Constitution et évaluation d'un jeu de données linguistiques en français pour l'analyse des fonctions lexicales encodées dans les modèles neuronaux de type FlauBERT

Vincent Bellue



Constitution et évaluation d'un jeu de données linguistiques en français pour l'analyse des fonctions lexicales encodées dans les modèles neuronaux de type FlauBERT

Vincent
BELLUE

Sous la direction de Maximin COAVOUX et Olivier KRAIF

Laboratoire : Laboratoire d'Informatique de Grenoble (LIG)

UFR LLASIC

Département Sciences du langage

Mémoire de master 2 mention Sciences du langage - 20 crédits

Parcours : Industries de la langue - orientation professionnelle

Année universitaire 2019-2020

Constitution et évaluation d'un jeu de données linguistiques en français pour l'analyse des fonctions lexicales encodées dans les modèles neuronaux de type FlauBERT

Vincent
BELLUE

Sous la direction de Maximin COAVOUX et Olivier KRAIF

Laboratoire : Laboratoire d'informatique de Grenoble

UFR LLASIC
Département Sciences du langage

Mémoire de master 2 mention Sciences du langage - 20 crédits

Parcours : Industries de la langue - orientation professionnelle

Année universitaire 2019-2020

Remerciements

Je tiens tout d'abord à remercier Maximin Coavoux et Olivier Kraif sans qui je n'aurais pas pu effectuer mon stage. Je les remercie aussi pour leur disponibilité et tous leurs précieux conseils et exemples explicatifs.

Je remercie également Agnès Tutin d'avoir accepté d'être ma tutrice de stage à l'UGA ainsi que l'aide qu'elle m'a apportée concernant les fonctions lexicales.

Enfin, je remercie Rachel qui m'a toujours aidé tout au long de mes deux années de master mais aussi durant mon stage. Je remercie aussi Florent, Justine, Lucie, Marie, Myriam et Oussama avec qui j'ai passé deux merveilleuses années de master.

DÉCLARATION ANTI-PLAGIAT

1. Ce travail est le fruit d'un travail personnel et constitue un document original.
2. Je sais que prétendre être l'auteur d'un travail écrit par une autre personne est une pratique sévèrement sanctionnée par la loi.
3. Personne d'autre que moi n'a le droit de faire valoir ce travail, en totalité ou en partie, comme le sien.
4. Les propos repris mot à mot à d'autres auteurs figurent entre guillemets (citations).
5. Les écrits sur lesquels je m'appuie dans ce mémoire sont systématiquement référencés selon un système de renvoi bibliographique clair et précis.

PRÉNOM : VINCENT

NOM : BELLUE

DATE : 14/09/2020

Table des matières

Introduction

Le master « Industries de la langue » proposé par l'Université Grenoble Alpes (UGA) est un master mêlant les sciences du langage à l'informatique. Il a pour vocation de former de futurs experts en Traitement Automatique de la Langue (TAL). Dans le cadre de ce master, il nous a été demandé de choisir entre un parcours recherche et professionnel. Le parcours professionnel, que j'ai choisi, permet de nous préparer au monde du travail avec des enseignements spécifiques tels que la gestion de projet en 1ère année et l'élaboration d'un projet professionnel pour des commanditaires en 2e année. De plus, un stage de fin d'études d'une durée de 4 à 6 mois est obligatoire afin de valider son diplôme.

J'ai choisi un stage qui a été proposé par Maximin Coavoux, membre du Laboratoire d'Informatique de Grenoble (LIG) et Olivier Kraif, membre du Laboratoire de Linguistique et Didactique des Langues Étrangères et Maternelles (LIDILEM). Celui-ci s'est déroulé du 02 mars 2020 au 31 août 2020 au LIG où j'ai été encadré par Maximin Coavoux et Olivier Kraif durant mes 6 mois de stage.

Mon stage a été organisé en deux parties. Tout d'abord j'ai constitué un corpus linguistique manuellement, puis par la suite j'ai développé plusieurs scripts d'extraction automatique d'informations sur mon corpus.

L'objectif principal de mon stage était de déterminer si le modèle de langage « FlauBERT » arrivait à distinguer les différences entre collocations et cooccurrences. Mon hypothèse était que s'il en était capable, alors il y aurait des éléments indicateurs dans l'encodage de ses vecteurs de mots.

Cependant, on ne peut affirmer avec certitude que FlauBERT en est capable sans avoir fait de tests au préalable. Toutefois, il existe un champ de recherche s'intéressant aux informations véhiculées par les vecteurs. C'est pour cela que j'ai effectué plusieurs expérimentations comme la similarité cosinus ou l'entraînement d'un réseau de neurones, permettant tous deux d'obtenir des informations sur les vecteurs de FlauBERT.

Mon plan est construit en plusieurs parties. Tout d'abord, je commencerai par une présentation de mon lieu mon stage. Ensuite, je présenterai les différents domaines qui ont été en jeu durant mon stage dans la partie état de l'art. Puis, j'expliquerai les différentes expérimentations que j'ai effectuées ainsi que leurs résultats. Finalement, je ferai une conclusion sur ce que ce stage a apporté sur le sujet mais aussi ce qu'il m'a apporté personnellement.

1 - Structure d'accueil

Ce stage est issu d'un partenariat entre le Laboratoire d'Informatique de Grenoble (LIG) et le Laboratoire de linguistique et didactique des langues trangères et maternelles (LIDILEM) à Grenoble. Il s'inscrit dans le cadre de la chaire « IA et langage ».

Nous commencerons d'abord par une présentation du LIDILEM et du LIG puis d'une présentation de l'équipe GETALP.

1.1 - LIDILEM

Le Laboratoire de Linguistique et Didactique des Langues Étrangères et Maternelles est un laboratoire grenoblois créé en 1987. Ses locaux se trouvent actuellement sur le campus universitaire de Saint-Martin-d'Hères, au sein de l'UGA.

Le laboratoire possède des membres permanents ainsi que des doctorants et stagiaires. Leurs recherches sont centrées autour de plusieurs axes en rapport avec les sciences du langage, dont le TAL ou encore la constitution et exploitation de corpus. L'étude des langues et langages est donc très importante pour les membres du laboratoire. Ils sont de plus en collaboration avec plusieurs organismes dont le LIG, qui possède aussi une équipe qui s'intéresse au TAL. Plusieurs projets ont vu le jour au LIDILEM comme le projet EMOLEX ou encore PhraseoRom.

1.1.1 - EMOLEX

EMOLEX est l'un des leurs anciens projets, datant de 2009. L'une des vocations de ce projet était d'analyser le comportement syntaxique et combinatoire des lexies[1] dans plusieurs langues. EmoConc, un outil informatisé a été développé dans le cadre de ce projet. Il permet notamment de faire des recherches de concordance entre plusieurs mots au sein d'une même phrase. Il a depuis été adapté et tenu à jour par Olivier Kraif sous le nom de « Lexicoscope ». Cet outil a la capacité d'analyser plusieurs autres corpus qui ont été ajoutés au fil des ans, comme ceux du projet PhraseoRom.

1.1.2 - PhraseoRom

Le projet PhraseoRom est l'un des derniers projets du laboratoire, terminé en 2020. Il s'intéressait à la phraséologie dans les romans. Les corpus de PhraseoRom ont été numérisés et sont maintenant analysables par le Lexicoscope.

1 Il s'agit de toute unité du lexique, comme un mot ou une expression.

1.2 - LIG

Le Laboratoire d'Informatique de Grenoble (LIG) est un laboratoire grenoblois fondé en 2007. Il possède 25 équipes réparties selon 5 axes de recherche, tous en rapport avec les sciences informatiques. De plus, les équipes sont réparties sur 3 sites géographiques à travers Grenoble et son agglomération.

Nous nous focaliserons sur le site du campus universitaire de Saint-Martin-d'Hères et plus particulièrement le bâtiment IMAG. Il s'agit d'un bâtiment à la pointe de la technologie, accueillant 800 enseignants-chercheurs, chercheurs, doctorants et stagiaires. Plusieurs laboratoires y possèdent des locaux, dont le LIG.

1.3 - GETALP

L'équipe GETALP (Groupe d'Étude en Traduction Automatique/Traitement Automatisé des Langues et de la Parole) est l'une des équipes du LIG. Ses locaux sont situés au sein du bâtiment IMAG, au 3e étage. Cette équipe a vu le jour en même temps que la création du LIG en 2007. Il s'agit en réalité du successeur du GETA, lui-même successeur du CETA (1959), qui était l'un des premiers laboratoires à faire du TAL en France.

Le GETALP possède plusieurs membres, tous unis autour des thématiques du TAL. L'un des points forts du GETALP est sa pluridisciplinarité. En effet, l'équipe est très polyvalente, possédant des membres spécialisés dans les sciences informatiques mais aussi dans les sciences du langage. Plusieurs membres sont d'ailleurs enseignants-chercheurs et proposent des enseignements à l'UGA.

La traduction automatique, le traitement de la parole et de l'écrit sont au centre de leurs axes de recherches et plusieurs projets développés par le GETALP ont vu le jour depuis 2007. L'un de leurs derniers projets est le modèle de langage « FlauBERT », mais aussi « FLUE ».

1.3.1 - FlauBERT

FlauBERT (Le et al., 2019) est un modèle de langage développé lors d'une collaboration incluant le GETALP en 2019. Il s'agit d'une version française de BERT (Devlin et al., 2019), initialement développée par *Google Research* en 2018. FlauBERT a été entraîné sur de gros corpus du français comme Wikipédia. Ces nouveaux modèles ont permis une grande avancée dans le domaine du TAL et notamment les plongements de mots (*word*

embeddings), grâce à la possibilité d'obtenir des vecteurs contextuels au lieu de vecteurs statiques (cf. 2.9 - Vecteurs statiques et contextuels, page 17).

1.3.2 - FLUE

Les développeur·es de FlauBERT ont aussi créé « FLUE » (French Language Understanding Evaluation) en même temps. Il s'agit d'une agrégation de jeux de données en français, permettant l'évaluation de FlauBERT sur diverses tâches de TAL telles que la classification de textes, la paraphrase ou encore l'étiquetage morpho-syntaxique. FLUE est une ressource similaire à GLUE (Wang et al., 2018).

2 - État de l'art

Le Traitement Automatique de la Langue (TAL) est un domaine réunissant plusieurs disciplines clés. Parmi elles se trouve la linguistique, qui joue une part importante dans la compréhension des phénomènes linguistiques en jeu et permet une analyse approfondie de ceux-ci. L'informatique joue aussi une grande part dans le TAL, notamment grâce à l'automatisation de tâches telles que la fouille et l'analyse automatique de données grâce à des algorithmes. Le TAL englobe aussi d'autres disciplines comme l'intelligence artificielle qui joue un rôle très important dans l'apprentissage machine.

Dans cette partie, nous traiterons des différentes notions qui ont été vues ou mises en pratique durant ce stage et dont la compréhension a été nécessaire.

2.1 - Le lexique

Le lexique est l'ensemble des mots d'une langue. Pour le français, qui est une langue vivante, le lexique est ouvert, c'est-à-dire qu'au fur et à mesure de l'évolution du français son lexique évolue aussi, notamment avec de nouveaux mots ou combinaisons de mots. Le lexique diffère selon les langues, c'est pour cela que je ne me focaliserai que sur le lexique français.

2.2 - Polysémie

La polysémie est la propriété d'un mot à posséder plusieurs sens. On dit alors qu'il est polysémique. Chaque sens est utilisé dans un contexte spécifique. La plupart des mots de la langue française possèdent cette propriété. Par exemple le mot *brillant* peut être défini de plusieurs manières. Il peut s'agir du sens « quelque chose qui brille » ou encore « quelque chose qui sort du commun, de remarquable ». Lorsque l'on produit un énoncé, le sens utilisé sera choisi par le locuteur selon le contexte d'énonciation ainsi que l'idée qu'il souhaite exprimer à travers cet énoncé. En effet, si l'on souhaite parler d'un élève qui sort de l'ordinaire, on parle souvent d'« un brillant élève ». Dans ce cas, *brillant* fonctionne de manière à indiquer qu'il s'agit d'un élève « qui sort du commun, de l'ordinaire » et ce n'est en aucun cas « un élève qui brille ».

2.3 - Cooccurrence

Lorsque plusieurs mots apparaissent au sein d'un énoncé, on parle de « cooccurrence », c'est-à-dire qu'ils apparaissent simultanément. Si l'on prend l'énoncé « un trou profond », qui est un exemple de cooccurrence possible entre *trou* et *profond*. Si l'on se focalise sur *profond*, celui-ci possède plusieurs significations possibles car il s'agit d'un cas de polysémie. Le sens de *profond* utilisé par « trou profond » est celui d'« un trou d'une hauteur élevée ». Sémantiquement, cet énoncé signifie qu'il s'agit bien d'un trou d'une hauteur élevée. C'est ce qu'on appelle la « compositionnalité sémantique ». Il s'agit de « la résultante de la composition du sens des éléments qui constituent l'énoncé » (Polguère, 2002, p. 45). En effet, lorsqu'il s'agit d'une simple cooccurrence, le sens de l'énoncé est construit à partir du sens de ses constituants.

2.4 - Collocations

Lorsque la cooccurrence entre plusieurs mots est très fréquente voire systématique, on dit alors qu'ils forment une « collocation ». Une collocation est régie par une unité lexicale qui est la base de la collocation, qu'on appelle la « base ». Elle est choisie librement et le sens de la collocation est dicté par le sens de cette base. La collocation possède un second élément appelé le « collocatif ». À l'inverse de la base, le collocatif n'est pas libre, c'est-à-dire qu'il est choisi en fonction de la base. Par exemple « gros chagrin » est une combinaison très fréquente entre *chagrin* et *gros.* Cette collocation est constituée de la base *chagrin* et du collocatif *gros*. On peut facilement remarquer que lorsqu'on parle d'un « gros chagrin », l'élément qui est choisi librement est *chagrin*, tandis que *gros* est choisi en fonction de *chagrin*. En effet, « la base contrôle la collocation, car du point de vue du locuteur, c'est le collocatif qui est choisi en fonction de la base, et non l'inverse » (Polguère, 2002, p. 48).

2.5 - Compositionnalité

On parle de compositionnalité sémantique lorsque la signification d'un énoncé est définie par le sens des mots le constituant. Une cooccurrence obéit entièrement au principe de compositionnalité sémantique. Par exemple, dans l'énoncé « le chat dort », nous pouvons aisément obtenir le sens de l'énoncé en analysant les sens des mots le constituant.

Cependant, les mots faisant partie d'une collocation ont un lien spécial, rompant parfois le principe de compositionnalité sémantique. Cela donne lieu à des expressions semi-

idiomatiques ou idiomatiques. L'idiomaticité est « une transgression du principe de compositionnalité sémantique, c'est un phénomène graduel » (Polguère, 2002, p. 45), c'est-à-dire que le principe de compositionnalité sémantique est plus ou moins respecté selon les cas. Il n'existe pas de nette séparation entre deux catégories distinctes qui seraient « compositionnelle » et « non-compositionnelle ».

Cependant, lors d'une collocation, une restriction combinatoire a lieu entre la base et le collocatif. Cela empêche fortement toute substitution du collocatif, qui est totalement dépendant de la base. Par exemple « une peur bleue » signifie « une peur extrême ». Un rapport analogique est établi entre la couleur *bleue* et la « coloration bleu pâle des lèvres et des extrémités (doigts et orteils) chez les malades cyanosés par le choléra » (*peur bleue — Wiktionnaire*, s. d.). Le principe de compositionnalité sémantique est alors rompu en utilisant *bleue* de manière métonymique. Pour finir, il n'est pas possible de substituer le collocatif *bleue* par l'un de ses synonymes sans changer le sens de la collocation.

2.6 - Fonctions lexicales

Les fonctions lexicales sont un moyen de modéliser les liens lexicaux qu'entretiennent les unités lexicales, sous la forme d'une fonction. Il peut s'agit une fonction lexicale paradigmatique, qui permet de modéliser la dérivation sémantique ou bien d'une fonction lexicale syntagmatique, qui relie une base à ses collocatifs (Systèmes Lexicaux – Alain Polguère, s. d.). Je ne traiterai dans ce mémoire que des fonctions lexicales syntagmatiques.

Par ailleurs, les fonctions lexicales sont classées en plusieurs catégories. Il existe des fonctions lexicales standards, non-standards ou encore complexes (Polguère, 2002, p. 50). Leur fonctionnement est universel, ne se limitant pas au français. De plus, il est envisageable de pouvoir les utiliser afin de décrire les liens lexicaux de futurs expressions ou combinaison de mots.

Les fonctions lexicales fonctionnent selon le principe suivant : $f(L) = \{L_{valeur}\}$ où L correspond au mot-clé et L_{valeur} correspond à la valeur (Polguère, 2002, p. 50).

Durant mon stage, je me suis intéressé à deux fonctions lexicales syntagmatiques standards connues sous le nom de « Bon » et « Magn ». Elles sont considérées comme standards car elles possèdent plusieurs propriétés comme la « couverture » ou la « diversité ». La couverture est le fait de posséder « un grand nombre de lexies L » et la diversité le fait

d'avoir une « assez grande diversité parmi toutes les valeurs retournées pour f(L) dans chaque langue donnée L » (Polguère, 2002, p. 51).

Si toutes les conditions ne sont pas remplies, une fonction lexicale est alors considérée comme non-standard. Je ne rentrerai pas dans les détails des fonctions lexicales non-standards et complexes, car je ne les ai pas utilisées durant mon stage mais celles-ci sont expliquées par Polguère (2002).

2.6.1 - Fonction lexicale Bon

« Bon » est une fonction lexicale standard fonctionnant avec les collocations. Elle décrit sous la forme d'une fonction le lien qu'entretiennent la base et le collocatif d'une collocation dont le sens exprime une notion de positivité, « l'évaluation positive/l'approbation du locuteur » (Polguère, 2002, p. 57). Une collocation pouvant relever de cette fonction lexicale est la collocation « idée brillante ».

Elle est représentée par la fonction lexicale « Bon » de cette manière : Bon(idée) = brillante.

S'agissant d'une collocation, on peut remarquer qu'il existe un certain degré d'idiomaticité concernant cette expression. Elle n'est pas figée car il est possible de remplacer *brillante* par des synonymes tels que *ingénieuse*, *remarquable*, *excellente*, mais d'autres expressions ne possèdent pas autant de liberté concernant les collocatifs employés.

2.6.2 - Fonction lexicale Magn

« Magn » est une fonction lexicale standard qui fonctionne aussi avec des collocations. Toujours sous la forme d'une fonction, elle montre cette fois-ci le lien qu'entretiennent la base et le collocatif d'une collocation dont le sens exprime une idée d'intensification.

Le collocatif vient influencer la base en lui rajoutant de l'intensité. Un exemple fonctionnant pour cette fonction lexicale est la collocation « dégoût profond » que j'ai aussi eu l'occasion d'utiliser durant mon stage.

La collocation « dégoût profond » est représentée par la fonction lexicale Magn de cette manière : Magn(dégoût) = profond.

Tout comme le précédent exemple, « dégoût profond » possède aussi un certain degré d'idiomaticité. Cette expression n'est pas totalement figée non plus car il est possible de substituer *profond* par quelques-uns de ses synonymes exprimant l'intensité, comme *immense*, mais le choix de collocatif reste bien plus restreint que pour « idée brillante ».

2.7 - Apprentissage machine

L'apprentissage machine est utilisé dans plusieurs domaines à l'heure actuelle, dont le TAL. Il s'agit d'un champ d'étude de l'intelligence artificielle permettant de faire apprendre certaines tâches à un ordinateur via un algorithme. En TAL, cette tâche pourrait être la traduction automatique de textes, la reconnaissance d'entités nommées (lieux, dates) ou encore l'analyse de sentiments (*sentiment analysis).*

Tout d'abord, il existe plusieurs types de données. Il peut s'agir de données annotées ou non annotées. Si elles sont annotées, cela permet de connaître à l'avance la bonne réponse à une question par rapport à un critère. Ce critère pourrait être par exemple de différencier une cooccurrence et une collocation. Cela aiderait la machine à aller dans la bonne direction en lui indiquant à quoi correspond une collocation et à quoi correspond une cooccurrence. On dit alors qu'il s'agit d'un « apprentissage supervisé ». Les bonnes réponses sont déjà connues, il est donc plus facile de faire apprendre quelque chose de cette manière. De plus, il est possible de modifier certains paramètres afin de faire en sorte que la machine puisse trouver le mieux possible la bonne réponse.

À l'inverse, il existe des cas où les données ne sont pas annotées, qu'on appelle « apprentissage non supervisé ». Lors d'apprentissages non supervisés, on ne cherche pas nécessairement à évaluer la capacité d'un modèle à donner la bonne réponse mais on tente plutôt d'obtenir des représentations à partir d'un critère. Ce critère pourrait être de classer des éléments similaires par groupes. La « classification ascendante hiérarchique » en est un exemple. Il s'agit d'une méthode de classification non supervisée. Des similarités sont analysées par un algorithme, puis de manière non supervisée, celui-ci va tenter de classer automatiquement les similarités, en faisant des regroupements. La représentation vectorielle des mots de FlauBERT est un autre exemple d'apprentissage non supervisé car le modèle apprend par lui-même à représenter les mots sous forme de vecteurs.

2.8 - Plongements de mots

Les plongements de mots, aussi connus sous le nom de *word embeddings* permettent de représenter les mots sous forme de vecteurs. Chaque mot est représenté dans un espace vectoriel. Cette théorie est fondée sur l'hypothèse distributionnelle (Harris, 1954), qui indique que les mots utilisés dans des contextes linguistiques similaires ont des sens proches. Par exemple *roi* est très proche de *reine* car ils sont tous deux utilisés dans des contextes très similaires. Pour obtenir des plongement de mots, il faut tout d'abord constituer un corpus, puis entraîner un réseau de neurones sur ce corpus.

Toutefois, il existe des modèles de langage pré-entraînés, comme « word2vec » (Mikolov et al., 2013) ou « fastText » (Bojanowski et al., 2017). Plusieurs paramètres ont un impact sur le positionnement des vecteurs. Tout d'abord, la taille et le type de corpus utilisé lors de l'entraînement a un impact majeur. Il peut s'agir d'un corpus spécialisé ou bien général, c'est-à-dire un corpus contenant un vocabulaire spécifique à un domaine. Tout dépend de ce que l'on souhaite obtenir comme informations. De plus, de très gros corpus permettent théoriquement de traiter plusieurs contextes dans lesquels les mots sont utilisés. Enfin, les paramètres peuvent aussi varier et pour une même méthode. Il est donc possible d'obtenir des vecteurs différents sur les mêmes données.

La méthode utilisée pour obtenir des vecteurs de mots est de créer un réseau de neurones. Celui-ci est entraîné sur des données d'entraînement afin d'apprendre les différents contextes dans lesquels les mots sont utilisés. En effet, les mots étant utilisés dans des contextes similaires auront tendance à avoir des vecteurs proches, comme *roi* et *château* par exemple.

La plupart des systèmes de plongements de mots possèdent des vecteurs fixes pour chaque mot. L'ensemble des vecteurs dépend de plusieurs facteurs, parmi eux : les données qui ont été utilisées pour entraîner le système. Par ailleurs, il existe des fichiers de vecteurs disponibles publiquement, comme ceux de fastText (cf. Annexe 1, page 61) et word2vec. Ces fichiers de vecteurs sont le résultat de l'apprentissage d'un réseau de neurones, mais les paramètres et méthodes utilisées par les deux modèles diffèrent. Bien qu'ils possèdent tous deux des vecteurs statiques, les vecteurs qu'ils proposent sont quand même différents. Les modèles de plongements de mots traditionnels de type fastText ou word2vec possèdent cependant des limites que nous allons voir dans la prochaine partie.

2.9 - Vecteurs statiques et contextuels

Il existe deux types de vecteurs de mots en TAL. D'un côté les vecteurs statiques tels que ceux utilisés par fastText ou word2vec et d'un autre côté les vecteurs contextuels comme ceux utilisés par FlauBERT.

Les vecteurs statiques sont des types de vecteurs utilisés par les anciens modèles de langage comme word2vec. Chaque mot est représenté par un vecteur qui reste le même indépendamment du contexte dans lequel il est utilisé. Par exemple, que le mot *pièce* soit utilisé dans le contexte « pièce d'une maison » ou « une pièce de monnaie » n'aura pas d'incidence. Il sera représenté par le même vecteur dans les deux cas. Étant donné qu'il s'agit d'un vecteur statique, il n'est pas modifié et reste fixe. La limite de ces modèles est qu'ils traitent très mal la polysémie des mots car le vecteur représentant chaque sens d'un mot est identique.

Par ailleurs, les vecteurs contextuels font partie d'une nouvelle ère de la représentation vectorielle des mots. L'un de leurs avantages est que le contexte de la phrase joue un rôle majeur pour la représentation vectorielle. Les mots composant une phrase ainsi que leurs sens auront donc un réel impact sur le résultat des vecteurs. Pour reprendre notre précédent exemple, *pièce* dans le contexte « pièce de monnaie » ou « pièce d'une maison » possédera cette fois-ci un vecteur différent, car le contexte ainsi que la composition de la phrase sera différent.

Nous allons maintenant voir deux modèles de langage disponibles à l'heure actuelle. Bien qu'il en existe d'autres, je me suis uniquement focalisé sur ces deux là, car il s'agit des deux modèles que j'ai utilisés durant mon stage.

2.9.1 - fastText

FastText est une librairie open source développée par le laboratoire Facebook's AI Research (FAIR). Elle possède notamment des modèles entraînés sur de gros corpus comme Wikipédia. Ces vecteurs de mots ont été pré-entraînés pour 157 langues différentes, dont le français. Ils peuvent être utilisés pour les plongements de mots ou encore la classification automatique de texte. Dans mon cas, j'ai utilisé un fichier de vecteurs pré-entraînés pour le français afin d'obtenir les vecteurs de mots. Le fichier est disponible au téléchargement sur le site de fastText. L'avantage de fastText est qu'il ne requiert aucune installation au préalable. Il suffit de télécharger le fichier de vecteurs. Puis, en quelques lignes de script il est possible de récupérer le vecteur d'un mot donné.

2.9.2 - FlauBERT

Avant de parler de FlauBERT, il est important d'introduire BERT. Il s'agit d'un modèle de langage pré-entraîné sur de gros corpus tels que Wikipédia. Il a la capacité de pouvoir représenter les mots d'une phrase sous forme de vecteurs contextuels, c'est-à-dire que selon le contexte et la manière dont est formée la phrase, les vecteurs de mots seront différents d'une phrase à l'autre.

Les modèles produits par BERT permettent des représentations des mots et des phrases qui permettent d'obtenir de très bons résultats pour les langues sur lesquelles ils sont entraînés, dont le français (Le et al., 2020). Il en est de même pour FlauBERT qui fonctionne sur le même principe.

Contrairement à fastText qui possède des vecteurs fixes, BERT et FlauBERT permettent de traiter des cas de polysémie des mots ainsi que des contextes dans lesquels les mots sont employés.

FlauBERT est une version française de BERT qui a été développée lors d'une collaboration incluant le GETALP du LIG. Elle a été entraînée sur des données publiques telles que Wikipédia (en français) en utilisant le supercalculateur du CNRS « Jean Zay ».

Afin de pouvoir utiliser FlauBERT, il faut installer la bibliothèque *transformers* (Wolf et al., 2020). Elle possède des modèles entraînés sur plus de 100 langues, dont le modèle FlauBERT entraîné sur le français.

Il s'agit un réseau de neurones profond qui procède de manière hiérarchique. Son principe de fonctionnement sera décrit dans les grandes lignes, en omettant certains détails du modèle.

Chaque token[2] est d'abord associé à un vecteur qui dépend de sa forme, formant la première couche du réseau de neurones. Puis, pour chaque couche suivante, le vecteur pour chaque token est calculé à partir de l'ensemble des vecteurs de la couche précédente, pondéré par un mécanisme d'attention.

2 Par exemple, les mots *le* et *chat* sont des tokens.

Voici une illustration permettant de voir le principe de fonctionnement de manière simplifiée :

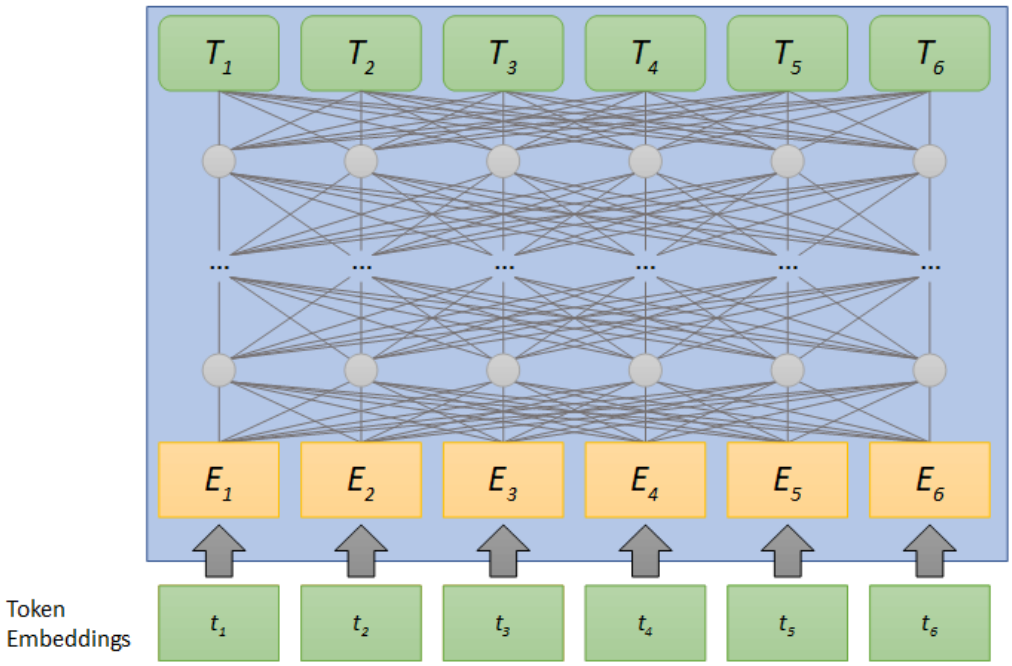


Illustration 1 : Adapté de https://cs.uwaterloo.ca/~jimmylin/BERT-diagrams-public.pptx (licence Creative Commons 4.0).

Par exemple, si l'on prend la phrase « Le chat dort sur le canapé. », afin de calculer le vecteur du mot *chat* de la 2e couche, FlauBERT calcule une attention sur les vecteurs de la 1ère couche. De manière schématique, une attention correspond à des scores normalisés sur un ensemble de tokens, par exemple :

Le : 5 % | chat : 30 % | dort : 60 % | sur : 0 % | le : 0 % | canapé: 5 %.

Ensuite, FlauBERT applique une transformation à une agrégation des vecteurs de la couche précédente, pondérée par l'attention. Finalement, ce mécanisme est utilisé plusieurs fois (attention multi-têtes, 12 fois dans le cas de FlauBERT) pour chaque token de chacune des couches (Vaswani et al., 2017).

3 - Présentation du stage

Mon stage s'est déroulé du 02 mars 2020 au 31 août 2020 sur une durée de 6 mois. Nous pouvons le scinder en deux parties distinctes qui sont aussi dans l'ordre chronologique.

Tout d'abord, une première partie concernant l'organisation et la constitution du corpus. Les tâches étaient essentiellement linguistiques, comme la constitution manuelle d'un corpus, la reconnaissance de collocations ou encore la reconnaissance de la compositionnalité d'un couple de mots.

Ensuite, la deuxième partie du stage a été essentiellement informatique. Il a été nécessaire de faire plusieurs algorithmes d'automatisation des tâches de fouille et d'extraction de données. En effet, plusieurs calculs informatiques ont été nécessaires afin d'obtenir les vecteurs correspondants aux collocations.

3.1 - Constitution du corpus (Partie linguistique)

3.1.1 - Choix des fonctions lexicales

Avant toute chose, il a été nécessaire de choisir quelles fonctions lexicales je devais traiter. Je me suis focalisé sur les fonctions lexicales « Bon » et « Magn ».

Il existe plusieurs fonctions lexicales, dont certaines sont dites « standards ». Les fonctions lexicales standards sont des fonctions lexicales qui acceptent beaucoup d'exemples avec des sens plus ou moins proches. En effet, il doit exister une multitude d'exemples qui soient utilisables pour une seule fonction lexicale afin que celle-ci soit considérée comme standard. C'est le cas des fonctions lexicales Bon et Magn.

3.1.1.1 - Bon

Tout d'abord, la fonction lexicale Bon exprime un sentiment de positivité. Le collocatif vient ajouter une notion de positivité à la base de la collocation (cf. 2.6.1 - Fonction lexicale Bon, page 14).

3.1.1.2 - Magn

Quant à la fonction lexicale Magn, il s'agit d'une notion d'intensification. Le collocatif agit sur la base en lui ajoutant de l'intensité (cf. 2.6.2 - Fonction lexicale Magn, page 14).

3.1.2 - Recherche des collocations

Une fois mes deux fonctions lexicales choisies, il fallait chercher des exemples qui puissent être utilisés avec Bon et Magn. De plus, mon corpus devait être assez conséquent sans toutefois être excessivement long à constituer.

Pour ce faire j'avais à ma disposition un fichier fourni par mes encadrants de stage (cf. FL_extract.tsv, page 62). Dans ce fichier sont référencés plusieurs collocations en lien avec les fonctions lexicales Bon et Magn. J'ai donc choisi, en accord avec mes encadrants de stage, 10 collocations pour la fonction lexicale Bon ainsi que 10 autres pour la fonction lexicale Magn.

Voici la liste des 10 collocations pour chaque fonction lexicale :

Bon : colère saine, fin observateur, beau couple, grand journaliste, idée brillante, élève brillant, ouïe fine, grand chanteur, beau temps et bonne douche.

Magn : soleil de plomb, silence de plomb, grand vent, grand ami, dégoût profond, sommeil profond, grand chagrin, mémoire d'éléphant, bilan lourd et lettre longue.

3.1.3 - Choix des collocations et cooccurrences

Avant de m'intéresser à cette partie, il me semble important d'introduire certains termes que je vais employer. Tout d'abord, il existe une différence entre « cooccurrence » et « collocation » (cf. page 12).

J'utiliserai le terme *collocatif* lorsqu'il s'agira du collocatif d'une collocation et le terme *cooccurrent* lorsqu'il s'agira du mot remplaçant le collocatif dans le cas d'une cooccurrence. Dans les deux cas de figure, je nommerai *base*, la base d'une collocation et l'élément de base d'une cooccurrence. Par exemple, « grand arbre » qui est un exemple de cooccurrence, possède la base *arbre* et le cooccurrent *grand* et « grand ami » possède la base *ami* et le cooccurrent *grand*.

Après avoir choisi mes 10 collocations pour chaque fonction lexicale, je devais trouver 10 contextes où il s'agissait d'un cas de collocation, ainsi que 10 contextes où il s'agissait d'une combinaison libre de mots (cooccurrence). Voici un exemple pour illustrer mes propos :

soleil	plomb
soleil	plomb
soleil	plomb
soleil	plomb
soleil	plomb
soleil	plomb
soleil	plomb
soleil	plomb
soleil	plomb
soleil	plomb

Illustration 2 : collocations.

balles	plomb
lamelles	plomb
balle	plomb
noyaux	plomb
cartouches	plomb
billes	plomb
boulet	plomb
bille	plomb
plaque	plomb
coupole	plomb

Illustration 3: combinaisons libres.

Pour « soleil de plomb », j'ai cherché 10 contextes où il s'agissait d'une collocation réunissant la base *soleil* et le collocatif *plomb*. Puis, pour le même collocatif, j'ai cherché 10 contextes où il s'agissait d'une combinaison libre de mots, c'est-à-dire une base et un cooccurrent. En effet, j'ai sélectionné 10 occurrences où le sens du mot *plomb* désignait « un objet en plomb ». Finalement, seule la base devait être différente entre un cas de collocation et un cas de cooccurrence.

De plus, la base choisie lors de la combinaison libre devait être différente pour chaque exemple de cooccurrence. Cependant, il était quand même possible d'utiliser deux fois le même mot, une fois au singulier et une autre fois au pluriel. Étant donné que FlauBERT possède des vecteurs contextuels, cela n'a pas d'incidence puisque le vecteur est défini par rapport à toute la phrase.

Une autre contrainte qu'il a été nécessaire de bien prendre en compte était que le mot que je choisissais à la place de *soleil* dans le cas d'une combinaison libre devait être issu de la même partie du discours que *soleil* (cf. Illustration 3 : combinaisons libres).

3.1.4 - Choix du corpus d'extraction (Lexicoscope)

Une fois les deux fonctions lexicales choisies (Magn et Bon), il fallait ensuite choisir un corpus où je devais récupérer mes exemples. Le corpus devait être diversifié, avec des exemples issus d'articles de presse ou de textes littéraires. C'est pour cela que j'ai utilisé le Lexicoscope. Il utilise plusieurs corpus, dont certains utilisés dans le cadre d'anciens projet du laboratoire.

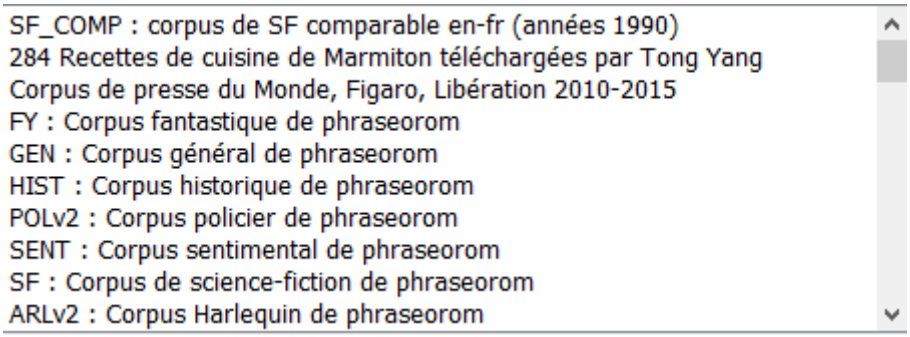


Illustration 4 : Exemples de corpus utilisés par le Lexicoscope.

La plupart des corpus sont des corpus littéraires issus du projet PhraseoRom, mais il existe aussi un corpus de presse (le Monde, Figaro, Libération) et un corpus de science fiction des années 1990.

Chaque corpus du Lexicoscope est annoté et numérisé. Il est donc possible de faire des requêtes via son interface. L'une d'elles est la « concordance ». Cette requête permet d'obtenir toutes les phrases où les mots sélectionnés apparaissent ensemble. Étant donné que je me suis focalisé sur les fonctions lexicales, je ne souhaitais obtenir que les contextes où deux mots apparaissaient ensemble. Par exemple pour « soleil de plomb », j'ai utilisé l'expression suivante :

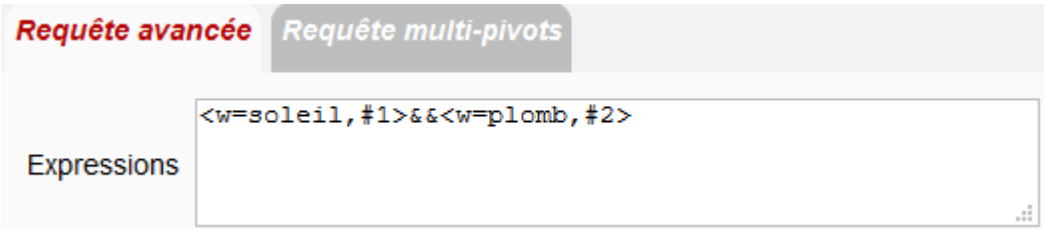


Illustration 5 : mot1=soleil et mot2=plomb.

Cette expression signifie que je souhaite obtenir tous les contextes où *soleil* et *plomb* sont dans la même phrase. Voici un exemple de résultat obtenu.

Num	Identifiant	Contexte gauche	Pivot	Contexte droit
1	s963	Les hommes rentrèrent vers une heure de l'après-midi, par une chaleur presque torride, sous un	soleil	de plomb.

Illustration 6 : résultat obtenu pour soleil+plomb dans le Lexicoscope.

3.1.5 - Différents corpus

Au total, 200 phrases ont été récoltées pour la fonction lexicale Bon ainsi que 200 autres pour la fonction lexicale Magn. Chaque fonction lexicale possède son propre corpus.

De plus, j'ai créé de nouveaux corpus à partir des corpus Bon et Magn durant le stage. Il s'agissait essentiellement de corpus de tests contenant 40 phrases, qui seront définis au fur et à mesure de ma présentation. Chaque corpus a été créé dans le but de répondre à une hypothèse en lien avec ma problématique.

Pour rappel, ma problématique était de savoir s'il était possible à FlauBERT d'encoder des informations propres aux collocations dans ses vecteurs et ainsi réussir à faire la différence entre des exemples issus de collocations et des exemples issus d'exemples de cooccurrences.

3.1.5.1 - Hypothèse principale

Mon hypothèse était que les exemples de collocations devaient être plus similaires entre-eux qu'ils ne le sont des exemples de cooccurrences. En effet, il existe un lien spécial entre la *base* et le *collocatif* d'une collocation dans le langage naturel, qui les différencie de simples cooccurrences (cf. 2.4 - Collocations, page 12). De plus, étant donné que FlauBERT permet de tenir compte du contexte entier d'une phrase pour construire ses vecteurs, je souhaitais savoir s'il lui était possible d'encoder dans ses vecteurs de mots, le lien spécial qu'entretiennent les collocations. En effet, dans sa façon de fonctionner, FlauBERT tient compte de tous les mots de la phrase. Ils sont donc étroitement liés et chacun des mots possède des informations sur les autres (cf. 2.9.2 - FlauBERT, page 18). Il n'est donc pas exclu que le lien qu'entretiennent une *base* et un *collocatif* puisse être différent de celui qu'entretiennent une base et un cooccurrent dans les vecteurs de mots de FlauBERT.

3.1.5.2 - Magn et Bon classés

Il s'agit d'un corpus de 400 phrases, obtenu en fusionnant les corpus *Magn* et *Bon*, puis en les classant par collocatifs/cooccurrents. Par exemple « grand journaliste » appartient au corpus *Bon* et « grand vent » appartient au corpus *Magn*. Ils possèdent tous deux le même collocatif. Ils ont donc été placés de manière à avoir 10 exemples de « grand journaliste », 10 exemples de « cooccurrent + journaliste » suivi par les exemples de « grand vent » et « cooccurrent + vent ».

3.1.5.3 - Silence et soleil de plomb

Il s'agit d'un corpus de 40 phrases, obtenu en prenant les 20 exemples de « soleil de plomb » et les 20 de « silence de plomb ». « Soleil de plomb » est constitué de 10 exemples de collocations et de 10 exemples de cooccurrences. « Silence de plomb » est constitué de 10 exemples de collocations et 10 exemples de cooccurrences.

Les collocations de « silence de plomb » et de « soleil de plomb » sont composées toutes les deux du collocatif *plomb.* Il s'agit d'exemples issus d'une même fonction lexicale avec une *base* différente.

Mon hypothèse était de dire que des collocations issues d'une même fonction lexicale, avec une base différente, seraient plus similaires entres-elles que des exemples libres, du fait de leur statut de collocation. Par exemple, « silence de plomb » serait plus similaire à « soleil de plomb » qu'il ne l'est de « lingot de plomb ».

3.1.5.4 - Soleil de plomb et colère saine

Il s'agit d'un corpus de 40 phrases, obtenu en prenant les 20 exemples de « soleil de plomb » et les 20 exemples de « colère saine ». « Soleil de plomb » est constitué de 10 exemples de collocations et de 10 exemples de cooccurrences. « Colère saine » est constitué de 10 exemples de collocations et 10 exemples de cooccurrences.

Les collocations de « soleil de plomb » et de « colère saine » sont issues d'une fonction lexicale différente. Toutefois, il s'agit dans les deux cas de collocations.

Mon hypothèse a été de dire que les collocations devraient être plus proches entre-elles que des cooccurrences, même s'il s'agit de deux fonctions lexicales différentes. En effet, une collocation est analysable par les fonctions lexicales, contrairement aux cooccurrences. Il y a donc une différence qui pourrait potentiellement être encodée dans les vecteurs de FlauBERT. Par exemple, « soleil de plomb » serait plus similaire à « colère saine » qu'il ne l'est de « lamelle de plomb ».

3.1.5.5 - Soleil de plomb et dégoût profond

Il s'agit d'un corpus de 40 phrases, obtenu en prenant les 20 exemples de « soleil de plomb » ainsi que les 20 exemples de « dégoût profond ». « Soleil de plomb » est constitué de 10 exemples de collocations et de 10 exemples de cooccurrences. « Dégoût profond » est constitué de 10 exemples de collocations et 10 exemples de cooccurrences.

Cette fois-ci, il s'agit de collocations issues de la même fonction lexicale, mais ayant une *base* et un *collocatif* différents.

Mon hypothèse était de dire qu'au sein d'une même fonction lexicale, les exemples issus de collocations seraient plus proches entres eux que de n'importe quel exemple de cooccurrences. Par exemple « soleil de plomb » devrait être plus similaire à « dégoût profond » qu'il ne l'est de « balle de plomb ».

3.2 - Organisation du corpus d'entrée

Cette partie s'intéresse à la manière dont j'ai organisé mes corpus d'entrée, c'est-à-dire les informations disponibles au sein de mes corpus. Tout au long de mon stage, j'ai ajouté de nouvelles données afin de répondre aux besoins de divers scripts. Je ne m'intéresserai qu'à la version finale des fichiers, qui est la plus aboutie. Toutefois l'ancienne version des corpus d'entrée est toujours disponible (cf. Annexe 1, page 61).

Chacun de mes corpus d'entrée a été constitué à la main en récupérant des exemples sur le Lexicoscope. Afin d'être facilement analysables par un algorithme, les fichiers sont dans un format tabulaire (fichier .tsv), c'est-à-dire que les colonnes sont séparées par des tabulations et les lignes par des retours à la ligne. De plus, les fichiers sont encodés en UTF8.

Voici une liste descriptive de chaque colonne ainsi que la raison de sa création (cf. Annexe 2, page 63 pour un aperçu).

3.2.1 - Données utilisées par les scripts

3.2.1.1 - Phrase

Il s'agit de la colonne correspondant à la phrase entière telle qu'elle a été récupérée sur le Lexicoscope. Certaines phrases sont issues de dialogues de romans mais il peut aussi s'agir d'articles de journaux. Il n'y a qu'une seule phrase par ligne.

3.2.1.2 - Base

Il s'agit du mot correspondant à la base tel qu'il est écrit dans la phrase. Afin de pouvoir utiliser mes scripts sur le corpus, il fallait que le mot écrit dans cette colonne soit identique à celui écrit dans la phrase. Par exemple, si un mot était écrit au masculin pluriel dans la phrase, il devait aussi être écrit au masculin pluriel dans la colonne Base.

3.2.1.3 - Genre de la base

Il s'agit du genre de la base. Par exemple, si la base était féminine, j'ai alors écrit « féminin » dans cette colonne. J'ai effectué cette tâche manuellement et ai vérifié à plusieurs reprises l'exactitude de ce que j'avais écrit.

3.2.1.4 - Nombre de la base

Il s'agit du nombre de la base. Par exemple, si la base était au singulier, j'ai alors écrit « singulier » dans cette colonne. J'ai effectué cette tâche manuellement et ai vérifié à plusieurs reprises l'exactitude de ce que j'avais écrit.

3.2.1.5 - Collocatif

Il s'agit du mot correspondant au collocatif ou au cooccurrent tel qu'il est écrit dans la phrase. Afin de pouvoir utiliser mes scripts sur le corpus, il fallait que le mot écrit dans cette colonne soit identique à celui écrit dans la phrase. Par exemple, si un mot était écrit au masculin pluriel dans la phrase, il devait aussi être écrit au masculin pluriel dans la colonne Collocatif.

3.2.1.6 - Genre du collocatif

Il s'agit de la colonne correspondant au genre du collocatif ou au cooccurrent. Par exemple, si le collocatif/cooccurrent était au masculin, j'ai alors écrit « masculin » dans cette colonne. J'ai effectué cette tâche manuellement et ai vérifié à plusieurs reprises l'exactitude de ce que j'avais écrit.

3.2.1.7 - Nombre du collocatif

Il s'agit du nombre du collocatif. Par exemple, si le collocatif était au pluriel, j'ai alors écrit « pluriel » dans cette colonne. J'ai effectué cette tâche manuellement et ai vérifié à plusieurs reprises l'exactitude de ce que j'avais écrit.

3.2.1.8 - Lemme collocatif

Il s'agit d'une colonne disponible uniquement pour le collocatif/cooccurrent. Pour faire fonctionner l'un de mes scripts il me fallait en effet cette information, afin de compter les occurrences de lemmes. Elle n'a pas été utile pour la base, donc je n'ai créé qu'une seule colonne.

3.2.1.9 - Compositionnel

Le nom de cette colonne est un abus de langage de ma part, j'aurais plutôt dû indiquer « Combinaison libre ». En effet, s'il s'agissait d'une cooccurrence, j'ai écrit « oui », afin

d'indiquer qu'il s'agissait d'une combinaison libre de mots, non analysable par les fonctions lexicales. Au contraire, s'il s'agissait d'une collocation, j'ai alors écrit « non » afin indiquer qu'il s'agissait d'un exemple de mots non analysables par les fonctions lexicales.

3.2.2 - Métadonnées

Voici les dernières colonnes disponibles au sein de mes corpus. Elles n'ont pas été utilisées pour mes scripts, mais possèdent des informations concernant les métadonnées. Elles pourront potentiellement être utiles à d'autres personnes souhaitant connaître l'origine des corpus.

3.2.2.1 - Corpus

Il s'agit du corpus dont a été extraite la phrase. Je n'ai pas privilégié un corpus plutôt qu'un autre et ai essayé de varier le plus possible les corpus utilisés. Cependant le corpus « LITT : Sélection d'oeuvres de PhraseoRom (5M[3] par sous-genre) » possédait beaucoup d'exemples analysables. Étant donné qu'il s'agit d'un corpus de sélection de plusieurs œuvres, certaines sont issus des autres corpus de PhraseoRom présents dans la liste.

La liste des corpus utilisés pour la constitution de mon jeu de données est la suivante :
FY : Corpus fantastique de PhraseoRom
Corpus de presse du Monde, Figaro, Libération 2010-2015
LITT : Sélection d'œuvres de PhraseoRom (5M par sous-genre)
HIST : Corpus historique de PhraseoRom
GEN : Corpus général de PhraseoRom
CONT : Corpus de contraste de PhraseoRom
SF : Corpus de science-fiction de PhraseoRom
SENT : Corpus sentimental de PhraseoRom
SF_COMP : corpus de SF comparable en-fr (années 1990)
POLv2 : Corpus policier de PhraseoRom
ARLv2 : Corpus Harlequin de PhraseoRom

3.2.2.2 - Identifiant (ID)

Il s'agit de l'identifiant de la phrase dans son corpus d'origine. Si le corpus d'origine est « Corpus de presse du Monde, Figaro, Libération 2010-2015 » et que l'ID est « s7764 », cela veut dire qu'il s'agit de la phrase 7764 de ce corpus.

3 Cinq millions de mots.

3.2.2.3 - URL

Il s'agit de l'adresse internet (URL) à laquelle on peut récupérer toutes les informations disponibles concernant la phrase. En cliquant dessus, nous sommes renvoyés sur une page contenant plusieurs informations, comme le contexte immédiat de la phrase ou d'autres données, comme la date de publication si celle-ci a été indiquée.

3.3 - Exploration du corpus (Partie informatique)

Nous allons maintenant nous intéresser à la deuxième partie de mon stage. Celle-ci concerne la partie informatique et plus particulièrement le développement de mes scripts.

Dans un premier temps, nous verrons les systèmes d'exploitation et logiciels que j'ai utilisés, nécessaires au bon fonctionnement des scripts.

Dans un deuxième temps, nous verrons quels scripts j'ai développés ainsi que leurs spécificités. Il ne s'agira que d'une explication synthétique.

3.3.1 - Environnement et langage de programmation utilisés

3.3.1.1 - Windows 10 et Ubuntu

J'ai utilisé deux ordinateurs durant le stage, l'un équipé du système d'exploitation « Windows 10 » et l'autre équipé de « Ubuntu 18.04.4 LTS ». Je n'ai pas rencontré de quelconque problème et toutes les installations de logiciels que j'ai faites sur Ubuntu ont aussi fonctionné avec Windows 10.

3.3.1.2 - Git

La totalité de mes fichiers a été versionnée grâce à Git. Il s'agit d'un logiciel libre, qui permet notamment de téléverser des fichiers sur un serveur ou encore de les télécharger depuis ce même serveur.

Il possède un système de suivi des bugs ainsi qu'un historique des versions déposées. Il est en effet possible de revenir à une version précédente en cas de problème. J'ai pour ma part utilisé le serveur Gitlab proposé par l'UGA pour déposer mes fichiers. Je n'ai essentiellement fait que téléverser mes fichiers au fur et à mesure de l'avancement de mon stage.

Tout d'abord, cela m'a permis de partager ce que j'avais fait avec mes encadrants de stage. Puis, en cas de problème j'avais aussi une copie de chaque étape du stage et pouvais revenir à d'anciennes versions.

De plus, j'ai eu la possibilité de créer un fichier récapitulatif de toutes mes expérimentations, balisé en Markdown. Il s'agit d'un langage de balisage très simple à comprendre et à utiliser.

3.3.1.3 - Python

L'intégralité de mes scripts a été développée avec le langage de programmation Python (version 3.7.6). Tout d'abord, Python a pour avantage de pouvoir être installé très facilement sur Windows et est installé de manière native avec Ubuntu. De plus, il est très facile d'utilisation, mais ce n'est pas pour autant qu'il ne permet pas d'effectuer des tâches avancées.

La documentation pour Python est de plus très étoffée et très claire. Il existe de nombreux tutoriels disponibles sur internet ainsi que plusieurs questions posées sur des forums, dont les réponses ont pour ma part permis de résoudre certains de mes problèmes.

Pour finir, il a été nécessaire d'utiliser Python durant mon stage car certaines librairies obligatoires de mon stage comme FlauBERT ne fonctionnent qu'avec Python.

3.3.1.4 - Anaconda

Il s'agit d'une distribution libre, très utilisée en apprentissage machine et possédant plusieurs milliers de packages et librairies pour Python. L'avantage d'Anaconda est qu'il permet de créer un environnement virtuel.

Cet environnement virtuel permet d'avoir une version de Python installée dans cet espace virtuel, qui fonctionnera avec tous les packages et librairie dont on a besoin. Cela évite certains conflits entre différents packages et librairies qui ne fonctionnent pas forcément de la même manière selon leur version.

3.3.1.5 - Geany

J'ai utilisé l'éditeur de texte Geany tout au long du stage car il s'agit d'un éditeur de texte que j'ai beaucoup utilisé durant mon master. Il possède une interface assez simple mais il modifie la couleur du code, permettant d'éviter plusieurs erreurs possibles lors de l'écriture de scripts.

3.3.1.6 - PyTorch

PyTorch a été l'une des bibliothèques de Python m'ayant le plus servi durant mon stage. Elle m'a notamment permis de manipuler facilement les types de données que j'ai

utilisés, comme les vecteurs de mots ou les matrices de similarité. Cette bibliothèque a donc été primordiale dans la réussite de mon stage.

3.3.1.7 - FlauBERT

Le modèle FlauBERT a été au centre de tout mon stage. En effet j'ai eu l'occasion de l'utiliser sur chacun de mes corpus afin d'obtenir des vecteurs contextuels (cf. 2.9.2 - FlauBERT, page 18).

Nous allons maintenant voir les différentes librairies nécessaires au bon fonctionnement de mes scripts.

3.3.2 - Librairies et packages utilisés

3.3.2.1 - argparse

Il s'agit d'un module déjà installé avec Python 3.7. Il permet de gérer les arguments et options donnés aux scripts en ligne de commande.

3.3.2.2 - NumPy

Numpy permet de créer des tableaux à plusieurs dimensions et m'a notamment permis de pouvoir écrire mes vecteurs, qui contiennent des nombres à virgules (*float*), dans mes fichiers de sortie.

3.3.2.3 - Agglomerative Clustering et dendrogram

Ces deux modules sont utilisés afin de construire des dendrogrammes, qui sont des diagrammes permettant d'illustrer des similarités. Ils utilisent le principe de regroupement hiérarchique ascendant, qui consiste à regrouper les données par similarités.

Le module « Agglomerative Clustering » fait partie de la bibliothèque « sklearn » et permet de réaliser le partitionnement des données (*clustering*). Le module « dendrogram » fait partie de la bibliothèque « scipy » et permet de configurer son dendrogramme, en modifiant par exemple la taille des feuilles, l'orientation de la figure ou encore les étiquettes utilisées.

3.3.2.4 - Seaborn

Seaborn est un module de Python permettant de pouvoir créer des *heatmaps*, c'est-à-dire une figure dont la couleur peut varier à certains endroits selon plusieurs paramètres, comme la similarité entre deux vecteurs.

3.3.2.5 - Pyplot

Pyplot est un module de « matplotlib ». Il m'a permis durant mon stage de visualiser mes figures comme les dendrogrammes ou *heatmaps* et de les sauvegarde sur mon disque dur sous divers formats (png, pdf).

3.3.2.6 - Pandas

Pandas est une librairie de Python qui permet de manipuler des données avec plus de facilité. Je ne l'ai que très peu utilisée mais elle m'a notamment permis de pouvoir rajouter des colonnes dans mes données sans pour autant devoir modifier des anciennes lignes de code.

3.3.3 - Développement des scripts

J'ai eu la possibilité de développer plusieurs scripts durant mon stage. Chacun d'eux a été créé dans le but de répondre à ma problématique qui était de savoir si FlauBERT était capable de reconnaître des collocations et que si tel était le cas, il serait possible d'en avoir la confirmation grâce à ses vecteurs de mots. Je parlerai des scripts que j'ai développés par ordre chronologique.

Tout d'abord, j'ai commencé par un script permettant l'extraction des vecteurs de mots d'une phrase en utilisant FlauBERT. Il s'agissait de la première étape obligatoire afin de pouvoir tirer des informations des vecteurs de FlauBERT.

Par la suite, j'ai développé un script permettant de créer une matrice de similarité (cf. page 37). À partir de cette matrice j'ai développé un autre script permettant de la transformer en *heatmap* (cf. page 37). Cette visualisation sous forme de *heatmap* a été très importante car elle a permis la visualisation des similarités entre les vecteurs, sous forme d'images colorées.

Ensuite, j'ai développé un autre script permettant de créer des dendrogrammes à partir de vecteurs. Cette fois-ci la visualisation des similarités entre vecteurs était sous la forme d'un diagramme où les vecteurs étaient classés par similarités de manière automatique.

Il a été intéressant d'utiliser deux méthodes différentes afin de classer mes vecteurs, car les représentations n'étaient pas identiques, l'une sous la forme d'un tableau coloré et l'autre sous la forme d'un arbre.

Après avoir terminé ces scripts, je les ai adaptés afin de pouvoir les utiliser avec des vecteurs de fastText. Cela m'a permis comparer les résultats de fastText avec ceux de FlauBERT.

Pour finir, j'ai développé un script de réseau de neurones permettant de comparer à quel point FlauBERT arrivait à reconnaître les collocations par rapport à fastText ou un modèle *baseline* (cf. page 40).

Tous mes scripts développés durant le stage sont désormais disponibles sur un serveur Github (cf. Annexe 1, page 61). Il s'agit d'une copie de la dernière version de mes fichiers qui étaient sur le serveur Gitlab de l'UGA. Ne sachant pas jusqu'à quand mon projet serait conservé sur Gitlab, j'ai préféré créer une copie sur Github qui est utilisable gratuitement.

3.3.4 - Fonctionnement général des scripts

Avant toute chose, mes scripts, dans leur version actuelle, fonctionnent avec FlauBERT et fastText. Ils n'ont pas été testé avec d'autres modèles. Les boucles utilisées pour parvenir au même résultat diffèrent entre FlauBERT et fastText. En effet, l'utilisation de FlauBERT est plus complexe et il a fallu plusieurs étapes additionnelles pour parvenir à des fichiers de résultats similaires. C'est pour cela que pour chaque script, j'indiquerai d'abord comment cela fonctionne pour FlauBERT, puis pour fastText si cela est nécessaire.

Pour finir, afin de réduire le nombre de lignes de code mais aussi rendre l'utilisation de mes scripts plus fluide, j'ai préféré créer un script pour chaque tâche à effectuer. Ces scripts font appel à des fonctions qui se trouvent toutes regroupées au sein d'un script principal.

3.3.5 - Spécificités de plusieurs scripts

Certains de mes scripts possèdent plusieurs particularités nécessaires afin de tenir compte de différents cas de figure possibles. Il s'agit des scripts de matrices de similarité, de dendrogrammes ainsi que le réseau de neurones. Les résultats obtenus sont différents selon le choix de l'utilisateur ainsi que le script exécuté.

Tout d'abord, lorsque ces 3 types de scripts sont exécutés, une question est posée à l'utilisateur « Voulez vous calculer la base et le collocatif ? oui/non ». L'utilisateur doit répondre par « oui » ou « non » à la question. S'il choisit « oui », cela indique au script qu'il doit faire une soustraction entre le vecteur de la base et celui du collocatif/cooccurrent. À l'inverse, si l'utilisateur choisit « non », cela indique au script qu'il ne faut calculer que le vecteur du collocatif/cooccurrent. Le fichier de résultat sera alors différent selon le choix de l'utilisateur.

Je vais maintenant expliquer pourquoi il est utile d'effectuer ou non cette soustraction de vecteurs.

3.3.5.1 - Pourquoi soustraire les deux vecteurs ?

Il existe plusieurs manières de représenter la cooccurrence, c'est-à-dire la présence de deux mots au sein d'une même phrase. La 1ère manière consiste à soustraire un collocatif/cooccurrent à sa base (base - collocatif/cooccurrent). Cette soustraction permet d'identifier l'information apportée par le collocatif, dans un rapport d'analogie. Il est possible de l'effectuer avec FlauBERT et fastText car la soustraction de vecteur engendre un nouveau vecteur porteur d'informations sur cette analogie.

Par ailleurs, une deuxième manière consiste à ne garder que le collocatif/cooccurrent. Cependant, elle ne fonctionne que pour FlauBERT ou tout autre modèle de langage possédant des vecteurs contextuels. En effet, les vecteurs de FlauBERT sont créés de manière à tenir compte du contexte entier de la phrase. Par exemple, le collocatif *profond* de la collocation « dégoût profond » ne sera potentiellement pas encodé de la même manière que dans la cooccurrence « trou profond » par FlauBERT. En ne prenant que le collocatif/cooccurrent, celui-ci peut être porteur d'informations sur les bases *dégoût* et *trou.*

Les vecteurs possèdent plusieurs coefficients. Par exemple FlauBERT possède 768 coefficients pour chaque vecteur et fastText en possède 300 pour chaque vecteur. Les scripts de calculs de similarité et de dendrogrammes font une soustraction de chaque coefficient du collocatif/cooccurrent à chaque coefficient de la base, c'est-à-dire que le 1er coefficient du collocatif/cooccurrent est soustrait au 1er coefficient de la base. Il en est de même pour les 767 autres coefficients de FlauBERT et les 299 autres coefficients de fastText.

De plus, il existe une 3e méthode que j'ai utilisée uniquement pour mon réseau de neurones. Elle consiste à concaténer les deux vecteurs. Cette fois-ci, au lieu de soustraire chaque coefficient, on les concatène. Cela donne un nouveau vecteur de 1536 coefficients pour FlauBERT (768x2) et 600 coefficients pour fastText (300x2). Il s'agit en fait de donner l'information globale sur la collocation, afin de déterminer si le réseau de neurones peut apprendre à reconnaître les collocations.

La soustraction de vecteurs est utilisée par les scripts de calculs de similarité ainsi que les scripts de création de dendrogrammes et la concaténation est utilisée uniquement par le réseau de neurones.

Pour finir, si l'utilisateur souhaite obtenir des matrices de similarité ou des dendrogrammes, il lui est possible de choisir entre deux alternatives représentées par deux scripts différents. La première alternative permet au script de prendre la totalité des collocations, c'est-à-dire prendre les 10 occurrences de « soleil de plomb » pour le cas de « soleil de plomb ». La seconde alternative indique au script qu'il ne doit choisir qu'une seule

collocation parmi les 10 occurrences disponibles. Par exemple, pour « soleil de plomb » le script ne choisira qu'une seule collocation au lieu des 10.

Je vais maintenant expliquer l'intérêt de ces deux alternatives possibles.

3.3.5.2 - Pourquoi ne prendre qu'un exemple ?

Nous avons remarqué que les 10 instances issues de collocations étaient très similaires entre elles au sein d'un même ensemble de 20 exemples. Cela s'explique par le fait qu'il s'agisse de la même collocation à 10 reprises, donc 10 fois soleil + plomb. Quand la classification automatique regroupe ces dix instances, cela peut être dû simplement à la présence de la base, indépendamment de l'interprétation du collocatif. Si l'on veut pouvoir distinguer les cas qui fonctionnent avec les fonctions lexicales des cas de combinaison libre, il faut faire varier les bases. Ainsi, au lieu de prendre 10 occurrences de *soleil* et *plomb*, le 2^{e} script ne prend qu'une seule occurrence. Des groupes de 11 exemples sont alors créés, au lieu de groupes de 20. Ce qui donne 10 exemples de combinaisons libres et un exemple à combinaison restreinte (collocation). Le défi sera alors de reconnaître, parmi ces 11 cas, lequel correspond à un exemple de fonction lexicale.

Cependant, ce cas de figure n'est utile que pour FlauBERT. En effet, fastText possède des vecteurs statiques, il n'y a pas besoin de choisir aléatoirement une occurrence de « soleil + plomb », car il s'agit à 10 reprises des mêmes vecteurs.

3.3.6 - Scripts d'extraction du vecteur de la base et du collocatif

3.3.6.1 - Description générale des scripts

Les scripts d'extraction de vecteurs permettent d'extraire les vecteurs correspondant à la base et au collocatif/cooccurrent pour chaque phrase. Il y a au total deux scripts pour effectuer cette tâche. L'un pour extraire des vecteurs de FlauBERT et l'autre pour extraire les vecteurs de fastText.

3.3.6.2 - Fonctionnement du script pour FlauBERT

Le script utilise le *tokenizer*[4] de FlauBERT sur la phrase entière. Chaque token est alors analysé afin d'obtenir son vecteur. Cependant, dans sa manière de tokeniser, FlauBERT découpe les tokens en plusieurs morceaux. Par exemple le token « lévrier » est découpé en deux parties : « l »suivi de « évrier ».

4 Il s'agit du terme anglais utilisé pour désigner un programme informatique d'analyse lexicale.

Il y a alors deux morceaux différents pour un seul et même token initial. Cette méthode de segmentation permet notamment de gérer les mots qui ne sont pas dans le vocabulaire de FlauBERT (o*ut of vocabulary words).*

Néanmoins, FlauBERT possède un vocabulaire constitué de ces morceaux de tokens ainsi que leur correspondance sous forme d'identifiant.

Par exemple, *lévrier* possède les identifiants 3600 pour « l », et 8100 pour « évrier ». Cela permet au script de pouvoir récupérer le vecteur correspondant à « l » de « lévrier » à l'aide de son identifiant dans le vocabulaire. Toutes les parties du token initial sont porteuses d'informations. Cependant, le script prend toujours le vecteur correspondant à la 1ère partie du token.

De plus, certaines phrases possèdent plusieurs occurrences du token correspondant à la base ou au collocatif, mais jamais les deux à la fois. Afin de tenir compte de ce cas de figure, le script choisit de manière automatique le bon couple au sein de la phrase. En effet, comme FlauBERT possède des vecteurs contextuels, chaque occurrence d'une base ou d'un collocatif/cooccurrent possède un vecteur différent.

Pour ce faire, le script compte le nombre de tokens dans la phrase et leur attribue un indice selon leur position dans la phrase, en commençant par 0. Par exemple dans la phrase suivante des identifiants sont assignés automatiquement à chaque token :

Le$_0$ lac$_1$ du$_2$ Lauvitel$_3$ est$_4$ un$_5$ lac$_6$ très$_7$ profond$_{8.9}$

Une soustraction entre *profond* et les deux occurrences de *lac* est effectuée. Le résultat le plus proche de 0 est alors sélectionné par le script.

Une fois les bons couples sélectionnés, le script crée un fichier en sortie contenant le vecteur de la base et celui du collocatif/cooccurrent (cf. Annexe 3, page 64).

3.3.6.3 - Fonctionnement du script pour fastText

FastText possède un fichier de vecteurs fixes qu'il faut télécharger au préalable.
Tout d'abord, le script parcourt le fichier d'entrée et récupère les bases et collocatifs/cooccurrents nécessaires. Il récupère ensuite le vecteur associé à chaque base et à chaque collocatif/cooccurrent dans le fichier. Étant donné que les vecteurs sont statiques, cette opération est très rapide car une base ou collocatif/cooccurrent déjà rencontré possède un vecteur identique aux précédentes occurrences.

Un fichier de sortie est alors créé, contenant le vecteur de la base ainsi que celui du collocatif (cf. Annexe 4, page 65).

3.3.7 - Matrice de similarité et *heatmap*

Les scripts de matrices de similarité et *heatmaps* permettant de créer une matrice de similarité suivie d'une *heatmap*. Il y en a 3 au total, dont 2 fonctionnant avec FlauBERT et le dernier fonctionnant avec fastText.

3.3.7.1 - Description générale

Ces scripts fonctionnent en deux parties. Tout d'abord, un parcours du fichier de vecteurs obtenu en sortie du précédent script est effectué (cf. Annexe 3 et Annexe 4). Ensuite, une matrice de similarité est créée (cf. Illustration 7 ci-dessous), puis transformée en *heatmap* (cf. Illustration 8, page 38).

3.3.7.2 - Détails des scripts

Matrice de similarité

La 1ère partie des scripts est dédiée à la création d'une matrice de similarité. Il s'agit d'un tableau à 2 dimensions. La similarité est calculée grâce à la similarité cosinus entre deux vecteurs. La valeur de similarité est comprise entre -1 et 1. Plus elle est proche de 1, plus les deux vecteurs sont similaires. Voici un exemple :

	A	B	C	D
1		Non/colère/saine	Non/colère/saine	Non/colère/saine
2	Non/colère/saine	1.0	0.6803779602050781	0.5601814985275269
3	Non/colère/saine	0.6803779602050781	1.0	0.547079861164093
4	Non/colère/saine	0.5601814985275269	0.547079861164093	1.0

Illustration 7 : exemple matrice de similarité 3x3.

Si le fichier en entrée contient 3 vecteurs, la matrice sera représentée par 3 colonnes et 3 lignes. À l'intérieur de la matrice se trouveront les résultats de la similarité cosinus entre les deux axes du tableau. La similarité entre le vecteur B1 et A3 se trouve en B3. Une valeur de 1.0 indique qu'il s'agit du même vecteur sur les deux axes (diagonale de la matrice).

Heatmap

La 2e partie des scripts permet de créer une *heatmap* à partir de la matrice de similarité. Il s'agit d'une figure représentant la matrice de similarité sous forme d'un tableau coloré. Chaque nombre est transformé en une couleur plus ou moins claire. Une similarité proche de -1 tend vers le noir, tandis qu'une similarité de 1 tend vers le blanc. Il y a toutefois des variations de couleurs entre ces deux valeurs. Plus la couleur est claire, plus la similarité est proche de 1. Voici un exemple issu de mon corpus :

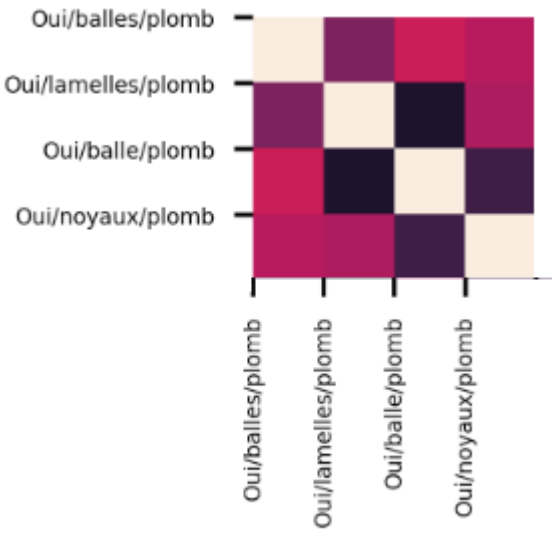


Illustration 8 : exemple de *heatmap* 4x4 (base-collocatif).

Dans cet exemple, on peut remarquer que « balle de plomb » est, d'après les résultats, plus similaire à « balles de plomb » qu'il ne l'est à « lamelles de plomb ».

Le script crée une matrice de similarité de taille 40x40, 200x200 ou encore 400x400 selon le corpus utilisé en entrée (cf. page 24). Par exemple, si le corpus en entrée est «silence et soleil de plomb », alors la matrice sera une matrice de taille variable, définie selon le script utilisé (cf. 3.3.5.2 - Pourquoi ne prendre qu'un exemple ?, page 35).

3.3.7.3 - Fonctionnement pour FlauBERT

Les scripts de matrices de similarité et *heatmap* analysent la totalité des exemples du corpus et effectuent des opérations différentes selon le script sélectionné par l'utilisateur. Si l'utilisateur souhaite des groupes de 11 exemples, les résultats ne seront pas les mêmes que le script qui sélectionne des groupes de 20 exemples. Pour le corpus « soleil de plomb et silence de plomb » (cf. page 24), le résultat sera soit sous la forme d'un *heatmap* de 22 exemples, 11 pour « soleil de plomb » et 11 pour « silence de plomb », soit sous la forme d'une *heatmap* de 40 exemples. De plus, il est aussi possible à l'utilisateur de choisir s'il souhaite le collocatif seul ou l'ensemble base-collocatif. (cf. page 33).

3.3.7.4 - Fonctionnement pour fastText

Le script de matrice de similarité et *heatmap* pour fastText analyse la totalité des exemples du corpus et effectue des opérations différentes selon le choix de l'utilisateur, c'est-à-dire s'il souhaite l'ensemble base-collocatif ou collocatif seul. Cependant, il n'y a pas de choix possible entre 11 exemples et 20 exemples par l'utilisateur. En effet, le script n'a pas

besoin de comparer chaque vecteurs entre eux. Étant donné qu'il s'agit de vecteurs fixes, si plusieurs occurrences d'un même couple de mots sont rencontrées (par exemple « soleil de plomb »), alors le script ne garde qu'une seule d'entre elles, sachant que toutes les autres occurrences possèdent les mêmes vecteurs.

Pour les corpus de 40 exemples, il s'agira d'une matrice de similarité et *heatmap* de 22 exemples au minimum. En effet, il est possible qu'une base soit utilisée à plusieurs reprises. Certaines collocations telles que « colère saine » sont aussi utilisées au pluriel dans mes corpus. De plus, certains couples de bases et cooccurrents peuvent théoriquement être présents à plusieurs reprises entre les différents corpus, car le corpus *Magn* ne tient pas compte des cooccurrences du corpus *Bon* et vice-versa. En effet, si l'on compare « grand journaliste » et « grand ami », qui ne sont pas issus du même corpus, il est possible que les deux mots utilisés par l'un soit aussi utilisé par l'autre.

3.3.8 - Dendrogrammes

3.3.8.1 - Description générale

Ce script permet de créer des dendrogrammes à partir d'un fichier de vecteurs. Un calcul de similarité cosinus est effectué entre tous les vecteurs afin de les regrouper par similarité. Un dendrogramme représentant ces similarités est alors créé.

3.3.8.2 - Détails du script

La représentation des vecteurs sous forme de dendrogramme est une méthode de regroupement hiérarchique. Pour ce faire, le script commence d'abord par récupérer chaque vecteur. Un calcul de similarité cosinus est alors effectué entre chacun des vecteurs. Les vecteurs ayant une similarité proche de 1 seront agglomérés. Finalement, plusieurs groupes de vecteurs sont créés et leur proximité est représenté par les branches d'un arbre. Plus les branches sont proches des feuilles de l'arbre, plus les exemples regroupés ensemble sont proches. Voici un exemple :

Illustration 9 : exemple de dendrogramme.

Dans cet exemple, nous pouvons remarquer que le dendrogramme est séparé en plusieurs groupes. Un groupe composé de « bille de plomb», « balle de plomb » et « masse de plomb » et un autre composé de « cage de plomb », « tablette de plomb » et « plaque de plomb ». Plus haut dans l'arbre, les deux groupes se rejoignent (représenté par le trait bleu). D'après le dendrogramme, le vecteur correspondant à *cage* est plus similaire à celui de *tablette* qu'il ne l'est à celui de *plaque*. Il est cependant plus similaire à *plaque* qu'il ne l'est de *bille*. En effet, ils ne font pas partie du même groupe et sont rejoints par le trait bleu qui est beaucoup plus haut dans l'arbre.

3.3.8.3 - Fonctionnement pour FlauBERT et fastText

Les scripts de dendrogrammes analysent la totalité des exemples du corpus et effectuent des opérations différentes selon le choix de l'utilisateur. Les questions et options suivent le même principe que pour les matrices de similarité et *heatmaps*. (cf. 3.3.7.3 - Fonctionnement pour FlauBERT et 3.3.7.4 - Fonctionnement pour fastText).

3.3.9 - Entraînement d'un réseau de neurones

3.3.9.1 - Fonctionnement général

Ce script est un réseau de neurones à propagation avant, permettant d'entraîner un classifieur automatique sur des vecteurs. Le classifieur est entraîné sur des vecteurs de mots de FlauBERT dans un cas et fastText dans l'autre, puis comparé à un « système *baseline* ». Le système *baseline* est un système qui choisit toujours la réponse dont la probabilité est la plus élevée par rapport au genre de la base (féminin/masculin). L'intérêt est de savoir à quel point un classifieur peut prédire s'il s'agit d'une collocation ou d'une simple cooccurrence.

3.3.9.2 - Détails du script

Le script commence tout d'abord par récupérer tous les vecteurs du fichier en entrée. Il peut s'agir de vecteurs issus d'une soustraction d'une base et d'un collocatif/cooccurrent ou d'un collocatif/cooccurrent seulement. Étant donné qu'il s'agit d'un apprentissage supervisé, le script possède aussi les étiquettes de référence correspondants aux vecteurs. Il s'agit des réponses attendues dont nous avons connaissance de l'exactitude, car elles sont étiquetées à la main.

Le classifieur est alors entraîné sur des vecteurs et les étiquettes de référence leur correspondant. Il apprend ainsi de manière automatique à différencier une cooccurrence d'une collocation. Par exemple si le vecteur correspond à la collocation « colère saine » et que

l'étiquette de référence est « 0 », alors cela indique que ce vecteur correspond à une collocation et non pas une cooccurrence.

Étant donné que les corpus ne possèdent au maximum que 400 vecteurs, il a été nécessaire de faire une validation croisée (*cross validation*), c'est-à-dire entraîner le classifieur sur 380 vecteurs et le tester sur les 20 vecteurs restants, en prenant soin de ne pas inclure les mêmes vecteurs durant l'entraînement et le test. De plus, le corpus étant trop petit il n'y a pas eu de phase d'évaluation, d'où l'intérêt de la validation croisée.

Le script découpe donc le corpus en plusieurs parties de 20 vecteurs. Étant donné que les vecteurs sont dans le même ordre d'apparition que dans le corpus, les groupes de 20 vecteurs correspondent forcément au même collocatif/cooccurrent. Par exemple sur le corpus de 200 vecteurs, il y aura 10 groupes de 20 vecteurs. Le 1er groupe correspondra à « soleil de plomb » (10 exemples libres et 10 exemples de collocations). Le script commence donc par sélectionner 20 exemples qui seront exclus lors de l'entraînement et qui seront utilisés uniquement pour tester le classifieur. Cela fait au total 10 entraînements sur 180 exemples et 10 tests sur 20 exemples. Ensuite, une moyenne de la précision des 10 entraînements/tests est effectuée afin d'obtenir un pourcentage global de précision. La précision correspond à la capacité du classifieur à donner la bonne réponse. La bonne réponse étant celle de l'étiquette de référence, une comparaison entre les réponses du classifieur et les étiquettes de référence est effectuée. Pour finir, la précision du système *baseline* reste la même peu importe le cas (ensemble base-collocatif ou collocatif seul).

3.3.9.3 - Entraînement sur FlauBERT

Dans un premier temps, j'ai commencé par comparer le classifieur entraîné sur les vecteurs de FlauBERT à mon système *baseline*. Cela a permis de mettre en évidence à quel point FlauBERT était capable de reconnaître la différence entre des vecteurs issus de collocations et des vecteurs issus de cooccurrences. J'ai effectué cette manipulation avec des vecteurs de collocatifs/cooccurrents seuls mais aussi des vecteurs issus de soustraction d'une base et d'un collocatif/cooccurrent.

3.3.9.4 - Entraînement sur fastText et FlauBERT

Dans un second temps, j'ai comparé le classifieur entraîné sur des vecteurs de FlauBERT au classifieur entraîné sur les vecteurs de fastText. Cependant, pour représenter les vecteurs de FlauBERT et fastText, j'ai eu recours à la méthode de concaténation de vecteurs (cf. page 34).

4 - Résultats et interprétations

Cette partie exposera les résultats que j'ai obtenus suite à mes expérimentations, suivis d'une interprétation des résultats.

Pour rappel, ma question initiale était de savoir si FlauBERT était capable d'encoder des informations concernant les fonctions lexicales dans ses vecteurs de mots et ainsi différencier les collocations des cooccurrences. Pour ce faire j'ai effectué plusieurs expérimentations durant mon stage qui ont permis d'obtenir des informations pouvant m'aider à répondre à ma question initiale. Cependant, je ne me focaliserai que sur certains résultats, dont je parlerai tout au long ce cette partie.

Dans un premier temps, j'analyserai les résultats obtenus à partir des scripts de *heatmaps* et des scripts de dendrogrammes sur les 3 corpus de 40 exemples, car ils sont plus facilement analysables et ont été créés dans le but spécifique de vérifier mes hypothèses (cf. page 25). Je me focaliserai donc sur les fichiers de résultats possédant 22 exemples et j'analyserai les résultats obtenus par le calcul base-collocatif, avant de m'intéresser à ceux issus du calcul du collocatif seul (cf. pages 33, 38 et 40).

Chaque corpus de test concernant les *heatmaps* et dendrogrammes que je vais analyser est constitué de la manière suivante : 22 exemples, dont 20 issus de cooccurrences libres et 2 issus de collocations. Par exemple pour « Silence et soleil de plomb », il s'agira d'une collocation provenant de « silence de plomb » et une autre provenant de « soleil de plomb ». Le but est alors de déterminer si les 2 collocations sont plus proches entre elles qu'elles ne le sont du reste des exemples. De plus, je me concentrerai uniquement sur les résultats obtenus par FlauBERT car ils sont en rapport direct avec ma problématique.

Dans un second temps, j'exposerai les résultats obtenus par mon classifieur automatique sur les 3 corpus suivants : « FL_MAGN », « FL_BON » et « FL_MAGN_BON_CLASSE » (cf. page 24). J'analyserai les résultats obtenus par le classifieur sur des vecteurs issus de la soustraction d'une base et d'un collocatif, ainsi que sur des vecteurs issus de collocatifs seuls (cf. page 33).

Par ailleurs, la totalité des figures et résultats que j'ai obtenus, y compris ceux de fastText, sont disponibles dans le dossier de mon stage. Comme certaines figures sont très grandes, il ne m'est pas possible de les montrer entièrement dans cette partie. Les illustrations ne seront que des aperçus des résultats. Pour récupérer les figures, il suffit de suivre les instructions en Annexe 1 Téléchargement du projet, à la page 61 ou alors de les voir en annexes (cf. Table des annexes, page 60).

4.1 - Comparaison soleil de plomb et silence de plomb

4.1.1 - *Heatmaps*

Voici donc mon analyse concernant les deux *heatmaps* de « silence de plomb » et « soleil de plomb » (cf. page 25). Les illustrations ne sont que des aperçus. Pour voir les *heatmaps*, cf. pages 66 et 67.

4.1.1.1 - Analyse base-collocatif

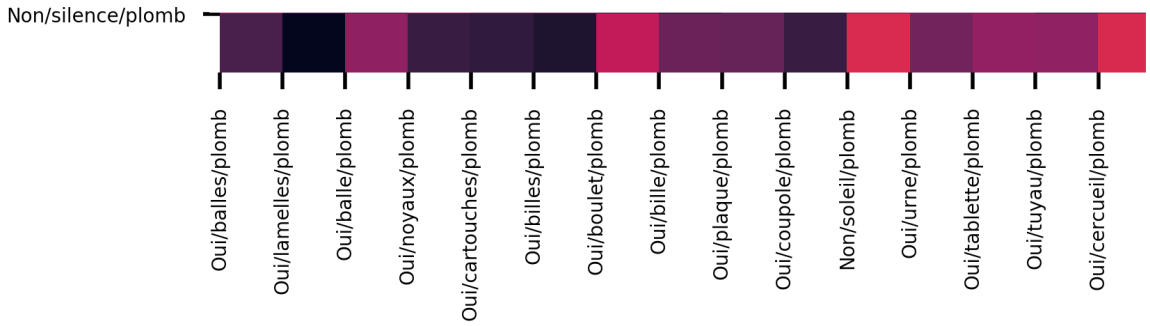


Illustration 10 : heatmap de silence/soleil de plomb (base-collocatif).

Cette *heatmap* montre la similarité entre les collocations/cooccurrences issues de « silence de plomb » et celles issues de « soleil de plomb ». Nous pouvons remarquer que les exemples les plus similaires à « silence de plomb » sont « boulet de plomb », « soleil de plomb » ainsi que « cercueil de plomb ». En effet, une couleur claire indique une plus grande similarité qu'une couleur foncée (cf. Heatmap, page 37).

4.1.1.2 - Analyse collocatif seul

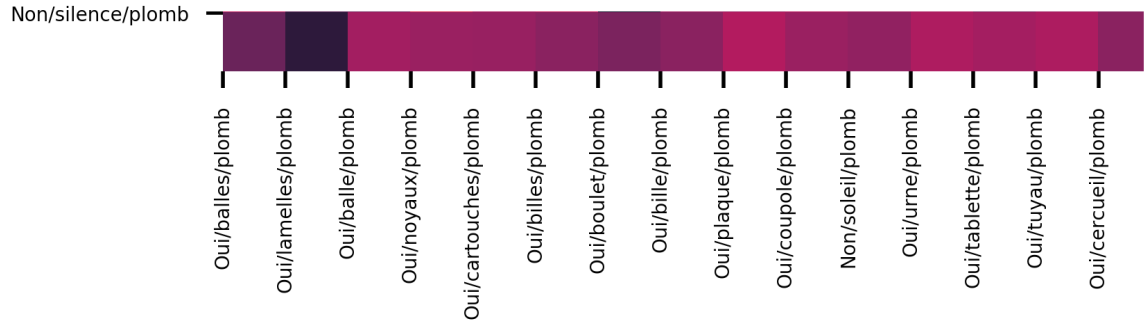


Illustration 11 : *heatmap* de silence de plomb vs soleil de plomb (collocatif seul).

Contrairement à la précédente *heatmap*, celle-ci ne s'intéresse qu'à la similarité du collocatif/cooccurrent *plomb* entre les exemples issus de « soleil de plomb » et ceux issus de « silence de plomb ». Nous pouvons remarquer que *plomb* de « silence de plomb » n'est pas plus similaire à *plomb* de « soleil de plomb » qu'il ne l'est des autres. En effet, toutes les

cases restent à peu près de la même teinte, sauf *plomb* de « lamelles de plomb » qui est plus foncée. Il n'y a donc pas d'exemples très similaires, car ils sont tous de couleur assez foncée.

4.1.2 - Dendrogramme

Voici maintenant mon analyse des dendrogrammes. Pour voir les figures en entier cf. pages 68 et 69.

4.1.2.1 - Analyse base-collocatif

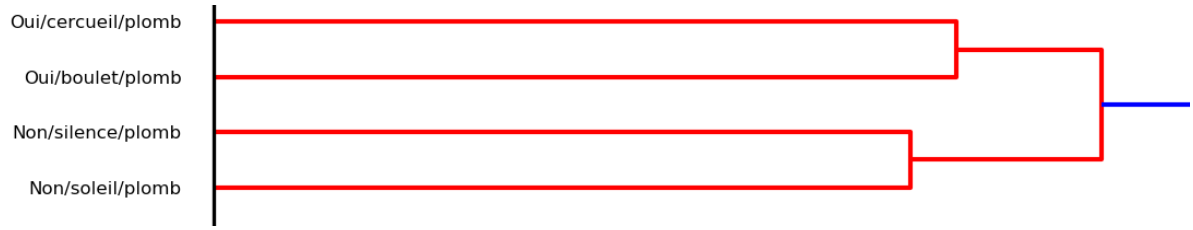


Illustration 12 : dendrogramme silence de plomb et soleil de plomb (base-collocatif).

Ce dendrogramme s'intéresse à la similarité entre « soleil de plomb » et « silence de plomb ». Nous pouvons remarquer qu'il y a un regroupement composé des collocations « silence de plomb » et « soleil de plomb » ainsi que les cooccurrences « cercueil de plomb » et « boulet de plomb » avec lesquelles les collocations s'agglomèrent plus loin. Cela indique qu'il existe une certaine similarité entre ces 4 exemples, mais « silence de plomb » et « soleil de plomb » sont plus similaires entre eux que des 2 cooccurrences.

4.1.2.2 - Analyse collocatif seul

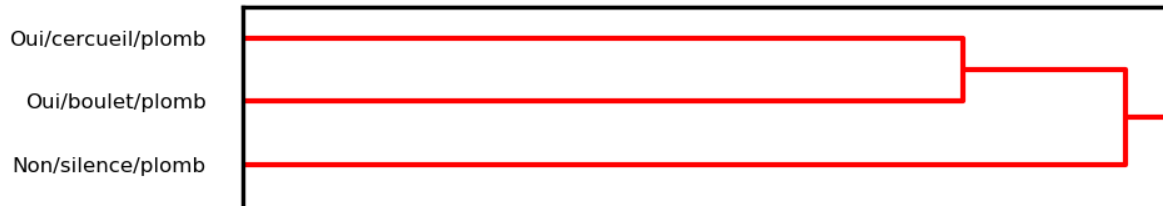


Illustration 13 : dendrogramme silence de plomb et soleil de plomb (collocatif seul).

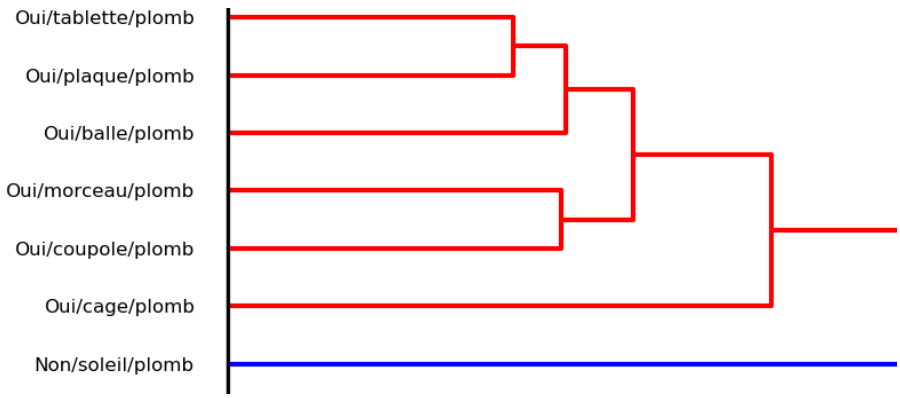


Illustration 14 : dendrogramme silence de plomb et soleil de plomb (collocatif seul)

(2).

Ces deux figures représentent le même dendrogramme à deux endroits différents. Nous pouvons remarquer qu'il y a un regroupement composé de *plomb* de la collocation « silence de plomb » et *plomb* des cooccurrences « boulet de plomb » et « cercueil de plomb » (Illustration 13). Par ailleurs, la branche reliant « Soleil de plomb » au reste des exemples en rouges apparaît beaucoup plus haut dans le dendrogramme. Seul le collocatif de « soleil de plomb » reste à part. Cela indique que le collocatif de « silence de plomb » est plus similaire aux cooccurrents qu'au collocatif de « soleil de plomb » (cf. page 69 pour le dendrogramme entier).

4.1.3 - Analyse globale

Je vais maintenant poursuivre avec mon interprétation des résultats obtenus (illustrations 10 à 14).

Les résultats ne vont pas dans le sens de mon hypothèse. Celle-ci était que les deux collocations seraient plus similaires entre elles qu'elles ne le sont des cooccurrences libres et de même pour le collocatif seul par rapport au cooccurrent seul (cf. 3.1.5.1 - Hypothèse principale, page 24).

Les deux collocations sont certes un peu plus similaires entre elles qu'elles ne le sont des autres exemples (illustrations 10 et 12) mais elles ne forment pas un groupe à part et restent de la même couleur. Une explication possible des résultats obtenus en illustration 12 est qu'étant donné que le collocatif est le même, les résultats sont tous à peu près similaires (cf. page 66).

Par ailleurs, pouvons remarquer que pour le cas des collocatifs seuls, que le groupe qui était formé par les deux collocations n'existe plus et qu'au contraire, *plomb* de la collocation « soleil de plomb » n'est pas du tout similaire au reste des exemples car il est représenté d'une autre couleur (illustration 14).

Pour rappel, dans leur mode de fonctionnement, les scripts de dendrogrammes et *heatmaps* sélectionnent aléatoirement un exemple de « soleil de plomb » et de « silence de plomb ». Il ne s'agit donc pas nécessairement des deux mêmes exemples entre chaque figures.

Cependant, je cherchais à évaluer si FlauBERT arrivait à distinguer les collocations des cooccurrences, donc si tel était le cas, le choix des deux collocations n'aurait pas d'incidence majeur sur les résultats obtenus.

4.2 - Comparaison soleil de plomb et colère saine

4.2.1 - *Heatmap*

Je vais maintenant poursuivre mon analyse en m'intéressant aux *heatmaps* issues de « soleil de plomb » et « colère saine » (cf. page 25). Les illustrations ne sont que des aperçus. Pour voir les *heatmaps* entièrement cf. pages 70 et 71.

4.2.1.1 - Analyse base-collocatif

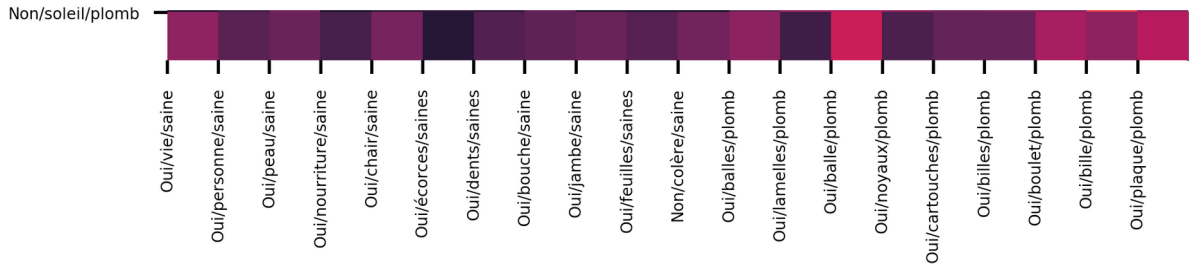


Illustration 15 : *heatmap* soleil de plomb/colère saine (base-collocatif).

Nous pouvons remarquer que « soleil de plomb » et « silence de plomb » ne sont pas les plus similaires. En effet, l'exemple le plus similaire à « soleil de plomb » est la cooccurrence « balle de plomb ». De plus, dans une moindre mesure, « bille de plomb » et « coupole de plomb » sont quand même plus similaire à « soleil de plomb » que « colère saine » ne l'est. Cela signifie qu'il n'y a pas de réelle similarité entre les deux collocations.

4.2.1.2 - Analyse collocatif seul

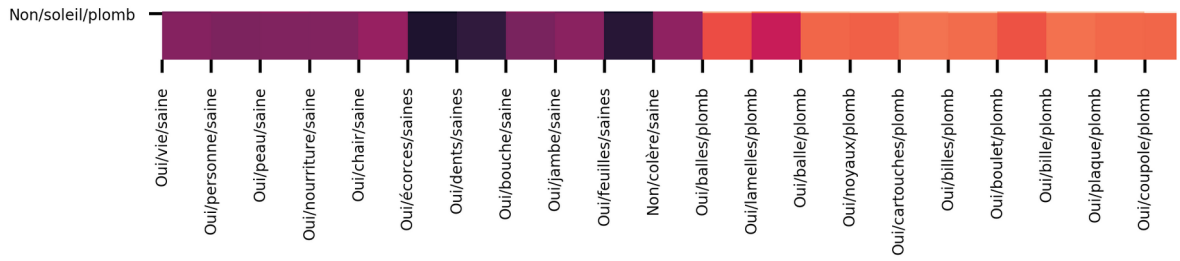


Illustration 16 : *heatmap* soleil de plomb/colère saine (collocatif seul).

Cette fois-ci, nous voyons bien qu'il y a deux groupes distincts. À gauche, le groupe des exemples issus de « colère saine » et à droite les exemples issus de « soleil de plomb ». Le groupe en orange correspond aux collocatifs *plomb*, tandis que le groupe en violet correspond aux exemples issus de « colère saine ». Cela indique que les deux collocatifs ne sont pas pas plus similaires que leur exemple miroir, c'est-à-dire les exemples issus du même nom de collocatif.

4.2.2 - Dendrogrammes

Je vais maintenant poursuivre sur les dendrogrammes que j'ai obtenus à partir du corpus comparant « soleil de plomb » et « colère saine » (cf. page 72 pour le dendrogramme en illustration 17 et page 73 pour le dendrogramme en illustrations 18 et 19).

4.2.2.1 - Analyse base-collocatif

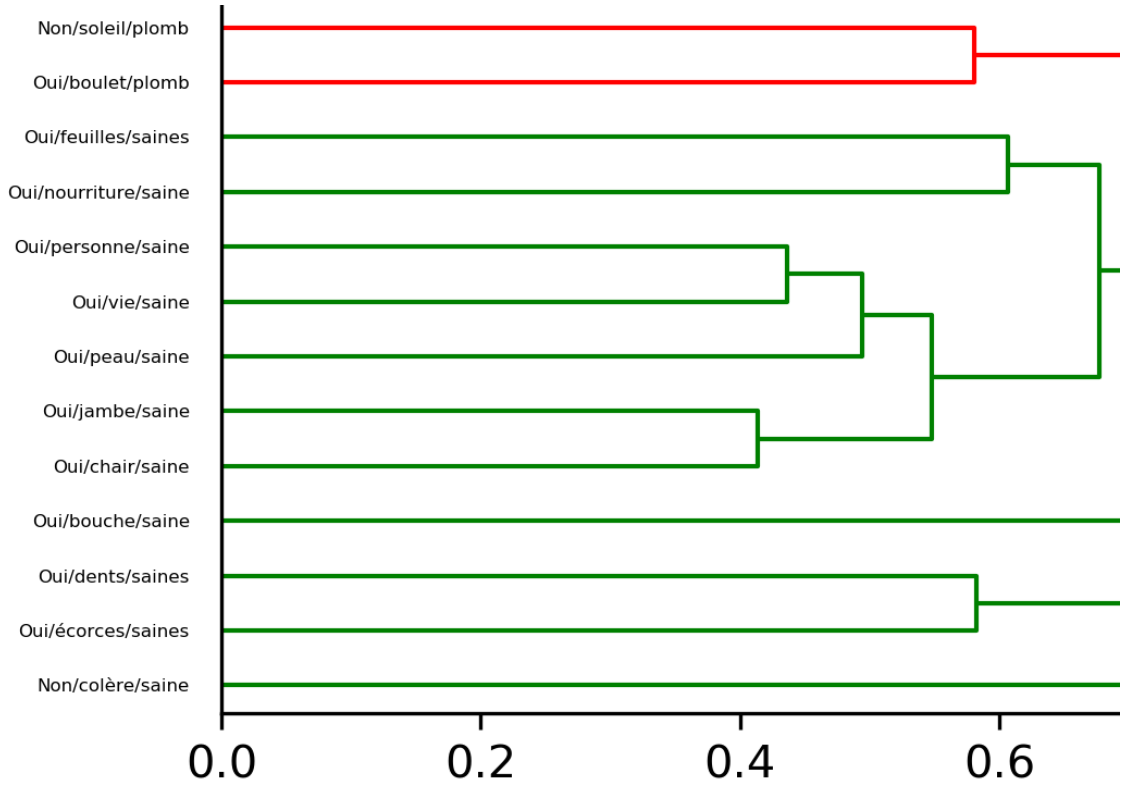


Illustration 17 : dendrogramme soleil de plomb/colère saine (base-collocatif).

Nous pouvons remarquer que dans ce dendrogramme, il y a un groupe composé d'une cooccurrence et de la collocation du corpus « soleil de plomb » (en rouge) et un second groupe composé des occurrences et de la collocation de « colère saine ». Les deux collocations sont de deux couleurs différentes et se rejoignent qu'à la toute fin de l'arbre (non visible sur l'illustration 17). Cela signifie que les exemples de même collocation (*saine* et *plomb*) sont regroupés entre eux, au lieu d'être regroupés par type (cooccurrence ou collocation).

4.2.2.2 - Analyse collocatif seul

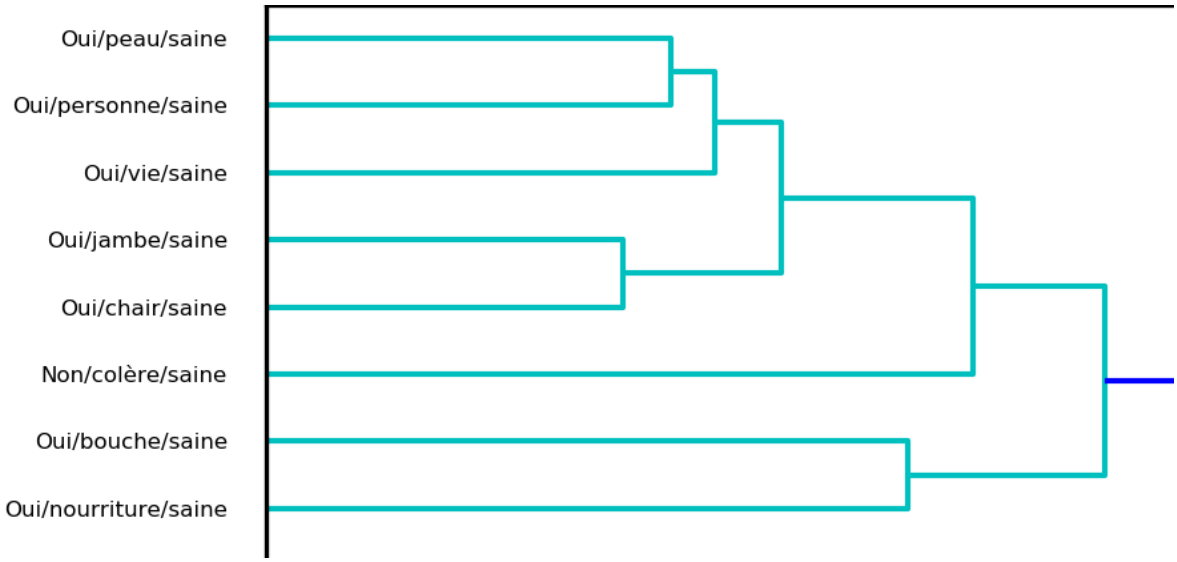


Illustration 18 : dendrogramme soleil de plomb/colère saine (collocatif seul)(1).

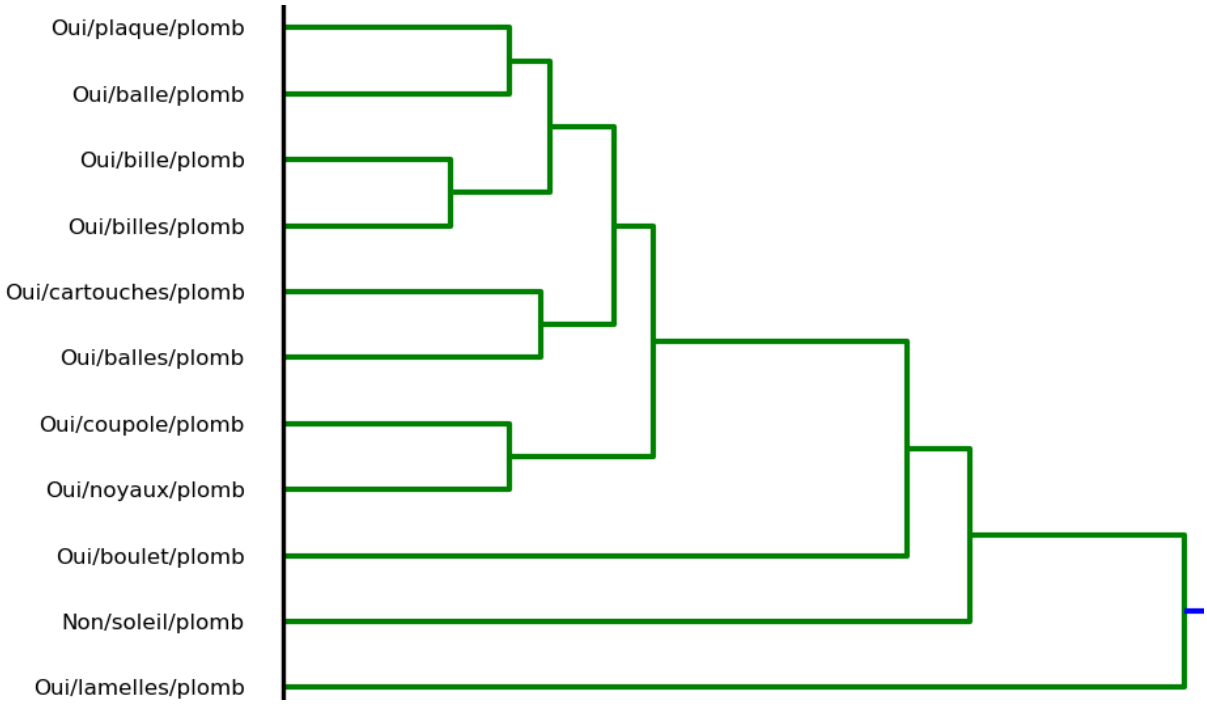


Illustration 19 : dendrogramme soleil de plomb/colère saine (collocatif seul)(2).

Comme pour la *heatmap* (illustration 16), le dendrogramme est découpé en deux parties distinctes. Chacune représente un collocatif différent. En bleu, les exemples de « colère saine » et en vert ceux de « soleil de plomb ». Les branches se regroupent aussi à la toute fin de l'arbre (non visible sur l'illustration). Il s'agit aussi d'un cas où les exemples sont agglomérés selon le nom du collocatif au lieu d'être agglomérés par type de combinaison de mots comme j'avais pu l'espérer (collocation ou cooccurrence).

4.2.3 - Analyse globale

Cette analyse globale concernera les résultats obtenus pour le corpus « colère saine et silence de plomb », à partir des illustrations 15 à 19.

Il s'agit d'exemples issus de deux fonctions lexicales différentes (cf. page 25). « Colère saine » correspond à la fonction lexicale Bon et « silence de plomb » à Magn.

Je commencerai d'abord par une analyse globale des collocations/cooccurrences, suivi d'une analyse des résultats des collocatifs/cooccurrents.

Les *heatmaps* ne montrent rien de concluant, c'est-à-dire qu'on ne voit pas une nette différence entre les collocations et les cooccurrences (illustrations 15 et 16). De plus, que ce soit le dendrogramme ou la *heatmap*, les deux collocations ne sont pas du tout similaires entre elles. En effet, elles ne se rejoignent qu'à la toute fin du dendrogramme, donc la collocation « soleil de plomb » est moins similaire à « colère saine » qu'aux cooccurrences de « colère saine ».

Par ailleurs, nous pouvons remarquer que lorsqu'il s'agit du collocatif/cooccurrent seul, il y a vraiment une très nette distinction entre les exemples de « colère saine » et ceux de « soleil de plomb » (cf. pages 71 et 73). Cela va dans le sens de l'hypothèse distributionnelle, c'est-à-dire que des mots utilisés dans des contextes similaires ont tendance à avoir des vecteurs similaires (cf. page 16), car tous les collocatifs de même nom sont regroupés ensemble.

De manière générale, les collocatifs différents (*plomb* et *saine*) ne montrent aucune affinité.

4.3 - Comparaison soleil de plomb et dégoût profond

4.3.1 - *Heatmap*

Dans cette partie je vais m'intéresser aux *heatmaps* issues de « soleil de plomb » et « dégoût profond » (cf. page 25). Les illustrations ne sont une fois de plus que des aperçus. Pour voir les *heatmaps* entièrement, cf. pages 74 et 75.

4.3.1.1 - Analyse base-collocatif

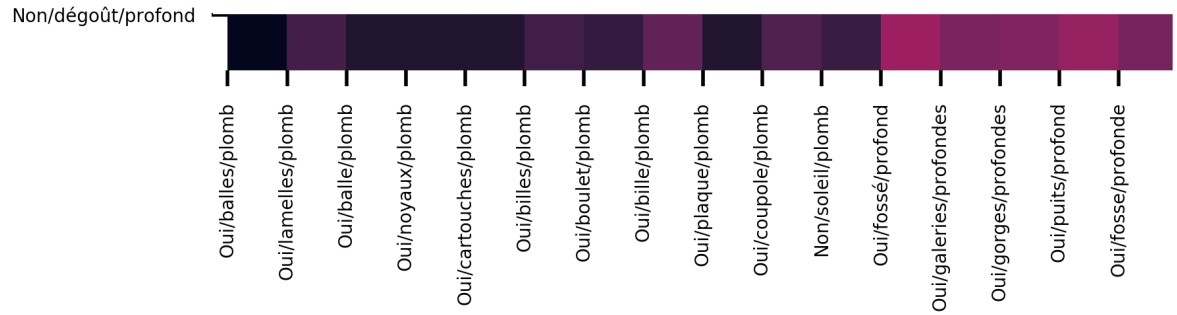


Illustration 20 : *heatmap* soleil de plomb/dégoût profond (base-collocatif).

Il n'y a pas de forte similarité entre « dégoût profond » et le reste des exemples. On peut remarquer cependant une légère différence de couleur entre les exemples de « soleil de plomb » et ceux de « dégoût profond ». En effet, « dégoût profond » est plus similaire aux cooccurrences du même type que lui que de la collocation « soleil de plomb ».

4.3.1.2 - Analyse collocatif seul

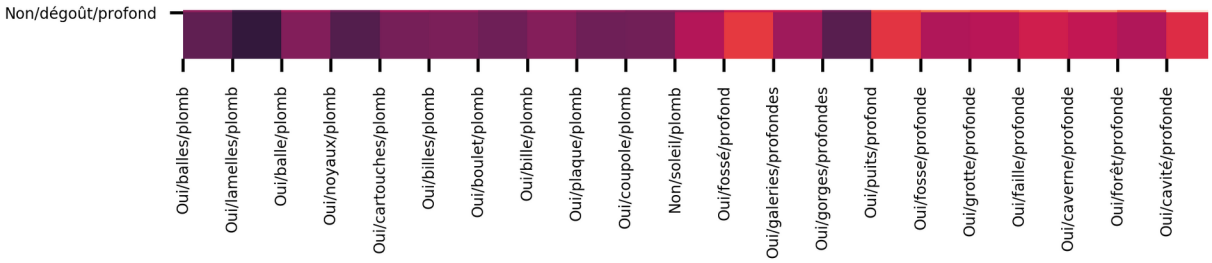


Illustration 21 : *heatmap* soleil de plomb/dégoût profond (collocatif seul).

Il y a deux groupes distincts représentant les deux collocations/cooccurrences. En effet, le collocatif *dégoût* de « dégoût profond » est plus similaire aux cooccurents *profond*, que des cooccurrents et la cooccurrence de « soleil de plomb ». On remarque cela car la teinte orange des cases indique une similarité plus élevée que les cases violettes.

4.3.2 - Dendrogrammes

Je vais maintenant m'intéresser aux dendrogrammes représentant les similarités entre « soleil de plomb » et « dégoût profond ». Le dendrogramme en illustrations 22 et 23 se trouve en annexe 15 (page 76) et le dendrogramme en illustration 24 se trouve en annexe 16 (page 77).

4.3.2.1 - Analyse base-collocatif

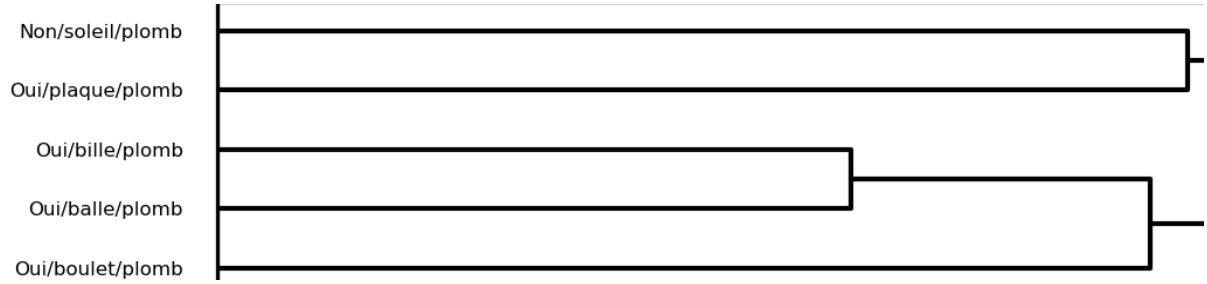


Illustration 22 : dendrogramme soleil de plomb/dégoût profond (base-collocatif)(1).

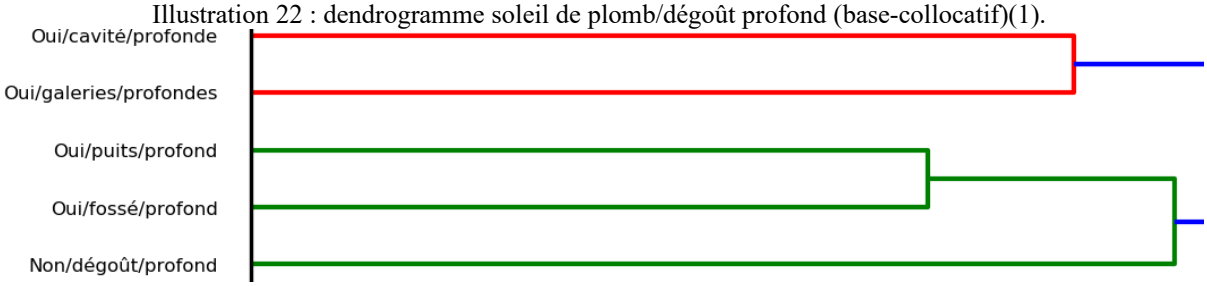


Illustration 23 : dendrogramme soleil de plomb/dégoût profond (base-collocatif)(2).

On remarque qu'il n'y a pas d'affinité entre les 2 collocations. En effet, plusieurs exemples issus de « soleil de plomb » sont agglomérés ensemble (en noir). Un groupe de 3 exemples est constitué de deux cooccurrences et de la collocation de « dégoût profond » (en vert). Ils rejoignent le groupe en noir qu'à la fin du dendrogramme. Ce qui indique que la similarité à laquelle je m'attendais entre les deux collocations n'existe pas.

4.3.2.2 - Analyse collocatif seul

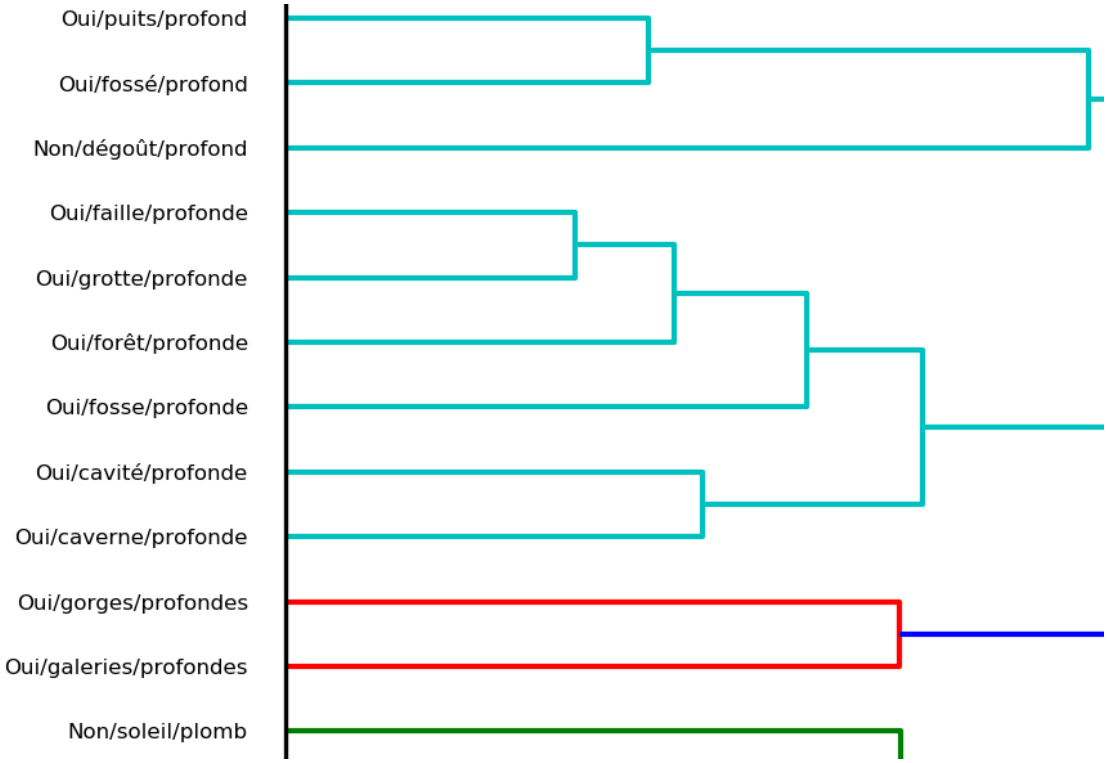


Illustration 24 : dendrogramme soleil de plomb/dégoût profond (collocatif seul).

Pour finir cette analyse des résultats, on peut remarquer que le collocatif de « soleil de plomb » et celui de « dégoût profond » ne sont pas non plus regroupés ensemble. Ils sont affichés chacun d'une couleur différente (vert et bleu). De plus, les groupes verts et bleus ne se rejoignent qu'à la toute fin du dendrogramme (non visible sur les illustrations). Cela indique que même pour le collocatif seul, il n'y a pas une grande similarité entre les deux collocatifs.

4.3.3 - Analyse globale

Voici ma dernière analyse et interprétation des résultats concernant les deux *heatmaps* et dendrogrammes de « soleil de plomb vs dégoût profond ».

Le but de ce corpus était de savoir si au sein d'une même fonction lexicale, les collocations étaient similaires entre elles, même si la base et le collocatif étaient différents. Les résultats ne sont pas concluants et vont dans le même sens que les précédents résultats (cf. page 49). Il n'y a pas un groupe composé des collocations et un autre composé des cooccurrences. Plusieurs éléments comme le genre et le nombre affectent les résultats. Cela est plus flagrant sur l'illustration 24. Nous pouvons voir que même au sein de cooccurrences il y a plusieurs groupes. Par exemple *profondes* de« gorges profondes » et *galeries* sont au

féminin pluriel et forment un groupe à part, tandis qu'un autre groupe est constitué de *profonde* au singulier, et un 3e groupe est constitué que de *profond* au masculin.

4.4 - Réseau de neurones

Pour cette partie, je vais procéder d'une manière différente. J'exposerai les résultats obtenus en illustrations 25 et 26, ci dessous. Il s'agit de tableaux récapitulatifs de toutes les expérimentations que j'ai effectuées avec le classifieur automatique. Chaque pourcentage correspond à l'exactitude obtenue par le classifieur (cf. page 41).

Pour rappel, le classifieur a été entraîné sur des vecteurs de FlauBERT ainsi que des vecteurs de fastText. J'ai en parallèle défini un système *baseline* (cf. page 40). Le classifieur a été entraîné sur les corpus Bon, Magn et « Bon et Magn classé ». Plusieurs résultats ont été possibles pour un même entraînement.. En effet, j'ai une première fois entraîné le classifieur sur des vecteurs issus de la soustraction d'une base et d'un collocatif, puis une seconde fois sur les vecteurs de collocatifs seuls.

4.4.1 - Analyse FlauBERT et *baseline*

Base-collocatif			
	Bon	Magn	Bon+Magn
FlauBERT	62.0%	78.0%	72.0%
Baseline	56.0%	62.0%	59.0%
Collocatif seul			
FlauBERT	66.5%	85.5%	83.0%
Baseline	56.0%	62.0%	59.0%

Illustration 25 : comparaison FlauBERT et *baseline.*

Nous pouvons remarquer en illustration 25 que de manière générale FlauBERT obtient une exactitude plus élevée que le système *baseline* pour chacun des corpus. L'exactitude du classifieur augmente lorsqu'on l'entraîne sur un vecteur de collocatif seul, de l'ordre de 5 à 10 % environ. La précision du systeme *baseline* reste la même dans les deux cas (base-collocatif et collocatif seul) car il n'y a pas de modification de son fonctionnement entre les deux tests (cf. page 40).

4.4.2 - Analyse FlauBERT et fastText (concaténation)

Dans cette dernière partie, je comparerai la précision obtenue par le classifieur sur des vecteurs concaténés de FlauBERT et de fastText. Cela permet de déterminer si un classifieur entraîné sur FlauBERT repère mieux la différence entre collocations et cooccurrences qu'un classifieur entraîné sur fastText.

Concaténation			
	Bon	Magn	Bon+Magn
FlauBERT	61.5%	86.0%	82.75%
FastText	58.5%	82.5%	72.5%

Illustration 26 : Comparaison FlauBERT et fastText.

Nous pouvons nous rendre compte que la précision du classifieur entraîné sur FlauBERT, obtient des résultats meilleurs que sur fastText pour chacun des 3 corpus. Le corpus « Bon + Magn » étant plus gros que les deux autres (400 contre 200) permet d'obtenir des résultats bien meilleurs. En effet, il y a en moyenne 3 % d'écart entre le classifieur entraîné sur FlauBERT et celui entraîné sur fastText. Cet écart est agrandi lorsqu'il s'agit du gros corpus, passant alors à 10 % d'écart.

4.4.3 - Analyse et interprétation générale

Les résultats du classifieur sont plus élevés lorsqu'il a été entraîné sur FlauBERT, peu importe le cas. (cf. illustrations 25 et 26). Le classifieur automatique obtient une meilleure précision lors de la reconnaissance entre une collocation et une occurrence. Cela laisse penser que les vecteurs de FlauBERT permettent de mieux appréhender les fonctions lexicales.

Conclusion

En conclusion, ce travail a bien permis de répondre à la problématique ainsi qu'aux différentes hypothèses. Cette problématique était de savoir si FlauBERT était capable d'encoder dans ses vecteurs de mots la différence entre collocation et cooccurrence.

Cependant, les résultats obtenus avec les *heatmaps* et les dendrogrammes ne sont pas donc pas allés dans le sens de l'hypothèse de départ.

Par ailleurs, les résultats du classifieur montrent des résultats positifs. En effet les résultats sur les vecteurs de FlauBERT sont supérieurs à ceux obtenus sur les vecteurs de fastText ou encore un système *baseline*. Cela indique qu'il y a certaines informations contenues dans les vecteurs contextuels de FlauBERT permettant de mieux cerner la différence entre une collocation et une cooccurrence.

De plus, il existe quelques pistes d'amélioration qui pourraient permettre d'améliorer les résultats. Par exemple, il serait possible de constituer un corpus avec un seul genre et un seul nombre (par exemple masculin singulier). Cela permettrait d'éviter les cas où plusieurs groupes de vecteurs se forment, même au sein d'un même groupe de 20 exemples. Cela est notamment visible sur l'illustration 24 et son analyse. Cependant, la constitution du corpus serait beaucoup plus longue.

Une autre méthode pourrait être de faire une moyenne pour chaque collocation, c'est-à-dire faire la moyenne des vecteurs des 10 collocations de « soleil de plomb » au lieu d'en choisir qu'une seule par exemple. Cela permettrait tout d'abord d'avoir des figures moins volumineuses et donc plus faciles à analyser. De plus, étant donné que les vecteurs de FlauBERT sont définis par la phrase entière, cela permettrait d'éviter des cas où le contexte des deux phrases serait similaire.

Pour finir, ce stage m'a également beaucoup apporté personnellement. J'ai tout d'abord appris à faire preuve de méthode, notamment concernant la classification des fichiers et la structuration des scripts. Un point fort de ce stage est aussi que j'ai pu approfondir mes connaissances en TAL, notamment l'entraînement d'un classifieur automatique, l'utilisation de PyTorch afin de manipuler des vecteurs ou encore la création de *heatmaps* et dendrogrammes que je n'avais jamais vu ou expérimenté en classe. Je ne connaissais pas non plus l'existence des modèles de langages possédant des vecteurs contextuels comme FlauBERT. Ces modèles sont assez récents et cela a été pour moi l'occasion d'en apprendre

plus sur le sujet. Pour finir, je peux affirmer que j'ai progressé en programmation durant mon stage. Ce stage aura en résumé été plus que bénéfique pour moi.

Bibliographie - Sitographie

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135-146.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

Harris, Z. S. (1954). Distributional Structure. WORD, *10*(2-3), 146-162. https://doi.org/10.1080/00437956.1954.11659520

Le, H., Vial, L., Frej, J., Segonne, V., Coavoux, M., Lecouteux, B., Allauzen, A., Crabbé, B., Besacier, L., & Schwab, D. (2019). Flaubert : Unsupervised language model pre-training for french. *arXiv preprint arXiv:1912.05372*.

Le, H., Vial, L., Frej, J., Segonne, V., Coavoux, M., Lecouteux, B., Allauzen, A., Crabbé, B., Besacier, L., & Schwab, D. (2020). FlauBERT : Des modèles de langue contextualisés pré-entraînés pour le français (FlauBERT : Unsupervised Language Model Pre-training for French). *Actes de la 6e conférence conjointe Journées d'Études sur la Parole (JEP, 33e édition), Traitement Automatique des Langues Naturelles (TALN, 27e édition), Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RÉCITAL, 22e édition). Volume 2 : Traitement Automatique des Langues Naturelles*, 268–278. https://www.aclweb.org/anthology/2020.jeptalnrecital-taln.26

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

peur bleue—Wiktionnaire. (s. d.). Consulté 8 septembre 2020, à l'adresse https://fr.wiktionary.org/wiki/peur_bleue

Polguère, A. (2002). *Modélisation des liens lexicaux au moyen des fonctions lexicales*. 37-60.

Systèmes Lexicaux – Alain Polguère. (s. d.). Consulté 9 septembre 2020, à l'adresse https://perso.atilf.fr/apolguere/fr/lexical-systems/

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need*. 5998-6008.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). Glue : A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020). HuggingFace's Transformers : State-of-the-art Natural Language Processing. *arXiv:1910.03771 [cs]*. http://arxiv.org/abs/1910.03771

Sigles et abréviations utilisés

Magn : Fonction lexicale *Magn*

Bon : Fonction lexicale *Bon*

GLUE : General Language Understanding Evaluation

FLUE : French Language Understanding Evaluation

BERT : Bidirectional Encoder Representations from Transformers

FlauBERT : Modèle de langage fonctionnant avec la bibliothèque *transformers,* similaire à BERT

Table des illustrations

Table des annexes

Annexe 1
Téléchargement du projet

Cette annexe permet d'accéder au serveur hébergeant la totalité de mes fichiers. Tous les dossiers sont classés de manière méthodique. De plus, chaque nom de fichier a été choisi de manière à posséder le plus d'informations possibles sur l'input utilisé et l'output obtenu.

Le lien pour accéder à mon projet est le suivant :

https://github.com/VincentBellue/stage_m2_bellue_vincent

Cependant, le fichier de vecteurs de fastText n'est pas présent sur le serveur en raison de sa taille volumineuse. Toutefois, il est possible de le télécharger à l'adresse suivante : https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.fr.300.vec.gz

Une fois téléchargé, il faut le placer au même endroit que le script qui utilise les vecteurs de fastText. Le chemin d'accès est le suivant :
« scripts/vecteur_base_collocatif/fasttext/script_base_colloc_vec_statique.py ».

Mon projet comprend un fichier récapitulatif appelé « README.MD ». Il permet de visualiser avec plus de facilité toutes mes expérimentations ainsi que leurs résultats.

Ce fichier possède plusieurs tableaux et listes récapitulatives des différents scripts et résultats obtenus. Lorsque les mots sont écrits en bleu, cela indique qu'il s'agit d'un hyperlien sur lequel on peut cliquer afin d'accéder directement au fichier en question. Voici un aperçu des parties du fichier « README.MD » :

- **Partie fichiers .tsv :** Il s'agit d'un tableau avec des liens cliquables comme l'emplacement des scripts, un exemple de fichier utilisé en entrée de ce script, ainsi que le fichier obtenu en sortie de ce script.

- **Partie Entraînement et test d'un réseau de neurones à une couche cachée** : Il s'agit des fichiers obtenus lors de l'entraînement de mon classifieur sur FlauBERT et fastText. Cela permet de voir les détails de chaque entraînement.

- **Partie Résultats et accuracy du réseau de neurones** : Il s'agit d'un tableau récapitulatif des résultats obtenus durant l'entraînement du classifieur sur FlauBERT et fastText. Chaque lien est cliquable afin de mener directement aux fichiers de résultats.

- **Partie Résultats *heatmap* et dendrogramme (figures)** : Il s'agit d'une liste récapitulative de chaque figure obtenue lors des différentes expérimentations.

Il n'est pas nécessaire de naviguer à l'intérieur des dossiers car le fichier README donne directement les liens vers les différents fichiers, mais voici la liste des différents dossier du projet :

- **ancienne version** : Il s'agit d'anciens fichiers que j'ai préféré garder.

- **inputs** : Il s'agit du dossier contenant les fichiers que j'ai constitué à la main.

- **outputs** : Il s'agit du dossier contenant les fichiers que j'ai obtenus en sortis de tous mes scripts. Ils sont classés par rapport au script duquel ils sont issus.

- **figures** : Il s'agit du dossier contenant toutes les figures obtenues lors de mon stage.

- **scripts** : Il s'agit du dossier contenant la totalité de mes scripts. Il y a à la racine un script nommé « fonctions.py » qui regroupe toutes mes fonctions utilisées. Les sous-dossiers correspondent aux différents scripts développés durant le stage. Tous les scripts sont commentés de manière à comprendre les boucles et actions effectuées par mes fonctions. Ces commentaires sont destinés à des développeurs uniquement.

- **divers** : Il s'agit du dossier contenant le fichier « FL_extract.tsv ».

Annexe 2

Exemple de fichier constitué à la main

Base	Genre-base	Nombre-base	Collocatif	Genre-collocatif	Nombre-collocatif	Lemme collocatif	Compositionnel	Phrase
colère	féminin	singulier	saine	féminin	singulier	sain	Non	Il aurait tant aimé que les deux fouettent enfin la terre d'une saine colère et redonnent aux
colère	féminin	singulier	saine	féminin	singulier	sain	Non	Après, au sortir de la projection, le sentiment oscille entre gêne poisseuse et saine colère
colère	féminin	singulier	saine	féminin	singulier	sain	Non	Comme elle ne s'est pas mise en colère depuis très longtemps, n'en ayant plus guère l'oc
colère	féminin	singulier	saine	féminin	singulier	sain	Non	Cette productivité qui étonne, inquiète parfois même les proches, trouve sa source dans u
colère	féminin	singulier	saine	féminin	singulier	sain	Non	Pauline le sentait soulevé de colère, mais une saine colère.
colère	féminin	singulier	saine	féminin	singulier	sain	Non	Les cinquante récits que je dus subir de leur promenade en ville, de cette exhibition honte
colère	féminin	singulier	saine	féminin	singulier	sain	Non	Il serait venu pour faire un scandale, une sorte de provocation entre le politique et la poési
colère	féminin	singulier	saine	féminin	singulier	sain	Non	Elle escomptait qu'il suffirait de dévoiler les faits pour que le monde entier soit secoué par
colères	féminin	pluriel	saines	féminin	pluriel	sain	Non	Les vrais dandys Leroy en profite pour nous livrer quelques saines colères contre les " cyl
colères	féminin	pluriel	saines	féminin	pluriel	sain	Non	Au début de son premier mandat, Boris Eltsine plaisait pour son côté moujik doté d'une bc
vie	féminin	singulier	saine	féminin	singulier	sain	Oui	Faut que tu dormes plus, il lui disait, faut que tu te reposes, faut que t'arrêtes de sortir com
personne	féminin	singulier	saine	féminin	singulier	sain	Oui	Une personne saine et enjouée, me dis-je, n'aurait pas fait ce que je faisais.
peau	féminin	singulier	saine	féminin	singulier	sain	Oui	La louve s'élança aussitôt vers le petit et entreprit de le nettoyer à grands coups de langue
nourriture	féminin	singulier	saine	féminin	singulier	sain	Oui	Une nourriture saine et complète lui suffit.
chair	féminin	singulier	saine	féminin	singulier	sain	Oui	Et puis, cette propreté, ce visage massif et diaphane, cette foulée aérienne, et cette odeur
écorces	féminin	pluriel	saines	féminin	pluriel	sain	Oui	Du revers de ma lame j'écartais une feuille quand un jeune arbre aux écorces saines et lu
dents	féminin	pluriel	saines	féminin	pluriel	sain	Oui	Du coin de l'oeil, je l'examine : le teint rose, la peau bien tendue sur les os qu'on devine à
bouche	féminin	singulier	saine	féminin	singulier	sain	Oui	– Au moins, toi, dit-elle en me souriant de sa large bouche saine et éclatante, tu es l'homm
jambe	féminin	singulier	saine	féminin	singulier	sain	Oui	Sa jambe saine, sur laquelle il avait transféré tout son poids, tremblait de fatigue.
feuilles	féminin	pluriel	saines	féminin	pluriel	sain	Oui	À bout de souffle, il s'appuya contre un saule et nota qu'on aurait dû élaguer l'arbre au prir

Annexe 3

Exemple de fichier de sortie (FlauBERT)

Ligne	Compositionnel	Base	Genre base	Vecteur base	Collocatif	Vecteur collocatif	Phrase
2	Non	colère	féminin	-0.69396853	saine	2.2910316 -0.7788	Il aurait tant aimé que les deux fouettent enfin la terre d'une saine colère et redon
3	Non	colère	féminin	-0.2692821 -?	saine	-0.017637461 -0.4	Après, au sortir de la projection, le sentiment oscille entre gêne poisseuse et sain
4	Non	colère	féminin	-0.7639316 -?	saine	0.4600613 -0.6413	Comme elle ne s'est pas mise en colère depuis très longtemps, n'en ayant plus g
5	Non	colère	féminin	-0.90649956	saine	0.71446526 -0.537	Cette productivité qui étonne, inquiète parfois même les proches, trouve sa sourc
6	Non	colère	féminin	0.22381037 0	saine	0.85836005 0.607	Pauline le sentait soulevé de colère, mais une saine colère.
7	Non	colère	féminin	0.21466786 -	saine	1.6083424 0.0739	Les cinquante récits que je dus subir de leur promenade en ville, de cette exhibiti
8	Non	colère	féminin	0.60392195 -	saine	1.9152865 -1.3887	Il serait venu pour faire un scandale, une sorte de provocation entre le politique e
9	Non	colère	féminin	-0.11317119	saine	1.3726021 -0.3581	Elle escomptait qu'il suffirait de dévoiler les faits pour que le monde entier soit sec
10	Non	colères	féminin	0.56659085 -	saines	1.0025258 0.1677	Les vrais dandys Leroy en profite pour nous livrer quelques saines colères contre
11	Non	colères	féminin	0.30813217 0	saines	2.757267 2.67573	Au début de son premier mandat, Boris Eltsine plaisait pour son côté moujik doté
12	Oui	vie	féminin	-0.7246965 -0	saine	2.2067962 0.3835	Faut que tu dormes plus, il lui disait, faut que tu te reposes, faut que t'arrêtes de s
13	Oui	personne	féminin	1.5140437 -1	saine	0.29074514 -0.50	Une personne saine et enjouée, me dis-je, n'aurait pas fait ce que je faisais.
14	Oui	peau	féminin	-0.09012088	saine	0.13255134 0.909	La louve s'élança aussitôt vers le petit et entreprit de le nettoyer à grands coups
15	Oui	nourriture	féminin	1.2324705 -1	saine	-0.10511114 -1.59	Une nourriture saine et complète lui suffit.
16	Oui	chair	féminin	0.15543789 0	saine	2.1481464 -0.8288	Et puis, cette propreté, ce visage massif et diaphane, cette foulée aérienne, et ce
17	Oui	écorces	féminin	1.684897 3.6	saines	2.158564 -0.9633	Du revers de ma lame j'écartais une feuille quand un jeune arbre aux écorces sa
18	Oui	dents	féminin	0.61691916 -	saines	0.961099 0.25038	Du coin de l'oeil, je l'examine : le teint rose, la peau bien tendue sur les os qu'on
19	Oui	bouche	féminin	-0.6541115 -0	saine	0.36783206 -0.860	– Au moins, toi, dit-elle en me souriant de sa large bouche saine et éclatante, tu e
20	Oui	jambe	féminin	0.3497222 1.	saine	1.984436 0.64768	Sa jambe saine, sur laquelle il avait transféré tout son poids, tremblait de fatigue.
21	Oui	feuilles	féminin	0.08979598 0	saines	2.5150597 -2.633	À bout de souffle, il s'appuya contre un saule et nota qu'on aurait dû élaguer l'arb

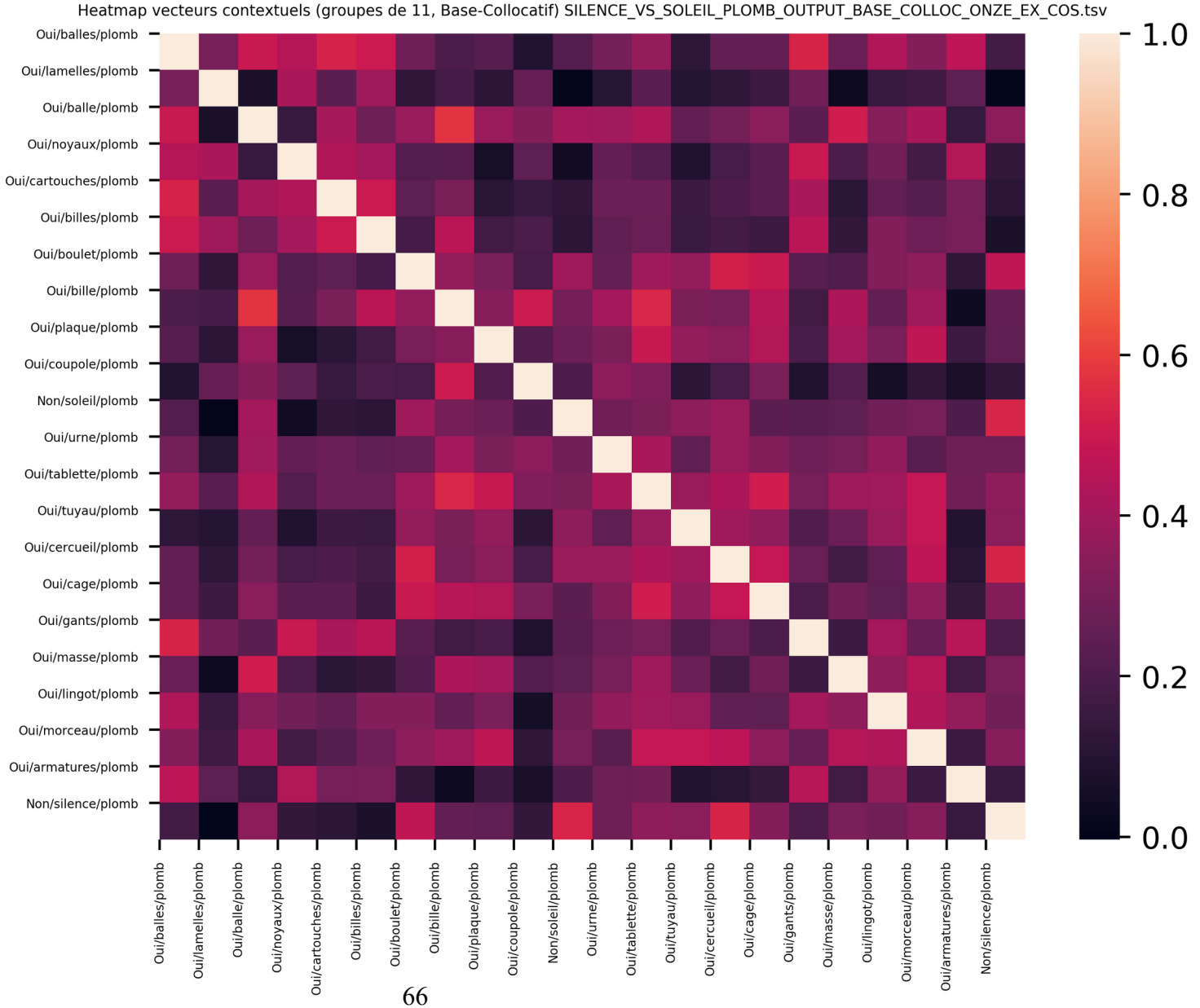
Annexe 4

Exemple de fichier de sortie (fastText)

Ligne	Compositionnel	Base	Genre base	Vecteur base	Collocatif	Vecteur collocat	Phrase
2	Non	colère	féminin	0.0018 -0.0523	saine	-0.0347 -0.0166	Il aurait tant aimé que les deux fouettent enfin la terre d'une saine colère et redon
3	Non	colère	féminin	0.0018 -0.0523	saine	-0.0347 -0.0166	Après, au sortir de la projection, le sentiment oscille entre gêne poisseuse et sain
4	Non	colère	féminin	0.0018 -0.0523	saine	-0.0347 -0.0166	Comme elle ne s'est pas mise en colère depuis très longtemps, n'en ayant plus g
5	Non	colère	féminin	0.0018 -0.0523	saine	-0.0347 -0.0166	Cette productivité qui étonne, inquiète parfois même les proches, trouve sa sourc
6	Non	colère	féminin	0.0018 -0.0523	saine	-0.0347 -0.0166	Pauline le sentait soulevé de colère, mais une saine colère.
7	Non	colère	féminin	0.0018 -0.0523	saine	-0.0347 -0.0166	Les cinquante récits que je dus subir de leur promenade en ville, de cette exhibiti
8	Non	colère	féminin	0.0018 -0.0523	saine	-0.0347 -0.0166	Il serait venu pour faire un scandale, une sorte de provocation entre le politique e
9	Non	colère	féminin	0.0018 -0.0523	saine	-0.0347 -0.0166	Elle escomptait qu'il suffirait de dévoiler les faits pour que le monde entier soit sec
10	Non	colères	féminin	-0.007 0.0041 0	saines	-0.0235 -0.0193	Les vrais dandys Leroy en profite pour nous livrer quelques saines colères contre
11	Non	colères	féminin	-0.007 0.0041 0	saines	-0.0235 -0.0193	Au début de son premier mandat, Boris Eltsine plaisait pour son côté moujik doté
12	Oui	vie	féminin	-0.0648 0.0506	saine	-0.0347 -0.0166	Faut que tu dormes plus, il lui disait, faut que tu te reposes, faut que t'arrêtes de s
13	Oui	personne	féminin	0.0077 -0.0247	saine	-0.0347 -0.0166	Une personne saine et enjouée, me dis-je, n'aurait pas fait ce que je faisais.
14	Oui	peau	féminin	0.0334 0.033 0	saine	-0.0347 -0.0166	La louve s'élança aussitôt vers le petit et entreprit de le nettoyer à grands coups d
15	Oui	nourriture	féminin	0.0104 -0.005 0	saine	-0.0347 -0.0166	Une nourriture saine et complète lui suffit.
16	Oui	chair	féminin	-0.1131 0.0681	saine	-0.0347 -0.0166	Et puis, cette propreté, ce visage massif et diaphane, cette foulée aérienne, et ce
17	Oui	écorces	féminin	0.0362 0.0305	saines	-0.0235 -0.0193	Du revers de ma lame j'écartais une feuille quand un jeune arbre aux écorces sa
18	Oui	dents	féminin	0.0614 0.2108	saines	-0.0235 -0.0193	Du coin de l'oeil, je l'examine : le teint rose, la peau bien tendue sur les os qu'on d
19	Oui	bouche	féminin	0.0627 0.0626	saine	-0.0347 -0.0166	– Au moins, toi, dit-elle en me souriant de sa large bouche saine et éclatante, tu e
20	Oui	jambe	féminin	-0.0666 0.0663	saine	-0.0347 -0.0166	Sa jambe saine, sur laquelle il avait transféré tout son poids, tremblait de fatigue.
21	Oui	feuilles	féminin	0.0795 0.033 0	saines	-0.0235 -0.0193	À bout de souffle, il s'appuya contre un saule et nota qu'on aurait dû élaguer l'arb

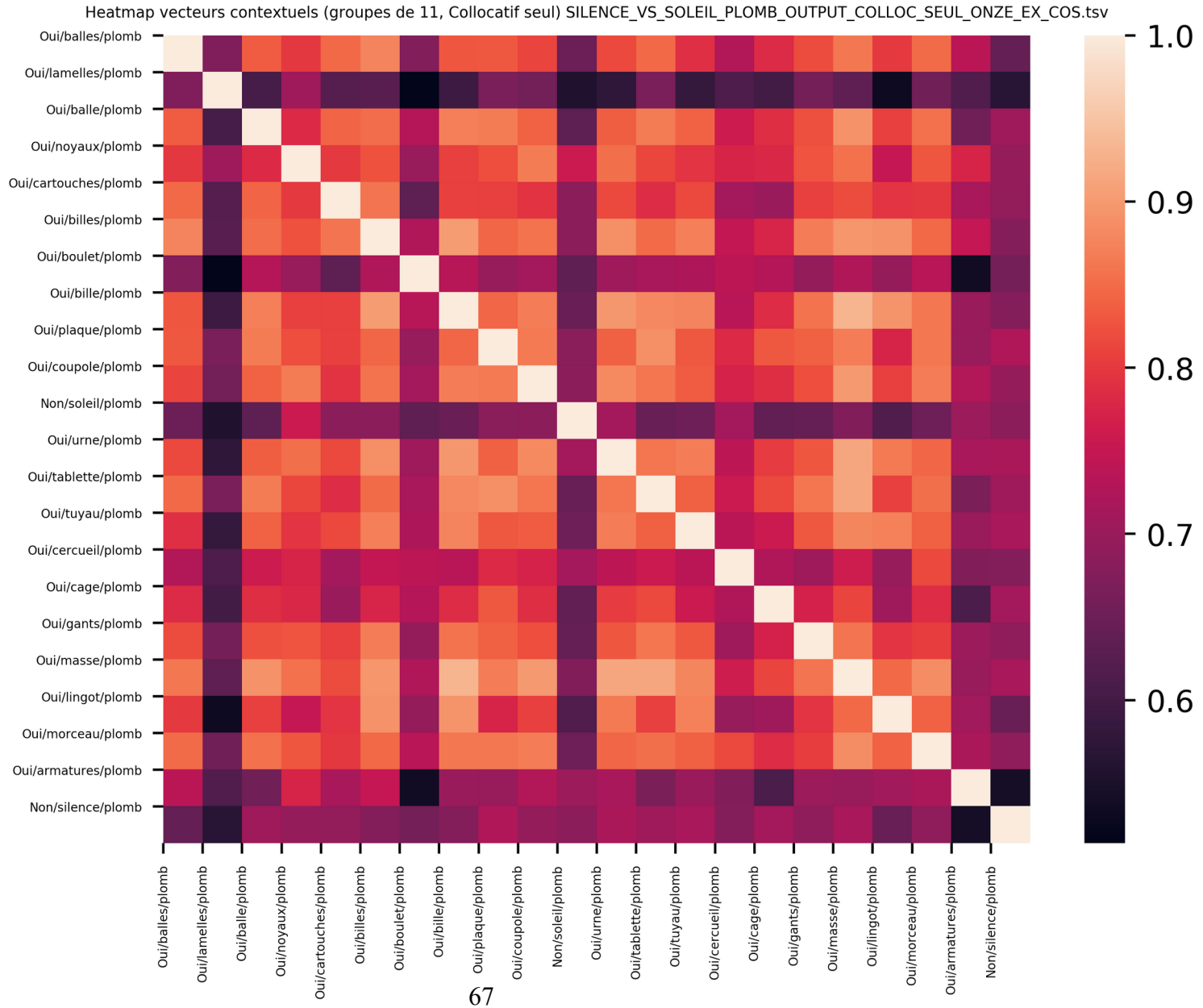
Annexe 5

***Heatmap* silence/soleil de plomb(1)**



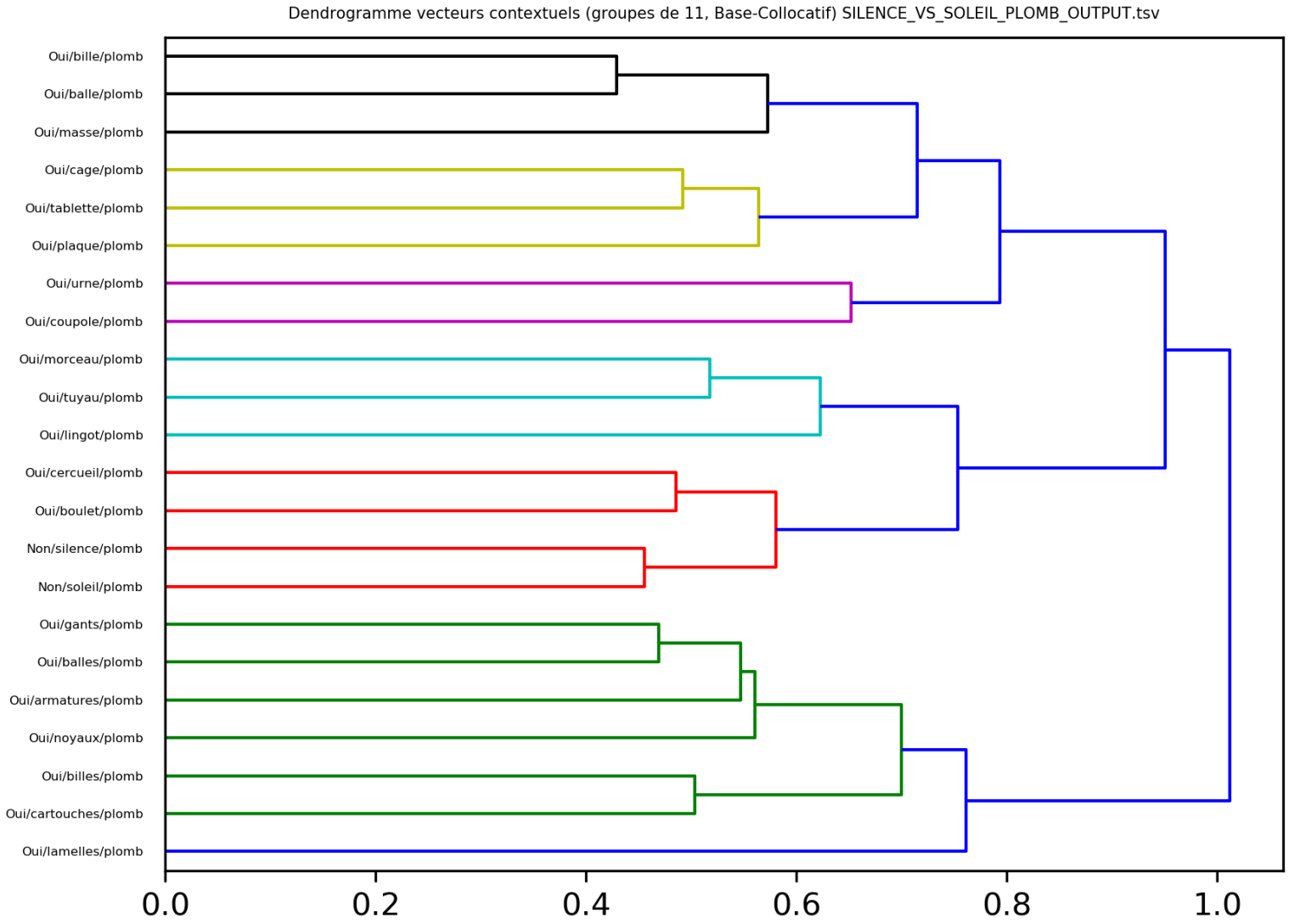
Annexe 6

Heatmap silence/soleil de plomb(2)



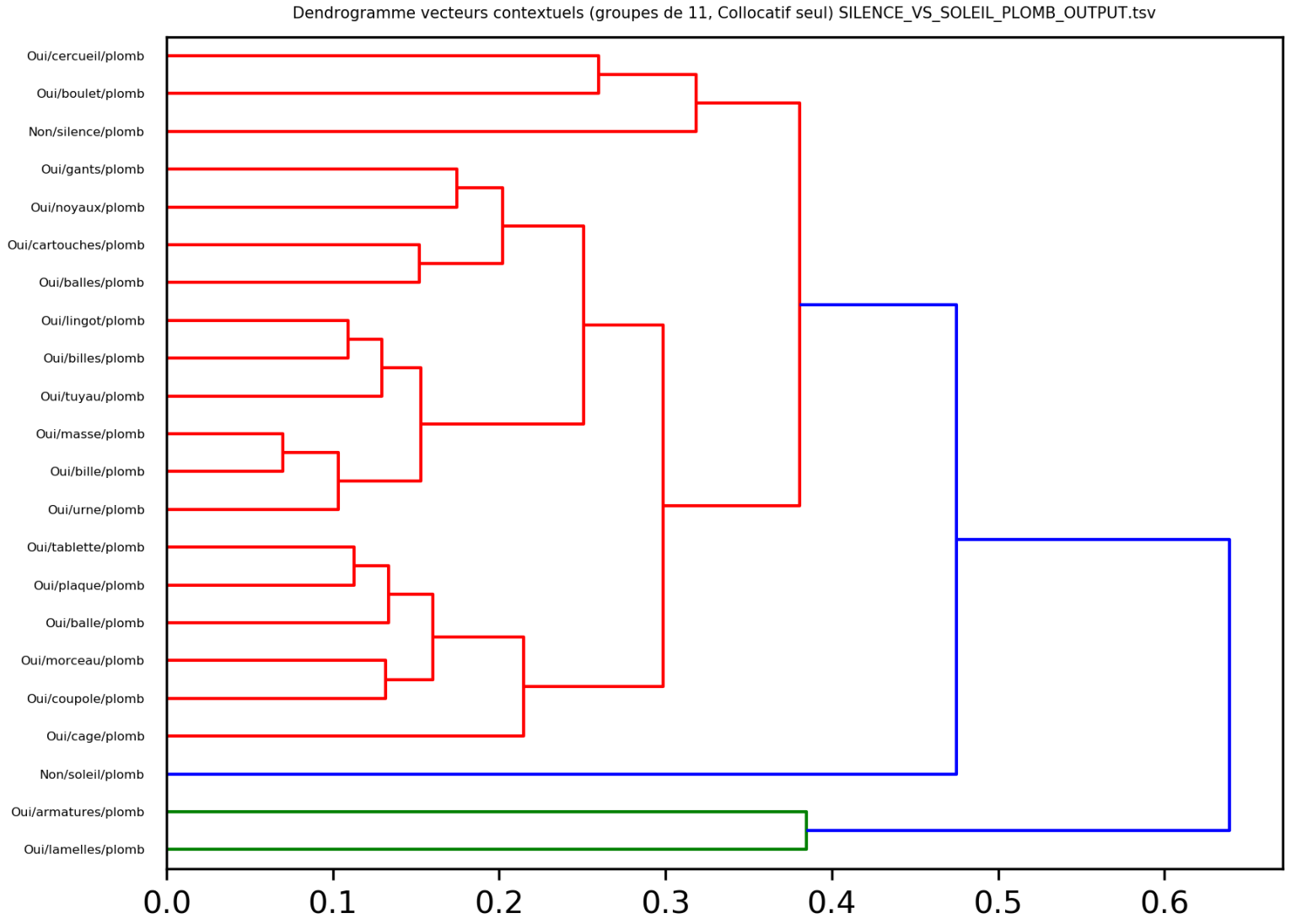
Annexe 7

Dendrogramme silence/soleil de plomb(1)



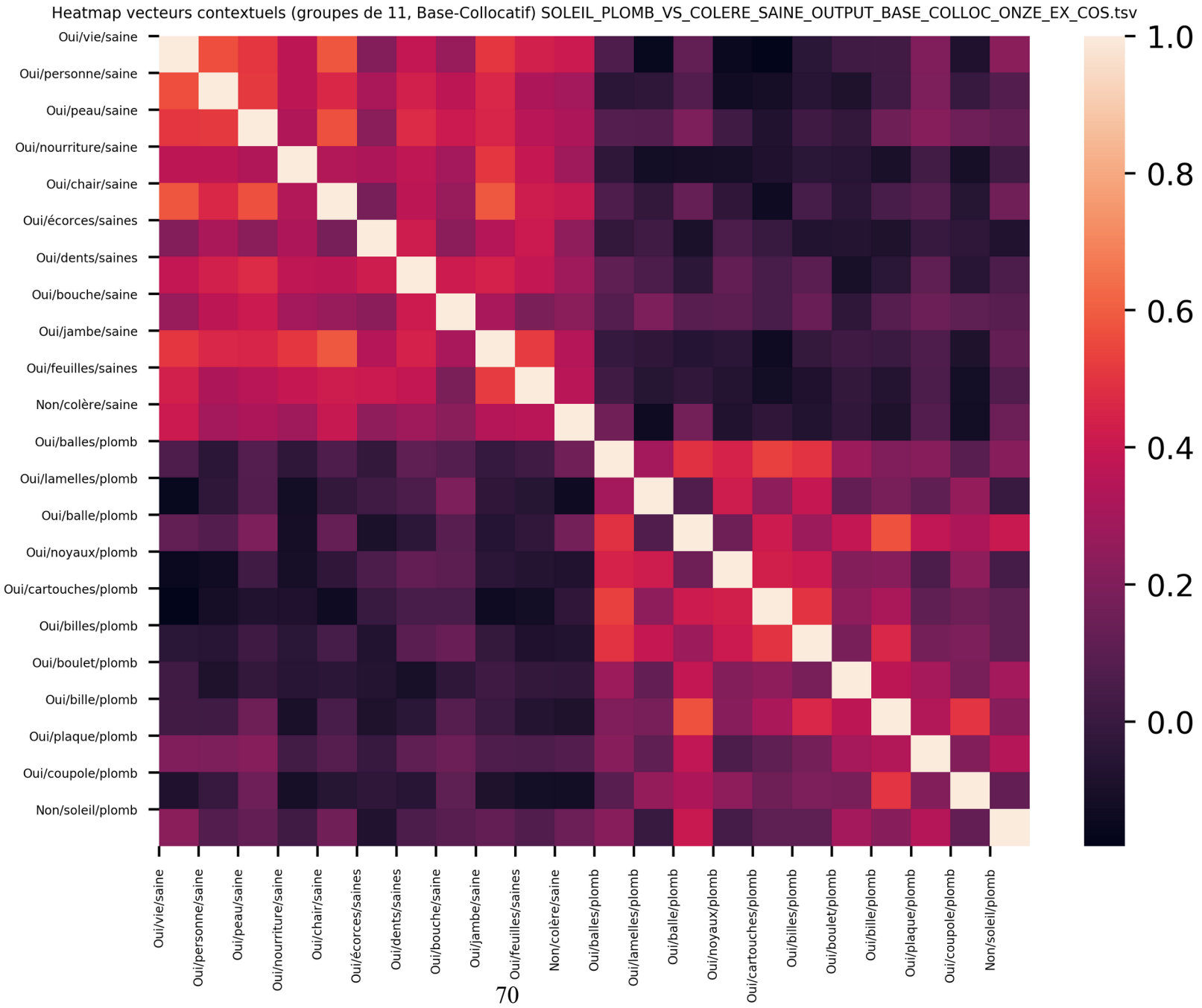
Annexe 8

Dendrogramme silence/soleil de plomb(2)



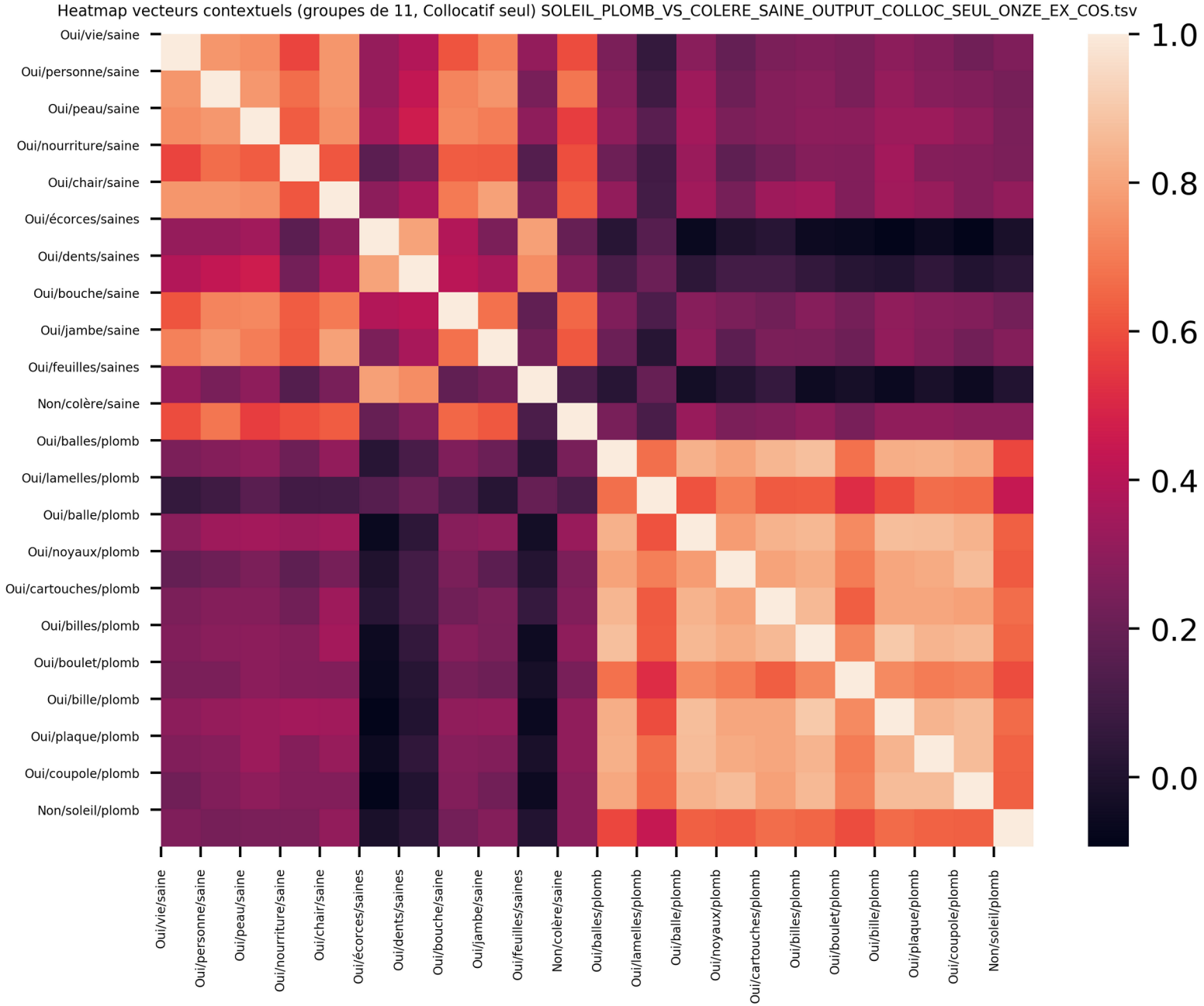
Annexe 9

***Heatmap* soleil de plomb/colère saine(1)**



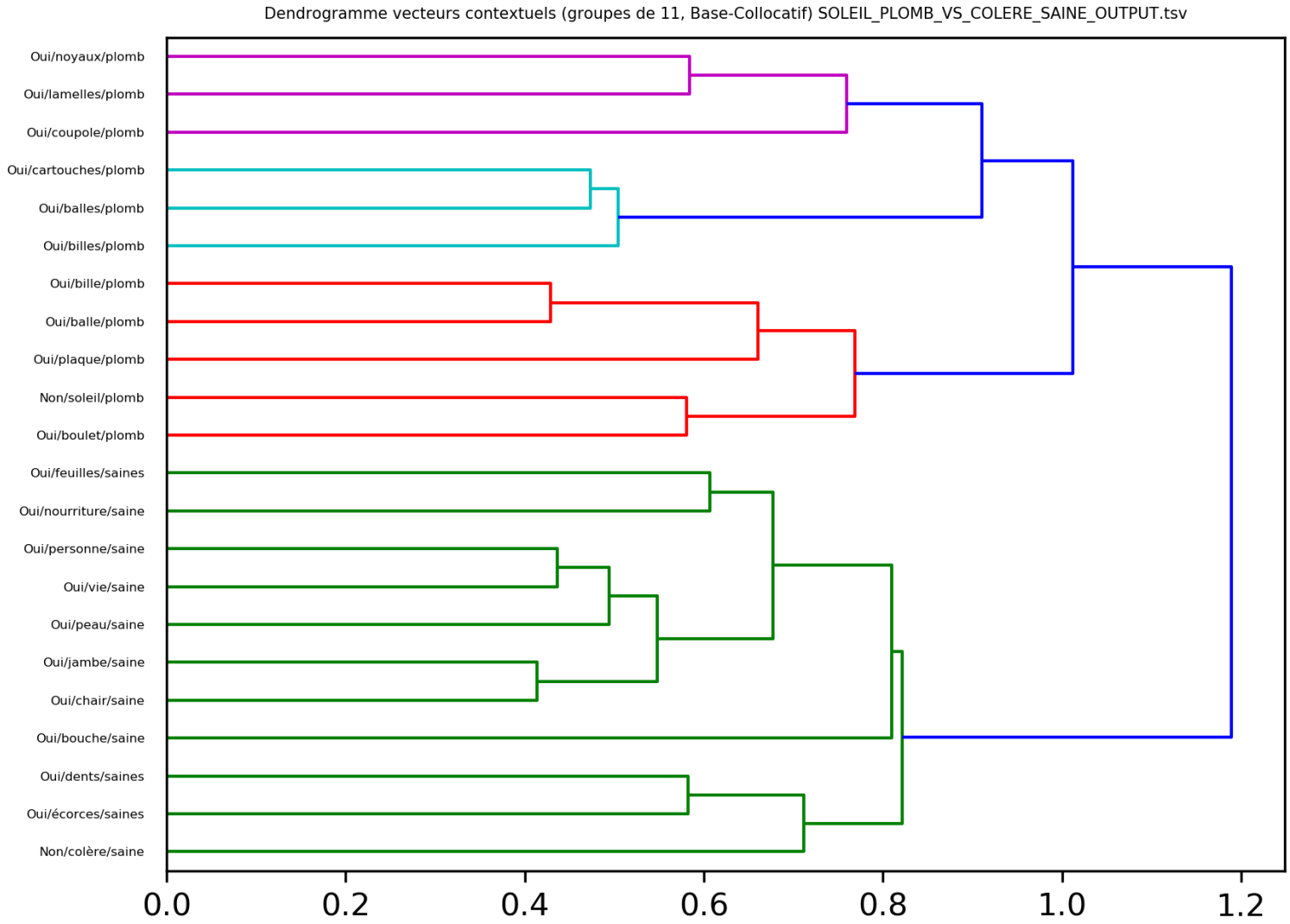
Annexe 10

***Heatmap* soleil de plomb/colère saine(2)**



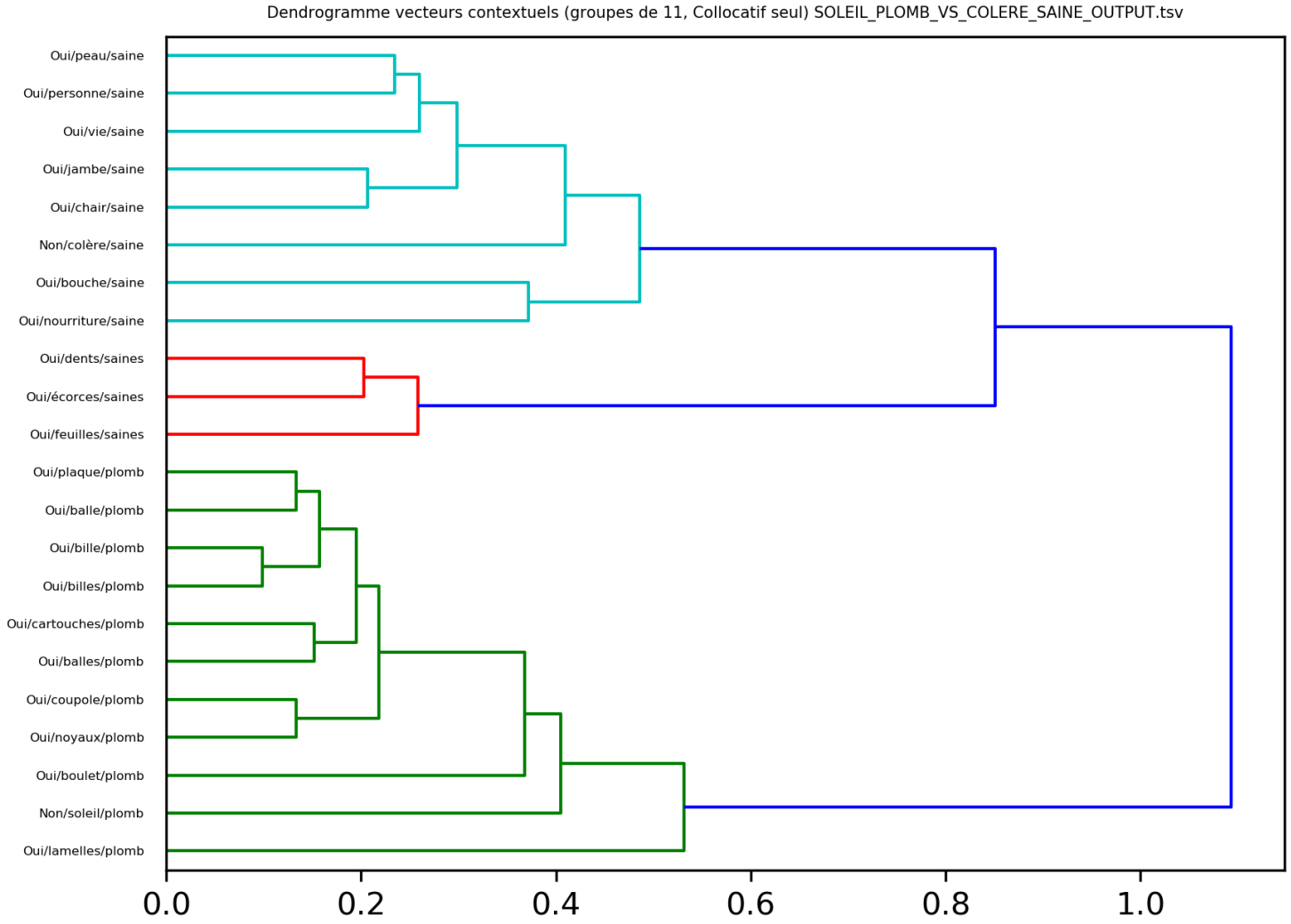
Annexe 11

Dendrogramme soleil de plomb/colère saine(1)



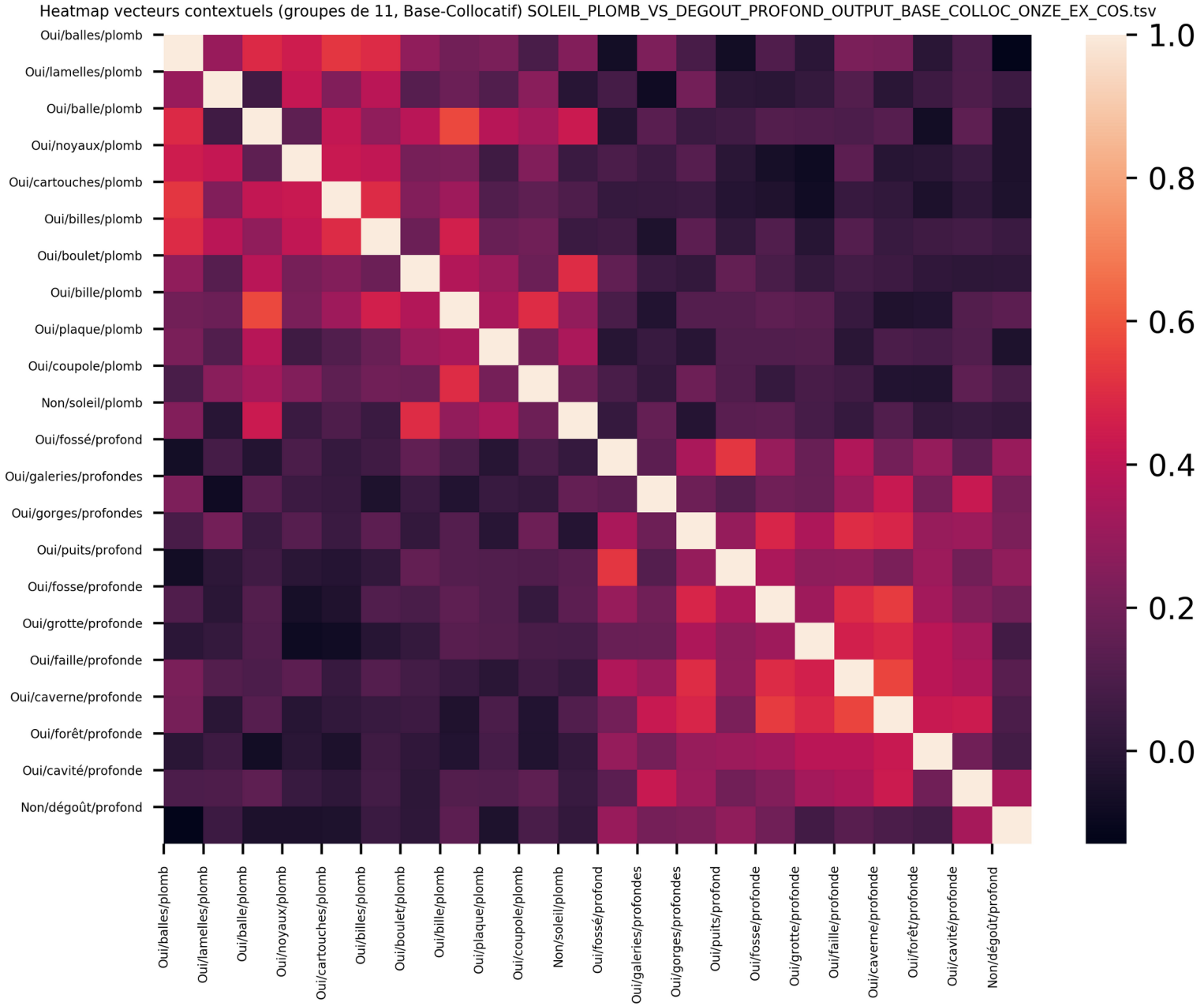
Annexe 12

Dendrogramme soleil de plomb/colère saine(2)



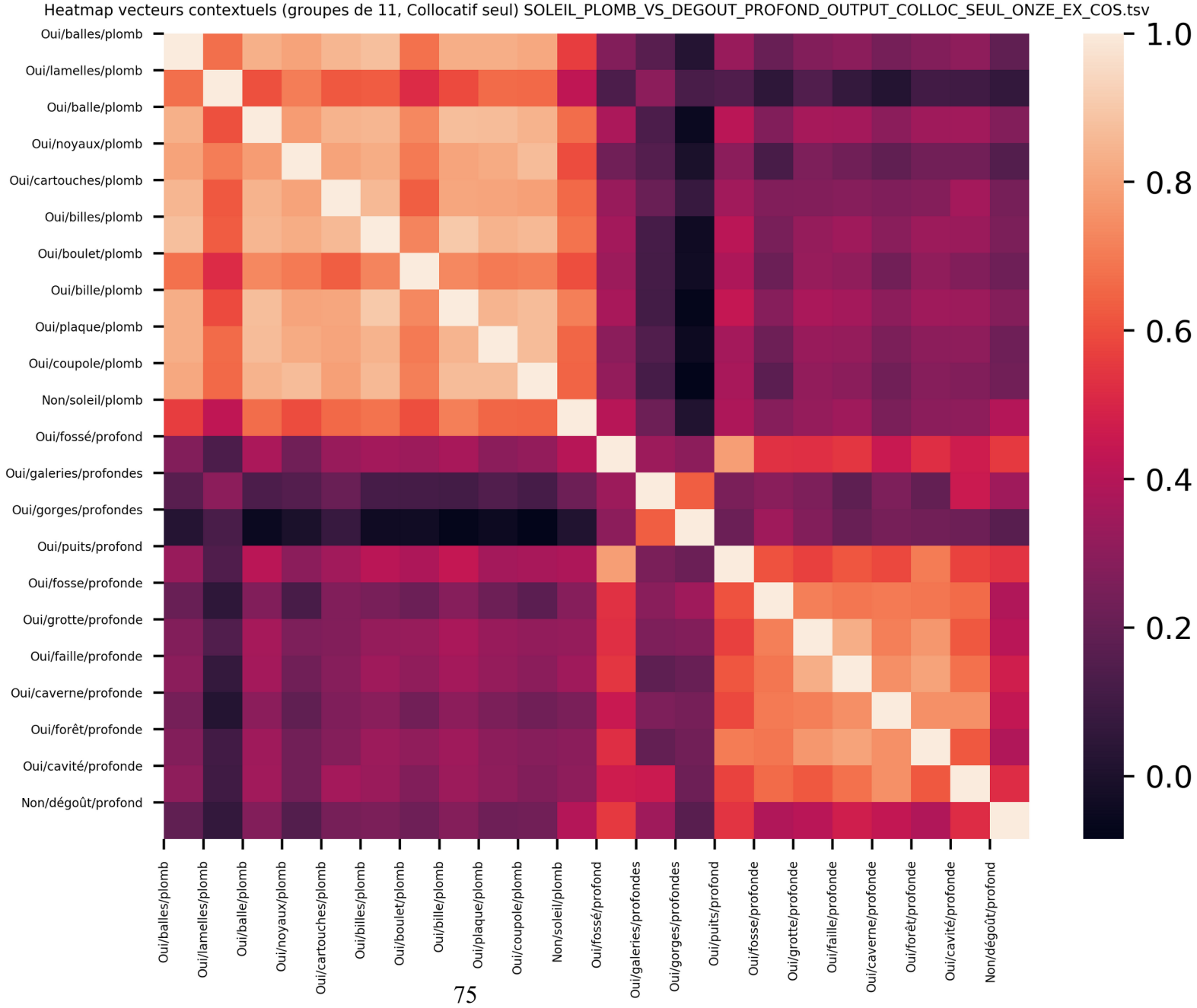
Annexe 13

***Heatmap* soleil de plomb/dégoût profond(1)**



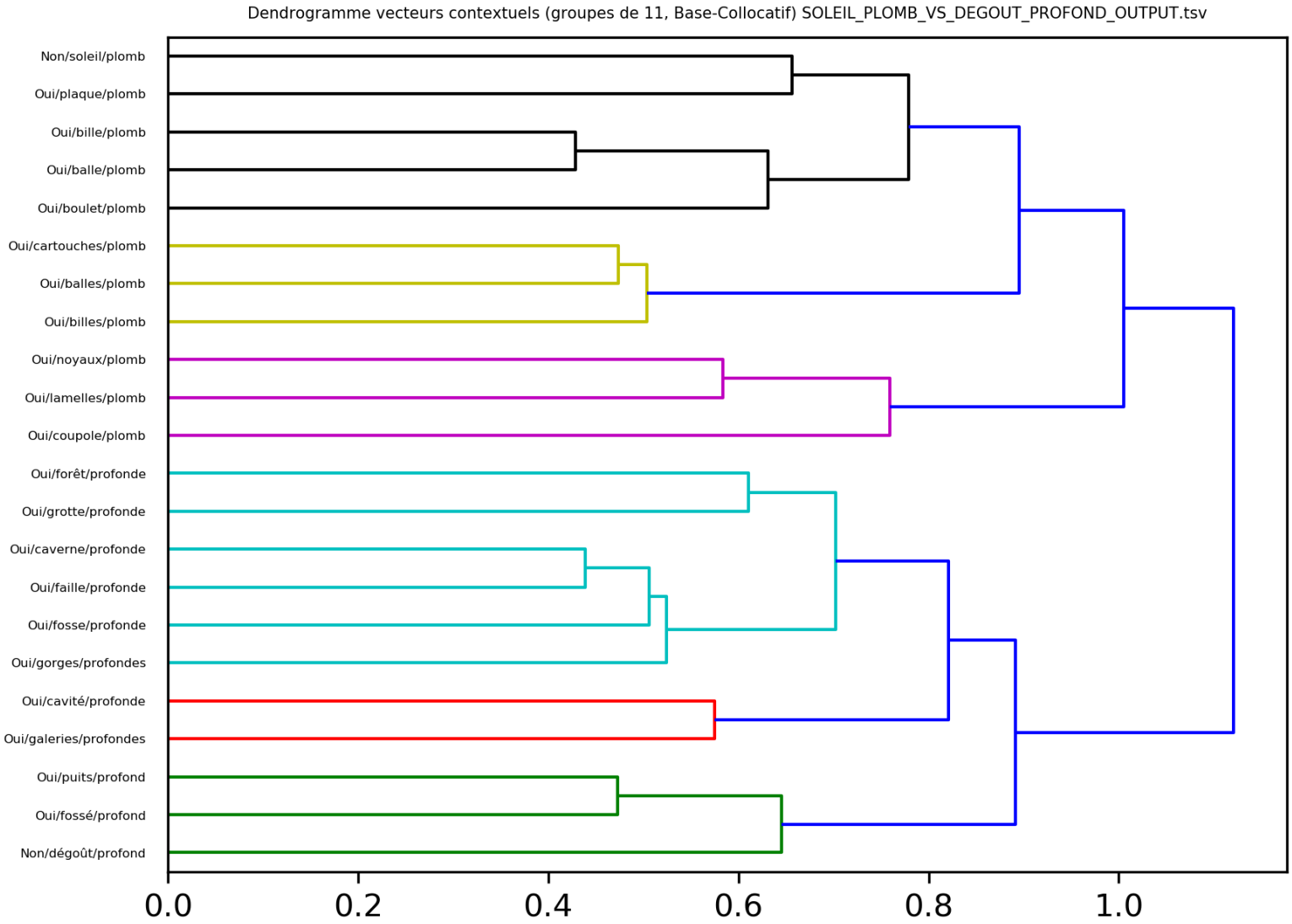
Annexe 14

***Heatmap* soleil de plomb/ dégoût profond(2)**



Annexe 15

Dendrogramme soleil de plomb/dégoût profond(1)



Annexe 16

Dendrogramme soleil de plomb/dégoût profond(2)

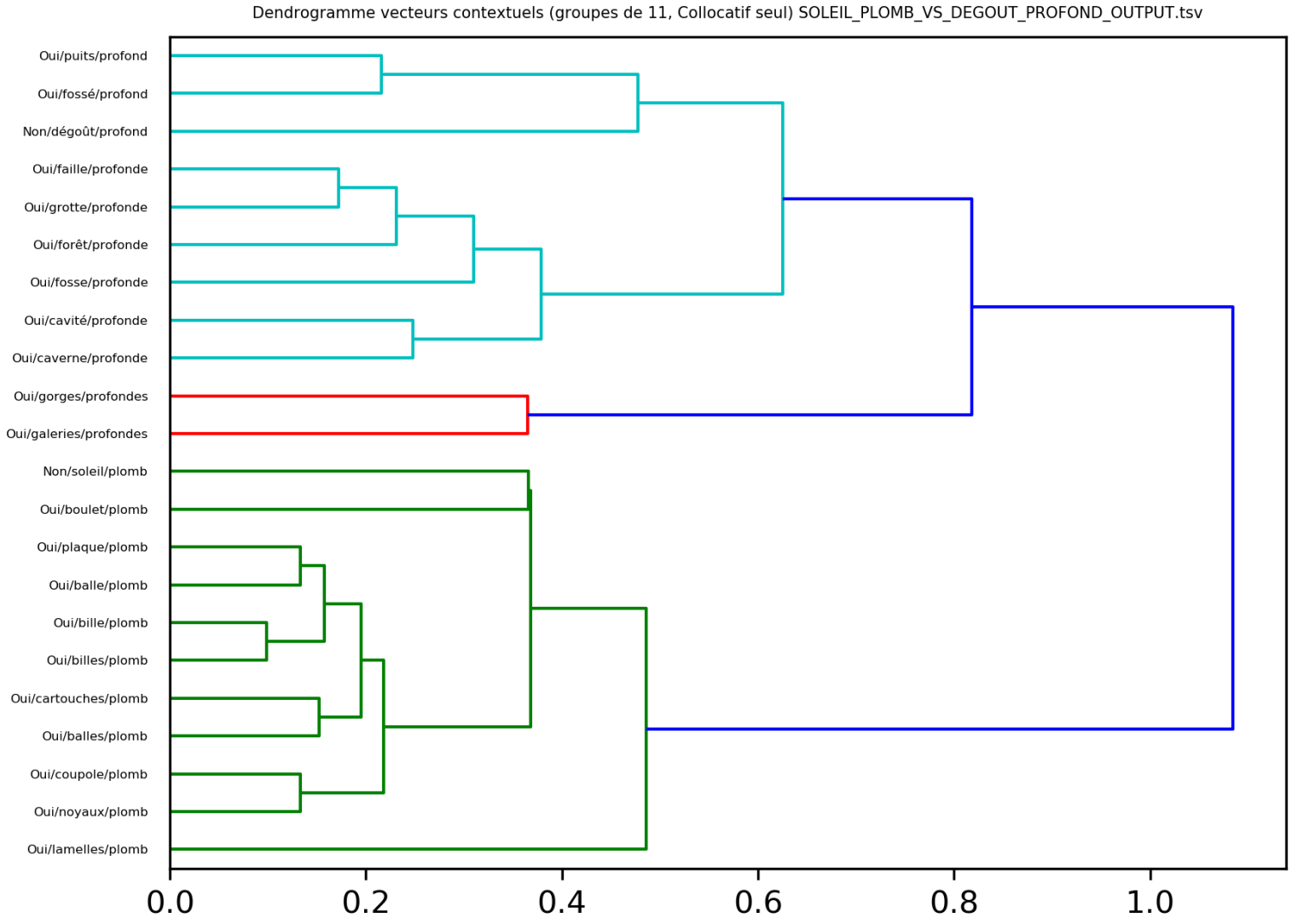


Table des matières

MOTS-CLÉS :polysémie, FlauBERT, plongement de mots, vecteurs contextuels

RÉSUMÉ

Chaque langue est constituée de mots qui lui sont propres. Dans la plupart des cas, ceux-ci sont polysémiques - ils possèdent plusieurs sens. La modélisation de la polysémie en Traitement Automatique de la Langue est une tâche difficile lorsqu'il s'agit de vecteurs de mots ; les systèmes de plongements de mots traditionnels ont certaines difficultés à traiter la polysémie. À l'aide de FlauBERT, qui est un nouveau modèle de langue développé en 2019, nous verrons qu'il est maintenant plus facile de traiter de la polysémie, notamment grâce à des vecteurs de mots contextualisés. Le contexte entier d'une phrase est pris en compte par FlauBERT afin de représenter chaque mot sous forme de vecteur. Après une brève analyse des différents domaines en jeu, je présenterai dans ce mémoire les différentes expérimentations que j'ai effectuées à l'aide des vecteurs de mots du système FlauBERT.

KEYWORDS : Polysemy, FlauBERT, Word embeddings, contextual vectors

ABSTRACT

Each language is made up of its own words. In most cases, these are polysemic, they have several meanings. Modeling polysemy in Automatic Language Processing is a difficult task when it comes to word vectors and traditional word embeddings systems have some difficulties in dealing with polysemy. Using FlauBERT, which is a new language model developed in 2019, we will see that it is now easier to deal with polysemy, especially with contextualized word vectors. The entire context of a sentence is taken into account by FlauBERT in order to represent each word as a vector. After a brief analysis of the different domains involved, I will present in this paper the different experiments I have performed using FlauBERT word vectors.