l’algorithme de Lesk avec des méthodes de clustering. Comme il a été montré dans [1], ce genre d’approche permet d’accroître sensiblement la performance du système.

5. RÉFÉRENCES

Choudhury, Jyotirmayee, Deepesh Kumar Kimtani, et Alok Chakrabarty. « Text Clustering Using a Word Net-Based Knowledge-Base and the Lesk Algorithm ». *International Journal of Computer Applications* 48, nº 21 (30 juin 2012): 20-24. https://doi.org/10.5120/7480-0545.

E. Grave*, P. Bojanowski*, P. Gupta, A. Joulin, T. Mikolov, *Learning Word Vectors for 157 Languages*

Mostafa, M S, M H Haggag, et W H Gomaa. « DOCUMENT CLUSTERING USING WORD SENSE DISAMBIGUATION », s. d., 6.

Navigli, Roberto. « Graph Connectivity Measures for Unsupervised Word Sense Disambiguation », s. d., 6.

Ranjan Pal, Alok, et Diganta Saha. « Word Sense Disambiguation: A Survey ». *International Journal of Control Theory and Computer Modeling* 5, nº 3 (31 juillet 2015): 1-16. https://doi.org/10.5121/ijctcm.2015.5301.

Yarowsky, David. « Unsupervised Word Sense Disambiguation Rivaling Supervised Methods ». In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics* -, 189-96. Cambridge, Massachusetts: Association for Computational Linguistics, 1995. https://doi.org/10.3115/981658.981684.

ANNEXE : IMPLÉMENTATION EN PYTHON

MATÉRIEL

Pour atteindre l'objectif de ce projet, nous avons la disposition des documents composés par l'***inventaire de sens*** et ***les données des 5 verbes***.

- Un fichier d'***inventaire de sens*** pour montrer comment on classifie les sens d'un verbe.
- 5 dossiers sont séparément pour les 5 verbes : « abattre », « aborder », « affecter », « comprendre » et « compter »
- Il y a 3 fichiers de formats différents dans chaque dossier. Le nom du fichier montre le verbe qu'il contient et le nombre de phrase.
- Le premier fichier y compris le corpus des phrases qui contiennent le verbe visé, des mots sans forme fléchie, POS de chaque phrase, analyse de dependancy de chaque phrase.

Pour les détails d'implémentation merci de se référer au fichier *readme.txt* dans le dossier contenant le code du projet.