TER - Rapport Final

Plongements de mots translingues et alignement de textes parallèles pour les dialectes alsaciens

Encadrante: Delphine BERNHARD

1. Contexte

La problématique de recherche est que les dialectes alsaciens ont de grandes variations graphiques. Comme l'alsacien est plus parlé qu'écrit, il y a beaucoup de variétés à l'écrit et il n'y a pas de standardisation orthographique. On peut prendre comme exemple le mot *aider*. En alsacien, il est tout à fait possible de l'écrire avec les différentes formes *halfa, halfe, helfe et hëlfe*. Ce type de variation à l'écrit peut être observée pour tous les mots. En plus de cette contrainte, il y a aussi très peu de ressources pour l'alsacien. Cette variabilité à l'écrit et le manque de ressources pour le dialecte alsacien ajoutent des contraintes pour le développement de programmes de traitement automatique des langues (TAL). Pour résoudre ces contraintes, une piste de recherche intéressante concerne l'utilisation de plongements de mots (*word embeddings*) translingues pour les dialectes alsaciens. Dans ce travail, nous nous sommes dans un premier temps concentrés sur les plongements de mots monolingues. La première phase est de comprendre les caractéristiques en monolingue avant de passer en translingue. La deuxième phase sera d'étudier les plongements de mots translingues.

2. Introduction

Le traitement automatique des langues (TAL) aussi appelé en anglais *Natural Language Processing* (NLP) est utilisé en informatique pour développer des méthodes pour le traitement des langues. "Sur la planète, il y a approximativement 7000 langues qui sont parlées par les humains, mais seulement une vingtaine de langues sont parlées par la moitié de la population" (*Cours Algorithmes du texte - 1_IntroductionTAL slides - Delphine BERNHARD*). Cela veut donc dire qu'une partie des langues sont en danger et ne disposent pas d'outils de TAL. Il existe différentes méthodes pour analyser une langue sous forme écrite.

Avant de passer aux évaluations et aux résultats, il est intéressant de connaître la situation linguistique en Alsace. Il faut savoir qu'il n'y a pas un seul dialecte alsacien, mais plusieurs, cela dépend de la zone géographique en Alsace. On a donc des dialectes alémaniques et franciques. La figure 1 présente une carte linguistique de l'Alsace qui regroupe les 6 dialectes alsaciens traditionnellement distingués. Il faut savoir que dans ces 6 dialectes, il est tout à fait possible d'avoir encore des sous-dialectes.

4 dialectes germaniques :

- Bas alémanique, Haut alémanique, Francique méridional palatin et Francique rhénan lorrain

2 langues d'oïl :

- Lorrain et Franc-comtois

Avant d'être appelé alsacien, il était appelé auparavant allemand, puis allemand alsacien. Ceci relève des parlers alémaniques et franciques. Les dialectes franciques sont aussi régulièrement inclus dans la dénomination "alsacien". Les frontières avec les langues ne sont pas forcément les mêmes frontières géographiques entre les pays, c'est pour cela qu'on retrouve l'alémanique dans différents pays. L'aire linguistique alémanique est bien plus large que la seule région Alsace, incluant notamment la Suisse.[1]

Ce qui est important à savoir c'est qu'il n'y a pas de norme à l'écrit, il n'y a pas d'orthographe. Chacun écrit comme bon lui semble. Les dialectes sont des langues parlées non codifiées. La langue normalisée, écrite et codifiée correspondante aux dialectes est l'allemand standard. Les dialectes sont antérieurs à l'allemand standard.[2]

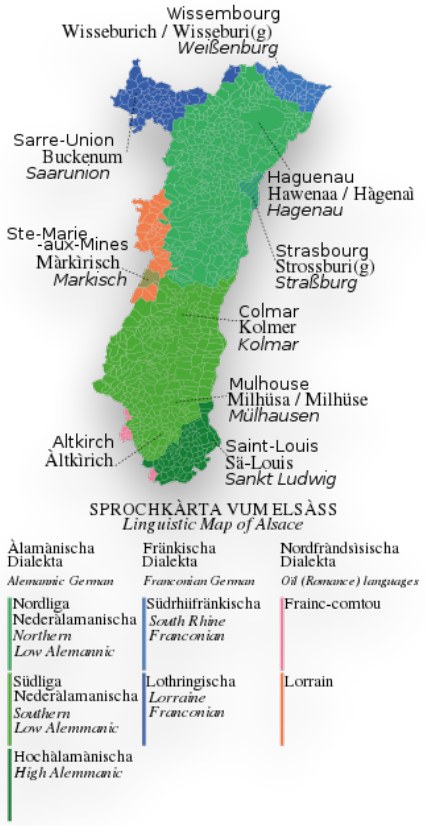

***Figure 1** : Carte linguistique*
https://en.wikipedia.org/wiki/Alsatian_dialect#/media/File:Linguistic_Map_of_Alsace.svg

3. Difficultés

Comme il n'y a pas de norme à l'écrit, ceci a un impact sur tous les traitements qu'on pourra faire parce qu'il y a le traitement de la variation dialectale, mais aussi l'absence de norme orthographique. Du coup, il y a de la variation à l'intérieur même de l'espace, mais en plus de cela il n'y a pas de norme graphique donc tout se croise. C'est pour cela qu'il n'y a pas une seule orthographe, mais plusieurs pour un mot, ce qui n'est pas le cas pour le français. Il est possible d'avoir un tout petit corpus, mais beaucoup de variations. Ces difficultés posent beaucoup de problèmes pour le traitement automatique de la langue alsacienne. (Nguyen & Grieve, 2020)

Les défis auxquels il faut faire face lorsque l'on travaille sur ce type de données, c'est que l'on dispose de données en petite quantité avec beaucoup de variations ce qui est très problématique pour les outils informatiques. Pour une machine, si l'on change un accent ce n'est plus le même mot. En regardant les résultats des expériences dans le rapport, on peut penser que les résultats sont mauvais et que dans plein de cas la machine ne trouve pas le bon résultat, mais en vrai non. Si l'on compare l'alsacien avec des langues comme le français, on n'arrivera pas à ce niveau-là, car en alsacien on dispose de données en petite quantité avec beaucoup de variations.

[1] https://www.olcalsace.org/fr/observer-et-veiller/definition-de-la-langue-regionale *(visité le 20/04/2021)*

[2] https://theconversation.com/lalsacien-nexiste-pas-was-isch-los-quen-est-il-159487 *(visité le 14/05/2021)*

4. Liste de contributions

- Test des plongements de mots pour une langue très peu dotée qui est l'alsacien
- Extraction du texte alsacien de pièces de théâtre
- Traitement des mots coupés
- Tokénisation (découpage en tokens) du corpus
- Utilisation de méthodes et outils
  - FastText, Word2Vec et Magnitude
  - Comparaison des modèles en les testant avec différents paramètres
- Évaluation et analyse de la qualité des vecteurs
  - Comparaison des corpus
- Scripts d'évaluation
  - Constitution de données d'évaluations qui n'existaient pas dans cette langue
  - Lexique bilingue évaluation
  - Deux mesures pour une analyse approfondie des mots (Fichier Excel)
- Conclusion :
  - Il vaut mieux avoir un corpus plus petit, mais de la famille du dialecte qui nous intéresse plutôt qu'un gros corpus avec des dialectes mélangés.

5. Travail réalisé

5.1 Références bibliographiques

Pour bien débuter le projet, j'ai dû dans un premier temps lire des références bibliographiques qui étaient citées dans le sujet. Ceci m'a permis d'avoir une vue générale sur les différentes méthodes qui existaient pour les plongements de mots, mais aussi leur fonctionnement. J'ai aussi pu consulter les outils qui étaient décrits dans les références et revoir les bases de Python.

5.2 Données et prétraitement

5.2.1 Corpus

Les données sur lesquelles les modèles sont évalués sont des corpus en alsacien et alémanique :

- THÉÂTRE pour le corpus de 8 pièces de théâtre
- OSCAR pour le corpus alémanique OSCAR
- DIVERS pour le corpus d'alsacien contemporain comportant divers genres de textes

Les premières évaluations sont faites sur THÉÂTRE qui a été numérisé par la Bibliothèque nationale et universitaire de Strasbourg[3]. Voici les titres des différentes pièces : *Der Pfingstmontag, D'r Hofnarr Heidideldum, Chrischtowe, Sainte-Cécile!, D'r poetisch Oscar, E Daa im Narrehüss, D'r Hoflieferant, In's Ropfer's Apothe[4].*

[3] https://numistral.fr *(visité le 11/03/2021)*

[4] Téléchargement des pièces: https://git.unistra.fr/methal/methal-sources *(visité le 11/03/2021)*

Pour avoir plus d'informations concernant le projet *MeThal Vers une macroanalyse du théâtre en alsacien*[5]*.* Les participants au projet sont Pablo Ruiz, Delphine Bernhard, Pascale Erhart, Dominique Huck et Carole Werner.

Après les premières évaluations, le corpus OSCAR[6] sera donné aux modèles en plus du corpus THÉÂTRE. C'est un site qui héberge des millions (*petabytes*) de corpus dans beaucoup de langues différentes. Ils sont obtenus par le filtrage du corpus Common Crawl. C'est quoi Common Crawl ?
Comme le souligne commoncrawl.org « *We build and maintain an open repository of web crawl data that can be accessed and analyzed by anyone. The Common Crawl corpus contains petabytes of data collected since 2008. It contains raw web page data, extracted metadata and text extractions.* » (Ils construisent et entretiennent un dépôt ouvert de données d'exploration du Web qui peut être consulté et analysé par n'importe qui. Les archives Web de Common Crawl se composent de pétaoctets de données collectées depuis 2008 [Ma traduction]).[7]

OSCAR est donc principalement destiné à être utilisé dans la formation de modèles linguistiques non supervisés pour le TAL. Le corpus était disponible en deux exemplaires. Il y avait un fichier original et un fichier dédupliqué. Le fichier original a une taille de 841,750 mots pour 5.0 MB et l'autre fichier contient 459,001 mots pour 2.8 MB. Dans le fichier dédupliqué, il y a des phrases qui ont été supprimées par rapport à l'autre fichier à cause de la duplication. C'est celui-ci que j'ai pris pour l'évaluation. C'est un corpus alémanique donc avec un mélange de mots en allemand.

Après le travail réalisé pour le rapport intermédiaire, j'ai rajouté un nouveau corpus DIVERS pour les évaluations. Le nouveau fichier est un corpus alsacien contemporain envoyé par l'enseignante Delphine BERNHARD. Ce fichier contient 3 colonnes. La seule qui nous intéresse est la deuxième, car elle contient les phrases alsaciennes déjà tokenisées. Il provient tout simplement de diverses sources collectées de différentes manières. Le fichier fait 5 MB et contient 48762 lignes.

5.2.2 Extraction du texte de pièces de théâtre

Les pièces de théâtre étaient disponibles dans un format XML-TEI. Il a donc fallu extraire le texte pertinent afin de constituer un corpus de texte brut utilisé pour la création des plongements de mots. Il y a deux catégories de pièces, une qui contient du texte en alsacien et une où les pièces sont écrites en allemand-alsacien. J'ai donc créé un script Python *extraire.py* qui utilisait la librairie Python *BeautifulSoup*. À la fin de l'extraction, pour chaque pièce de théâtre un fichier en format TXT a été créé qui contenait le texte extrait. Seul le texte en alsacien a été conservé.

5.2.3 Traitement des mots coupés

Les mots étaient coupés en deux, car ils étaient répartis sur la fin d'une ligne et le début de la ligne suivante dans les différents fichiers. Un mot comme *ge- brennt* était coupé en deux avec un espace entre le tiret et le mot suivant. Ceci posait problème pour l'entrainement du modèle, car il prenait ceci comme deux mots sauf que c'était en fait un

---

[5] https://methal.pages.unistra.fr/ *(visité le 11/03/2021)*

[6] https://oscar-corpus.com/ *(visité le 02/03/2021)*

[7] https://commoncrawl.org/ *(visité le 02/03/2021)*

mot. Il fallait donc le remettre ensemble soit en *ge-brennt* ou *gebrennt*. Pour l'instant, ma fonction le convertit en *ge-brennt*.

5.2.4 Tokénisation (découpage en tokens) du corpus

La tokenisation d'un texte consiste à le découper en mots. Par exemple quand on rencontre un signe de ponctuation ou des chiffres, etc afin de mieux le manipuler. Avant de donner un corpus de textes à notre modèle pour qu'il puisse s'entrainer, j'ai dû tokeniser tous les textes qui ont été extraits auparavant (*voir figure 2*). La tokenisation est faite avec le module *alsatian_tokeniser.py* qui a été conçu par l'enseignante Delphine BERNHARD. Le module est seulement utilisable pour des textes alsaciens. J'ai un fichier *tokenisation.py* qui importe des fonctions issues du fichier *alsatian_tokeniser.py* et à la fin crée un fichier en format TXT qui contient tous les textes tokenisés. La tokenisation se fait pour chaque phrase donc dès qu'il rencontre un signe de ponctuation.

Corpus	Nombre de tokens
THÉÂTRE	127643
OSCAR	523612
DIVERS	328829

***Figure 2** : Taille des différents corpus*

5.3 Utilisation de méthodes et outils

5.3.1 Plongements de mots ?

Les plongements de mots ou en anglais *word embeddings* est la technique de représentation d'un mot en vecteurs de dimension fixe. Chaque mot est donc représenté par un vecteur de nombres réels. Par exemple pour le mot *chat* on aurait [0.23432, -3.23421, 1.23644, etc]. Si le modèle voit ce vecteur, il sait que le mot caché derrière est *chat*. Un mot graphiquement similaire ou des mots qui ont presque le même contexte auront des vecteurs qui seront assez proches les un des autres. Par exemple, le mot chat et chien auront des vecteurs assez proches, car ces mots sont souvent utilisés dans des contextes similaires. C'est ainsi que chaque mot sera représenté par un vecteur de réels, et ils pourront donc être comparés par une mesure mathématique.[8]

5.3.2 FastText

FastText est une librairie pour l'apprentissage des représentations des mots et la classification de phrases. Depuis sa création, elle est de plus en plus utilisée et elle est très performante. Ce qui fait la différence avec d'autres librairies, c'est que FastText utilise des **n-grammes** et **crée** un vecteur pour un mot même s'il ne se trouve pas dans le corpus utilisé pour l'apprentissage (mot hors vocabulaire). Il vaut toujours mieux avoir un grand corpus. Si on a un petit corpus, le modèle ne capturera pas toutes les informations mais FastText s'en sort bien avec des petits corpus. FastText propose un apprentissage supervisé ou non supervisé. Dans notre cas, j'utilise l'apprentissage non supervisé.[9]

---

[8] https://gist.github.com/aparrish/2f562e3737544cf29aaf1af30362f469 *(visité le 12/02/2021)*

[9] https://fasttext.cc/docs/en/unsupervised-tutorial.html *(visité le 11/02/2021)*

Ensuite, FastText propose deux architectures pour le calcul de représentations de mots. On a *Skipgram* et *Cbow* (*continous-bag-of-words*). Cbow est plus **rapide** à apprendre, mais Skipgram donne de **meilleurs** résultats et fonctionne bien avec des corpus de petite taille. Le modèle Skipgram apprend à prédire un mot grâce à un mot proche. Le modèle Cbow prédit le mot en fonction de son contexte. Il existe un hyperparamètre **ws** qui est utilisé pour la taille de la fenêtre contextuelle. (Ruder, Vulić, & Søgaard, 2019)

Sur la gauche de la figure 3 on a Cbow et à droite Skipgram avec l'exemple : Ma voiture est **belle** et rouge.

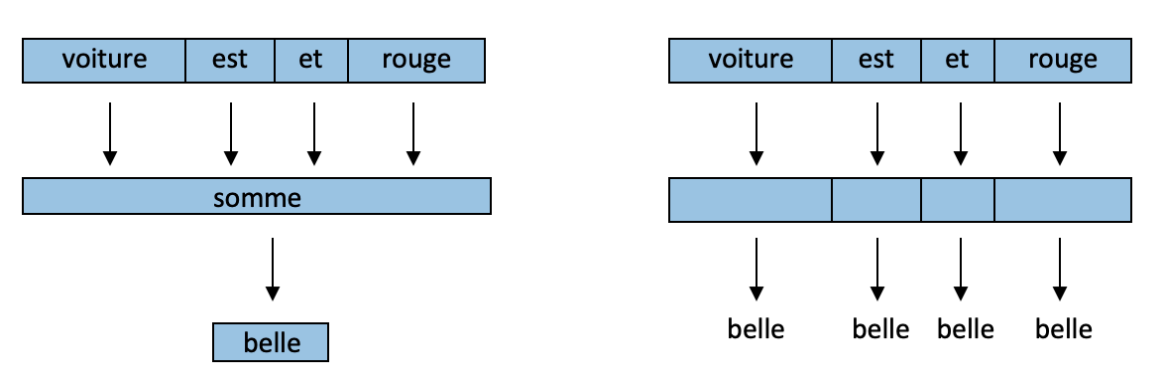

Figure 3 *: Différence entre Cbow et Skipgram*

FastText utilise les *n-grammes,* c'est-à-dire les séquences de caractères qui constituent un mot. Si on prend les hyperparamètres par défaut des n-grammes qui sont **minn à 3** et **maxn à 6**, le mot *hallo* a les n-grammes suivant : *<'hal', 'hall', 'hallo'>, <'all', allo'>, <'llo'>*.
Ces n-grammes ont un vecteur qui est assez proche du vecteur *hallo*. Pour les mots qui ne sont pas dans le vocabulaire, les architectures Skipgram et Cbow vont construire des vecteurs en utilisant les n-grammes de ces mots. On aura donc des vecteurs pour des mots qui ne sont pas dans le corpus.

Pour ce qui est des paramètres, il en existe une dizaine pour FastText mais ceux que j'ai utilisés sont :

- dim - La taille d'un vecteur [100]
- ws - La taille du contexte [5]
- lr - Taux d'apprentissage [0.05]
- Epoch - Le nombre de boucles sur nos données [5]
- minCount - Le nombre minimum d'occurrence d'un mot [5]
- minn - Longueur minimum d'un n-gramme de caractères [3]
- maxn - Longueur maximum d'un n-gramme de caractères [6]

Pour évaluer la qualité des vecteurs, j'utilise la fonctionnalité des *10 plus proches voisins*. Ceci est calculé avec la mesure de similarité en utilisant le cosinus des angles entre deux vecteurs. Si deux mots se ressemblent ou sont dans le même contexte, la mesure de similarité sera proche de 1 ou 0 pour des mots qui n'ont aucune relation entre eux. Il existe encore une autre fonctionnalité qui est souvent utilisée, les analogies. Les analogies nous permettent de poser une question sémantique ou syntaxique. Un exemple **syntaxique** serait *manger est à mangeant ce que vendre est à ?,* la bonne réponse est *vendant*. **Sémantique**: *Paris est à la France ce que Berlin est à ?*, la bonne réponse est

l'Allemagne pour que le résultat soit comptabilisé comme étant correct.

Pour avoir un modèle qui s'entraine bien, il faut ajuster les paramètres, mais il faut aussi que le corpus soit de bonne qualité et d'une taille assez grande. Les résultats des évaluations seront vus dans la section *5.4 Évaluation des résultats*. (Bojanowski, Grave, Joulin, & Mikolov, 2017)

5.3.3 Word2Vec

Word2Vec est aussi une méthode pour les plongements de mots comme FastText et aussi très connue. W2V se trouve dans la bibliothèque Python *Gensim*. De même que FastText, Word2Vec contient deux architectures pour créer des vecteurs, *Skipgram* et *Cbow*. Le fonctionnement des deux architectures ne change pas, elles sont identiques à FastText. Ce qui différencie W2V de FastText c'est que W2V n'utilise pas de **n-grammes** et ne sait pas **gérer** les mots qui ne sont pas dans le corpus (mot hors vocabulaire). (Mikolov, Chen, Corrado, & Dean, 2013)

En général FastText donne de **meilleurs** résultats que Word2Vec car il utilise les n-grammes et les mots non existants du texte. Nous utilisons des textes écrits en alsacien, nous avons donc beaucoup de variations graphiques pour les mots. Pour cette raison, la fonctionnalité n-grammes est très pertinente. Pour pallier peut-être ce problème, il existe une librairie Python qui s'appelle Magnitude (*voir 5.3.4 Magnitude*).

5.3.4 Magnitude

Magnitude est une librairie Python avec de nombreuses fonctionnalités qui est utilisée avec Word2Vec car elle a plus de fonctionnalités et surtout elle sait gérer les **n-grammes** et les mots **non existants** dans le corpus ce que Word2Vec ne sait pas faire. En plus de cela, Magnitude génère un format spécifique pour les vecteurs. Le fichier qui contient les vecteurs est plus léger et donne un accès plus rapide et plus efficace aux plongements de mots. Un tel programme qui est rapide et léger n'existait pas dans le monde de l'apprentissage automatique pour les vecteurs et c'est pour cette raison que Magnitude a été créé. (Patel, Sands, Callison-Burch, & Apidianaki, 2018)

On peut aussi utiliser Magnitude pour gagner en performance, car il est plus rapide que FastText et Word2Vec et prend moins de place en mémoire. Cela est donc intéressant pour de très grands corpus avec des dimensions de vecteurs grands.[10]

5.4 Évaluation des résultats

Une des étapes les plus importantes consistait à évaluer et analyser la qualité des vecteurs de nos modèles. Il fallait trouver un modèle qui était stable sur les corpus THÉÂTRE, OSCAR et DIVERS. Il fallait voir si le modèle qui était stable sur le corpus THÉÂTRE l'était toujours avec les autres corpus. Toutes les évaluations ont été créées à ma façon en suivant une méthodologie. Les modèles ont été testés sur FastText et Word2Vec. **Comme Word2Vec n'utilise pas la fonctionnalité n-grammes et qu'il ne gère pas les mots qui ne sont pas dans les corpus, les résultats n'étaient pas bons même en utilisant Magnitude sur Word2Vec.** Pour la suite, les évaluations seront faites

[10] https://github.com/plasticityai/magnitude *(visité le 27/02/2021)*

uniquement avec FastText. Je précise que ces premières évaluations et analyses ont été faites **uniquement** sur le corpus THÉÂTRE. Ensuite, les évaluations ont été faites en ajoutant les nouveaux corpus alsaciens.

5.4.1 Fréquence des mots

La fréquence des mots est le nombre d'occurrences d'un mot dans le texte. Comme on peut le voir sur la figure 4, il y a 11 mots dans le texte qui apparaissent plus de 1000 fois, 19 mots qui apparaissent au moins 750 fois, etc. Ceci est important à savoir, car cela nous montre que notre corpus THÉÂTRE a 16315 mots différents. Ce qui est **très petit** pour qu'un modèle s'entraine bien. Ceci nous laisse déjà présager que les résultats ne seront pas très bons, car notre modèle n'a pas assez de ressources pour apprendre, mais c'était attendu. Il faut aussi savoir qu'en alsacien, il est très rare, voire impossible à retrouver des articles sur la médecine, finance, etc. sur internet qui pourrait faciliter l'entrainement d'un modèle pour des domaines spécifiques. Ceci est la fréquence des mots uniquement sur le corpus THÉÂTRE.

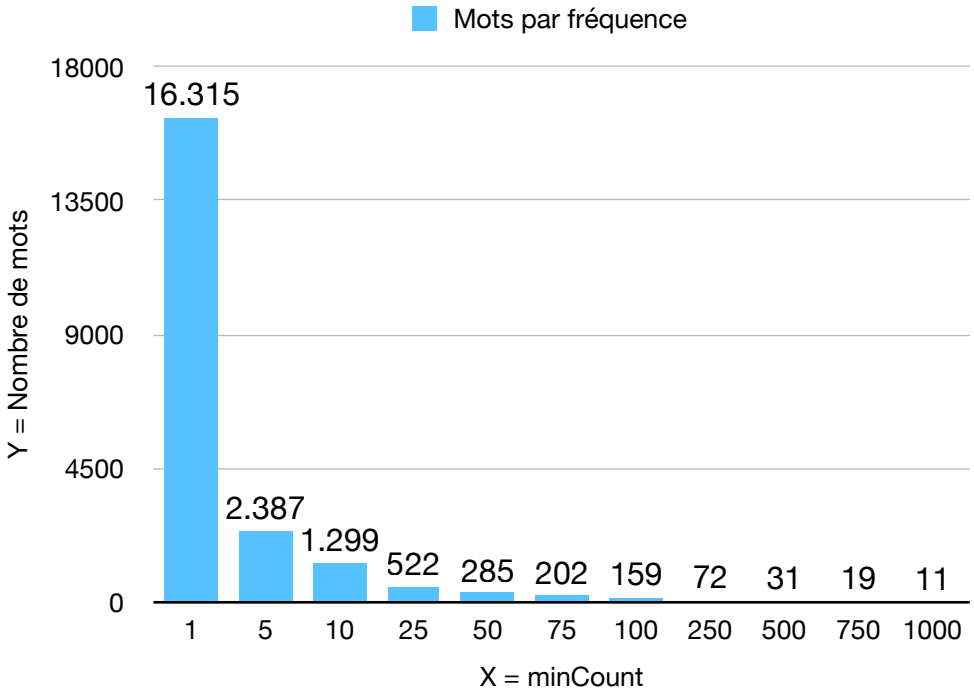

***Figure 4** : Fréquence des mots*

5.4.2 Catégories de mots fréquents

À l'aide de la figure 4, j'ai pu créer des catégories de mots en fonction de leur fréquence. Je les ai classés en 3 catégories:

- **Mots très fréquents** : *ich, wie, der*
- **Mots fréquents** : *Frau, mache, Prinz*
- **Mots peu fréquents** : *Kinigi, Mensch, gfresse*

Ce n'est pas la totalité des mots des catégories uniquement des exemples, j'ai pris 3 mots par catégorie. Pour la catégorie, *mots très fréquents* ce sont des mots qui apparaissent au moins 500 fois dans le texte. Pour *mots fréquents* ils apparaissent au moins 50 fois et pour les *mots peu fréquents* ils apparaissent au moins 5 fois. J'ai fait un

mélange de verbes, pronoms, adverbes/conjonction, articles, noms et une variation orthographique.

5.4.3 Scripts d'évaluation

J'ai créé un script d'évaluation avec un système de points pour les différents corpus, car on ne trouve aucun Benchmark pour la langue alsacienne donc aucune possibilité d'évaluer un modèle qui est entrainé pour cette langue. Ce script d'évaluation sera utilisé sur FastText et Word2Vec. Les modèles vont être testés sur les **analogies** et les 10 **plus proches voisins**. Une analogie que j'ai utilisée ressemble par exemple à ceci *model.get_analogies("König", "Mann", "Frau")* (König = Roi, Mann = Homme, Frau = Femme). Le modèle fait un calcul en faisant Roi-Homme+Femme. Le modèle donnera un résultat et l'on espère retrouver Reine. Chaque bon résultat donne 1 point. Ce calcul est faisable car chaque mot est constitué d'un vecteur avec des nombres réels donc il est possible de faire des opérations mathématiques. Pour cet exemple, les mots homme, femme et roi sont dans les corpus, mais j'ai aussi créé des analogies où 1 mot voir 2 mots ne sont pas dans les corpus. Ceci pour vérifier la robustesse du modèle. (Mundra & Socher, 2016)

Pour la fonctionnalité des 10 plus proches voisins, ils sont sélectionnés en utilisant les mots des trois catégories mentionnées ci-dessus. Un exemple que j'ai utilisé est : *model.get_nearest_neighbors('Frau').* Ceci va me donner une liste avec 10 mots qui sont les plus proches du mot *Frau*. Ceci est de nouveau faisable en utilisant les vecteurs. Un vecteur d'un mot qui est très proche du vecteur du mot *Frau* sera dans la liste des 10 mots. Si un des 10 mots dans la liste est un synonyme ou une variation graphique au mot *Frau*, je donnais 1 point car ce mot est pertinent. (Doval, Vilares, & Gómez-Rodríguez, 2020)

Les sections *5.4.11* et *5.4.13* sont aussi des scripts d'évaluation.

5.4.4 Evaluation des modèles FastText - Skipgram

L'objectif, c'était en premier de trouver un modèle stable sur le corpus THÉÂTRE et qu'il ait une perte basse sans aller vers un surapprentissage et que le temps d'entrainement soit correct. Ensuite, les modèles ont été testés sur les autres corpus. La perte indique la performance de prédiction du modèle, plus la perte est basse meilleur est le modèle. Pour trouver un modèle stable, il fallait tester plusieurs modèles en modifiant les différents paramètres qu'a Skipgram dans FastText. Il faut savoir qu'en modifiant les différents paramètres on peut avoir des modèles qui prennent plus de temps, des pertes moyennes moins élevées et qui génèrent de plus grands vecteurs.

J'en ai testé une dizaine et sur cette dizaine, j'ai pris les 5 meilleurs. Pour chaque modèle, j'ai relancé l'entrainement 10 fois en utilisant le script d'évaluation. Pour chaque entrainement, j'ai compté les points et à la fin j'ai fait une moyenne sur 10. Ceci me donne donc une moyenne pour chaque catégorie.

5.4.5 Résultats globaux des 5 modèles

La figure 5 représente la moyenne des 10 entrainements sur les 10 plus proches voisins par catégorie. Cette évaluation est faite sur le corpus THÉÂTRE. La barre bleu foncé dans le modèle 1 indique qu'en moyenne 1,6 mot pertinent sur 10 (liste des 10 mots) a été trouvé dans la catégorie peu fréquent.
Le meilleur modèle est le 2 mais le modèle 5 est aussi très bon donc ce sont les deux qui se détachent le plus par rapport aux autres. La figure 5 nous montre aussi que les modèles sont meilleurs pour les catégories 2 et 3. Les mots qui sont dans ces deux catégories ont un meilleur contexte. Les résultats des analogies (modèle 2 a le meilleur résultat) sont assez bas ne dépassant pas 5 sur 10, il existe plusieurs raisons. Par exemple que le mot recherché ne se trouve pas dans le texte ou qu'un des trois mots dans l'analogie ne se trouve pas dans le corpus et donc le résultat n'est pas le mot recherché.

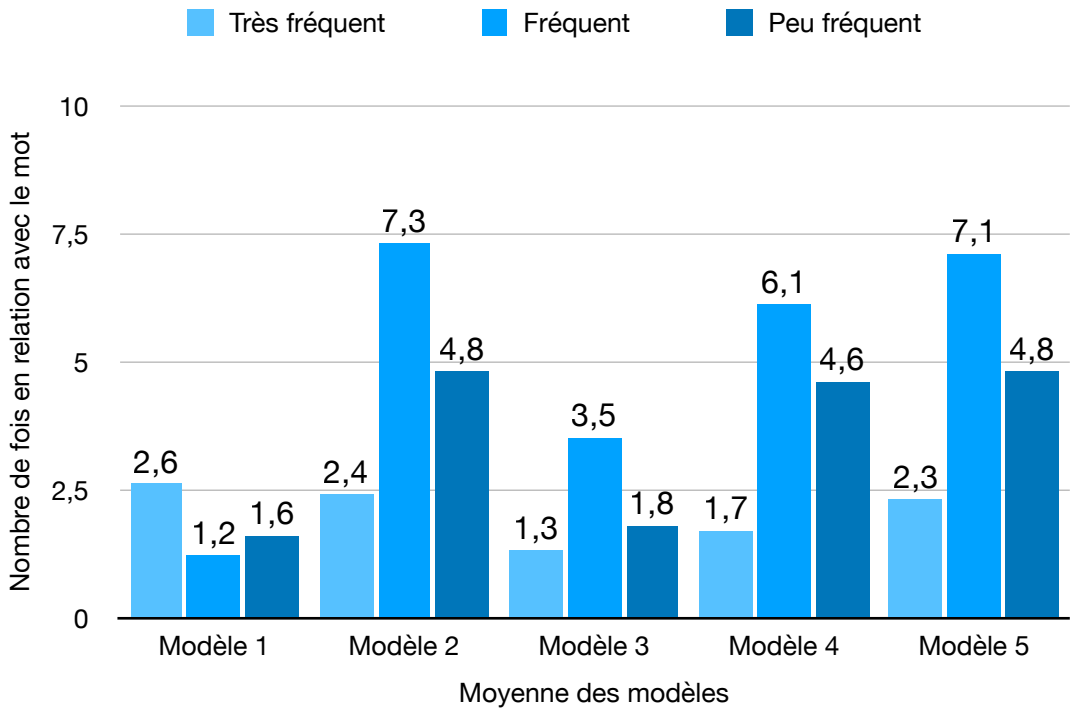

***Figure 5** : Moyenne des 10 plus proches voisins*

Donc pour différencier les deux modèles 2 et 5, j'ai créé deux nouveaux graphiques qui analysent le temps, les epochs et la perte moyenne des modèles. Sur la figure 6a, le modèle 2 prend moins de temps que le modèle 5 et que la perte moyenne est quasi égale. Ceci montre aussi que si le corpus est beaucoup plus grand, le modèle 2 prendra beaucoup moins de temps que le 5.

Le deuxième graphique sur la figure 6b, nous indique que les modèles sur notre corpus THÉÂTRE se stabilisent aux alentours de 50 epochs. Il n'y a pas d'utilité d'ajouter des epochs en plus, car cela ajoute du temps et on peut avoir un surapprentissage. Le modèle 2 est le plus performant pour notre corpus THÉÂTRE.

Les caractéristiques du modèle 2 :
fasttext.train_unsupervised(fichier, 'skipgram', dim = 300, ws = 5, lr = 0.1, epoch = 50, minCount = 1, minn = 2, maxn = 8)

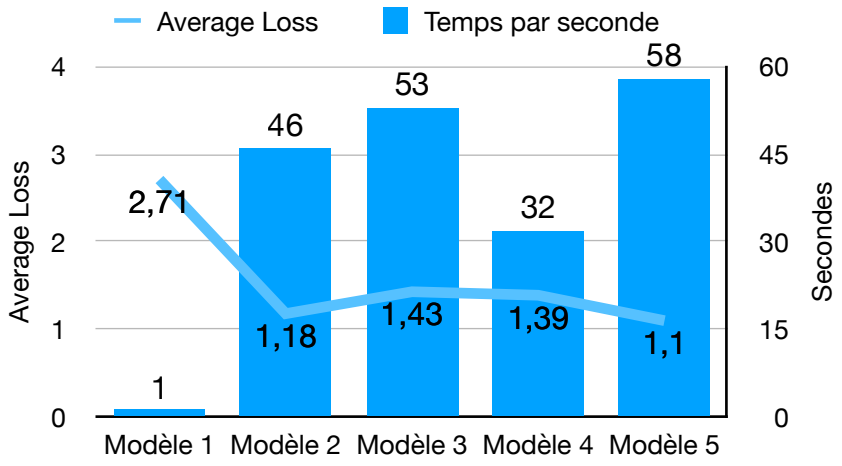

Figure 6a *: Perte et temps par seconde*

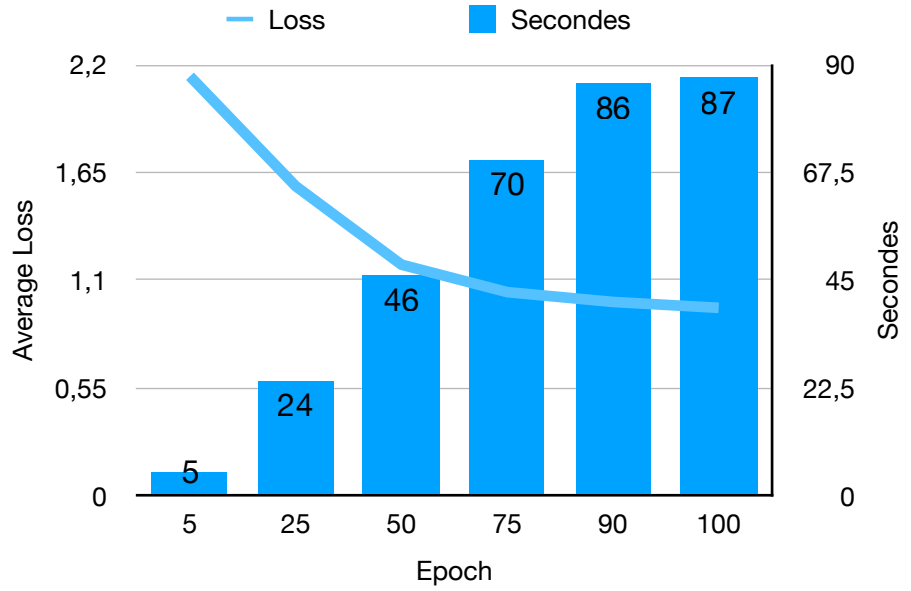

Figure 6b *: Perte et temps par rapport aux epochs*

5.4.6 FastText - CBOW vs Skipgram:

Sur la figure 7, ce sont les résultats sur le corpus THÉÂTRE entre l'architecture Cbow et Skipgram sur le même script d'évaluation. Les résultats sont assez clairs et indiquent qu'il faut utiliser Skipgram dans FastText pour la suite des évaluations. Skipgram a de meilleurs résultats dans les 3 catégories, mais aussi dans les analogies.

Comme mentionné au point *5.3.2 FastText*, Skipgram donne de meilleurs résultats car dans notre cas on travaille avec des petits corpus et il vaut mieux apprendre à prédire un mot grâce à un mot proche (Skipgram) que de prédire le mot en fonction de son contexte (Cbow). Skipgram sait aussi bien représenter les mots qui sont rares donc des mots qui ne sont pas fréquents.

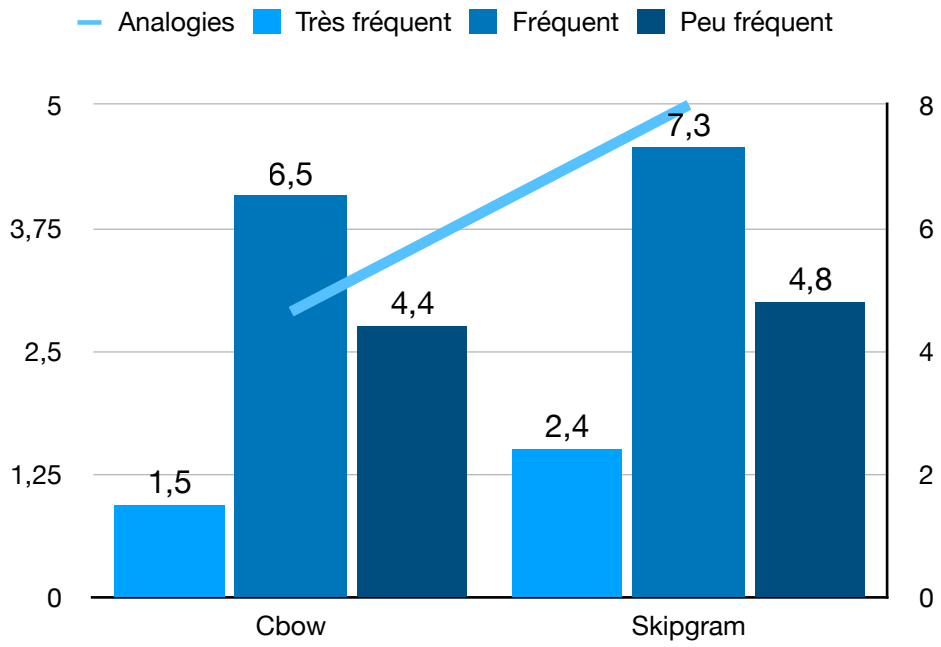

Figure 7 *: Moyenne pour Cbow et Skipgram*

5.4.7 Analyse du nouveau corpus alémanique OSCAR et évaluation:

J'ai ajouté le corpus OSCAR au corpus THÉÂTRE et j'ai testé plusieurs nouveaux modèles tout en gardant le modèle 2 qui a été élu meilleur modèle sur le corpus THÉÂTRE. Les modèles étaient évalués sur le même script d'évaluation. Sans surprise, le modèle 2 reste quand même le meilleur. Sur la figure 8, on peut voir l'évaluation sur le corpus THÉÂTRE et sur le corpus THÉÂTRE + OSCAR. On remarque donc que le nouveau corpus qui a été ajouté aide le modèle à mieux prédire. Ce corpus est dans notre cas une aide et aussi de bonne qualité pour notre modèle. Ceci montre donc qu'on a trouvé un modèle assez stable.

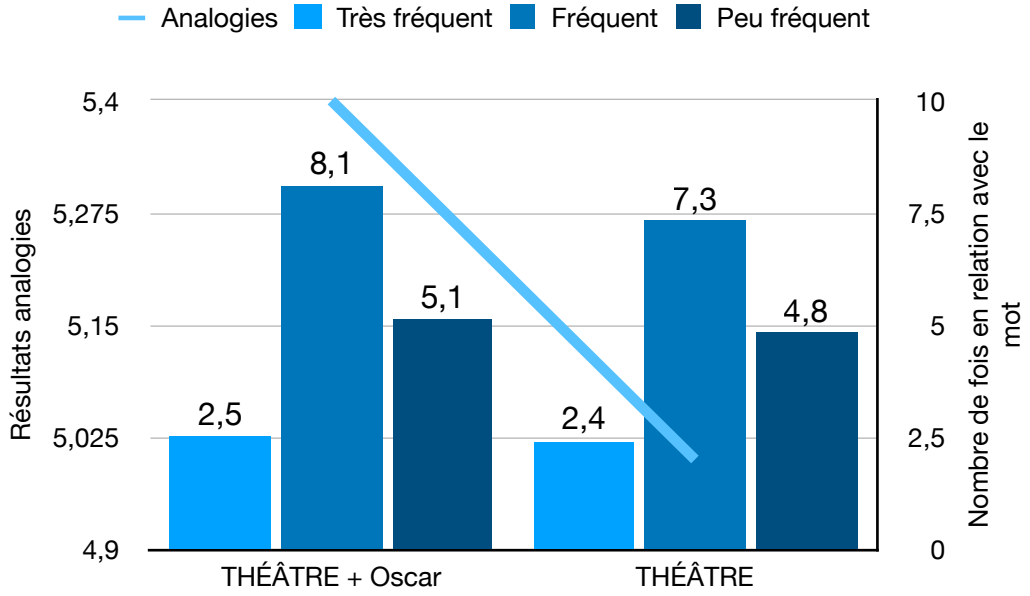

Figure 8 *: Comparaison des corpus*

5.4.8 PCA & t-SNE:

Voici une visualisation en *PCA (Principal Component Analysis)* mais aussi possible en t-*SNE*. PCA est une méthode qui permet de réduire la dimension pour la visualisation des données tout en gardant le maximum d'informations. Ceci permet de mieux les analyser et visualiser. t-SNE (*t-distributed stochastic neighbor embedding*) est aussi une méthode qui permet de réduire les dimensions. La différence c'est que t-SNE préserve que les similarités locales alors que PCA préserve la structure globale des données.

C'est un PCA du modèle 2 de FastText sur le corpus THÉÂTRE + OSCAR. La figure 9 affiche *37 mots prédéfinis* par moi-même. Ce sont des mots qui sont soit proches ou soit dans un même contexte.

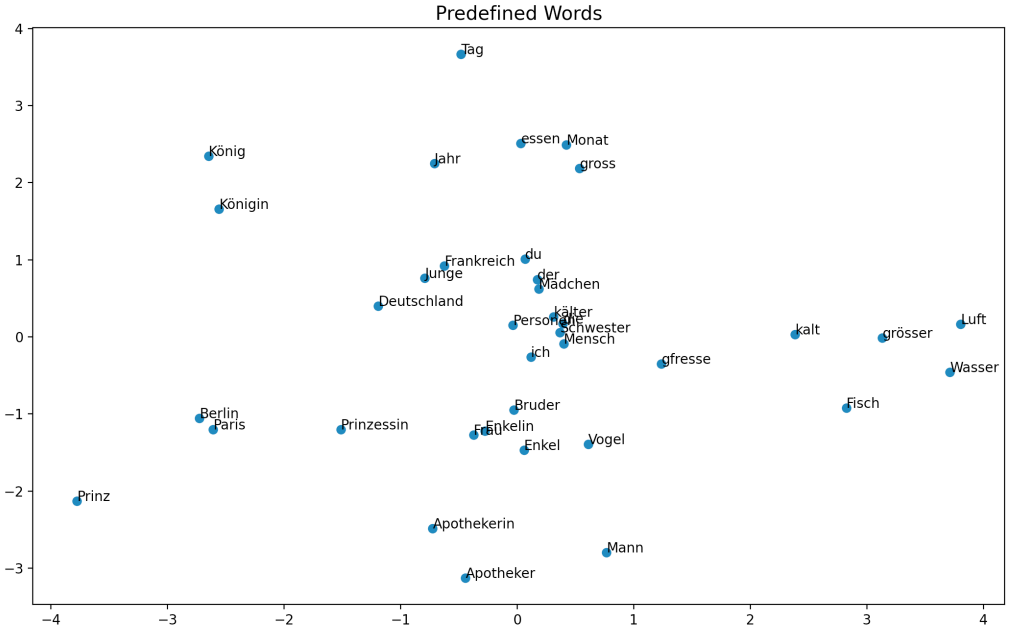

Figure 9 *: Mots prédéfinis*

5.4.9 Évaluation des modèles Word2Vec et Word2Vec avec Magnitude

L'objectif, c'était de trouver un modèle stable pour Word2Vec sur nos corpus THÉÂTRE + OSCAR et de comparer les résultats de W2V avec ceux de FastText. Pour en trouver un, j'ai fait les mêmes évaluations qu'avec FastText sur plusieurs modèles. Pour le prétraitement des données, j'ai mis chaque mot en minuscule, car Word2Vec donne pour chaque mot un différent vecteur. Par exemple, pour lui les mots maison, Maison et MAISON ne sont pas identiques et donneront des vecteurs différents. J'ai donc testé plusieurs modèles avec différents paramètres et j'ai pris les 3 meilleurs. La figure 10 représente les 3 modèles Word2Vec avec et sans l'utilisation de Magnitude.
On peut voir que chaque modèle sans utiliser Magnitude n'est pas bon. C'est parce qu'il n'utilise pas les n-grammes et qu'il ignore les mots non connus. Dans notre cas, ces fonctionnalités sont très importantes, car il y a beaucoup de variations graphiques en alsacien. Dans la plupart des cas, Magnitude sert à mieux représenter les vecteurs de W2V mais malheureusement les résultats ne sont pas bons. Pourquoi ? Car les plongements de mots ne sont déjà pas bien représentés par Word2Vec donc Magnitude ne pourra pas utiliser ces fonctionnalités en plus pour mieux représenter les vecteurs. Il n'est donc pas possible de trouver un modèle stable Word2Vec pour nos corpus

alsaciens. J'utiliserai donc à partir de maintenant FastText plus précisément le modèle 2 (*voir 5.4.5 Résultats globaux des 5 modèles*) pour la suite.

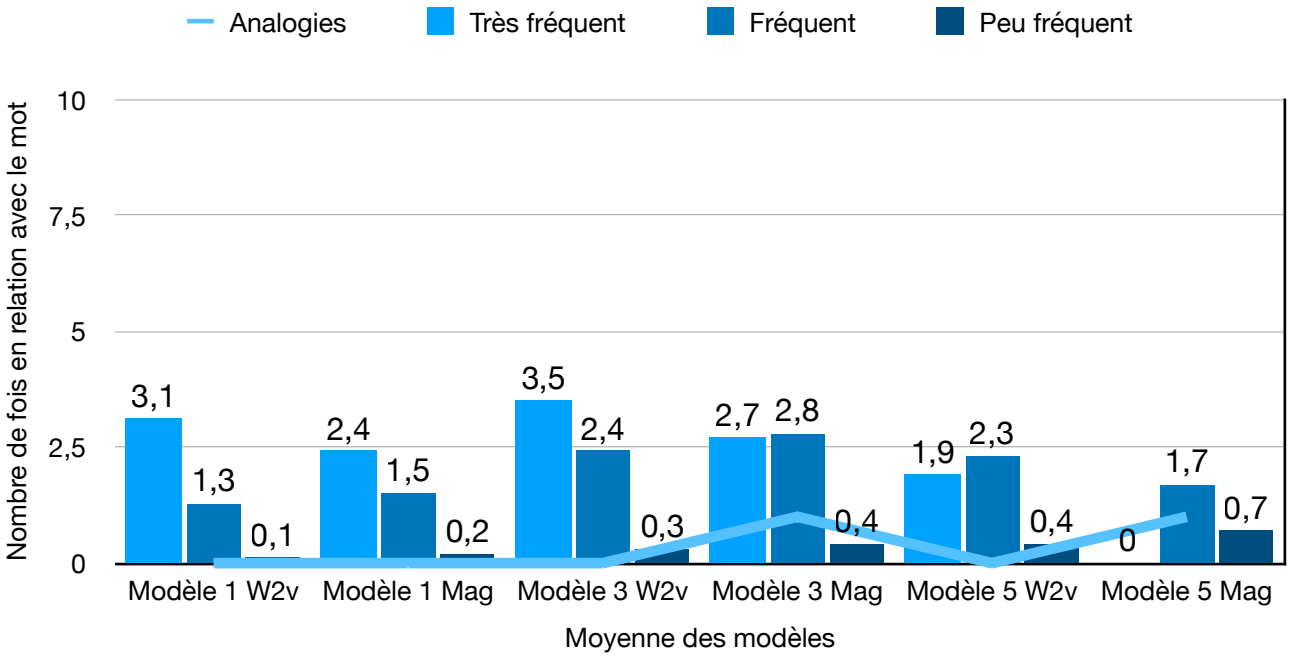

Figure 10 *: Résultats de 3 modèles W2v avec/sans Magnitude*

5.4.10 Ajout d'un nouveau corpus alsacien DIVERS

Ce nouveau fichier est un corpus alsacien contemporain envoyé par l'enseignante Delphine BERNHARD (*voir 5.2.1 Corpus*). Au début, j'ai évalué le modèle 2 de FastText uniquement sur ce corpus pour voir si le modèle était toujours stable et si le corpus était de bonne qualité. J'ai pu constater qu'en ajoutant ce corpus avec les autres corpus déjà présents, les résultats sont toujours bons donc le modèle est toujours stable et le corpus en lui-même est de bonne qualité. Donc pour les prochaines évaluations, on a un fichier qui contiendra les corpus THÉÂTRE + OSCAR + DIVERS.

5.4.11 Lexiques bilingues

Le lexique bilingue a été conçu par l'enseignante Delphine BERNHARD. Ce fichier m'aide à évaluer mon modèle. On retrouve des mots en français et la traduction alsacienne. Ces mots viennent d'un lexique/dictionnaire donc il est possible qu'on ne retrouve pas chaque mot dans nos corpus alsaciens. Avec ce fichier, on peut essayer de trouver de nouvelles analogies en français et les traduire en alsacien et voir si le modèle les trouve et de même pour les 10 plus proches voisins qui peuvent donner les synonymes ou des mots proches avec des variations graphiques.
Voici les nouveaux mots pour l'évaluation des plus proches voisins avec leur traduction :

- *Üsdruck = expressions*
- *Blätter = feuille*
- *Wàppa = blasons*
- *ànbiete = offrir*
- *Pàpier = papier*
- *singe = chanter*
- *Schüalmeischter = instituteur*
- *Storich = cigogne*
- *Donnerschdi = jeudi*
- *spiele = jouer*
- *Gugelhupf = kougelhopf*
- *Vàdder = père*
- *schneien = neiger*

5.4.12 Comparaison des corpus

Il était important de comparer les différents corpus à notre disposition afin de savoir lesquels nous permettent d'obtenir les meilleurs résultats. Dans notre cas, il est intéressant de voir si en gardant uniquement les corpus THÉÂTRE + DIVERS (pas beaucoup de ressources) on aura de meilleurs résultats qu'en ajoutant le corpus OSCAR. Cet ajout nous donnerait plus de ressources donc une plus grosse quantité de données. J'ai fait 3 évaluations, une évaluation avec uniquement les corpus alsaciens THÉÂTRE + DIVERS, une deuxième évaluation uniquement sur le corpus alémanique OSCAR et une dernière en mélangeant les corpus.
Dans notre cas, la question qui se pose est : Est-il préférable d'utiliser uniquement des textes alsaciens qui a comme conséquence qu'il y ait moins de ressources ou vaut-il mieux faire un mélange pour avoir plus de ressources à notre disposition ?

Il y a plusieurs proximités qui nous intéressent, la proximité sémantique, mais aussi la proximité graphique. Les évaluations sont toujours faites avec le modèle 2.

La question de savoir si les mots ciblés par l'évaluation se trouvent dans le corpus ou non est secondaire, car on cherche également à évaluer la pertinence des plongements obtenus pour les formes absentes des corpus. C'est une des limites de Word2Vec puisqu'il n'est pas capable de retrouver les mots qui ne sont pas dans le corpus. Mais le fait que FastText sache reconnaitre une variante graphique alors même qu'elle ne se trouve pas dans le corpus, est intéressant. Cela veut dire si l'on utilise sur des nouvelles données il est plus robuste. On cherche donc de se mettre dans une situation réaliste où l'on a des formes inconnues.

La figure 11a contient uniquement les mots de « fréquence de mots » (*voir 5.4.2 Catégorie de mots fréquents*). La figure 11b compare uniquement les résultats des mots du lexique bilingue. Les résultats sont bons quand il y a uniquement des corpus alsaciens (THÉÂTRE+DIVERS) et un mélange des 3. La figure 11b donne une information en plus qui est très importante. La barre bleu claire de THÉÂTRE + DIVERS est de 5,5 et celle de OSCAR + THÉÂTRE + DIVERS est de 5,8. Ce résultat est presque identique mais celui de la barre bleu foncé, il y a une grande différence de +1,1. Ceci veut dire qu'il vaut mieux utiliser un corpus avec le même dialecte qu'un corpus avec des dialectes mélangés.

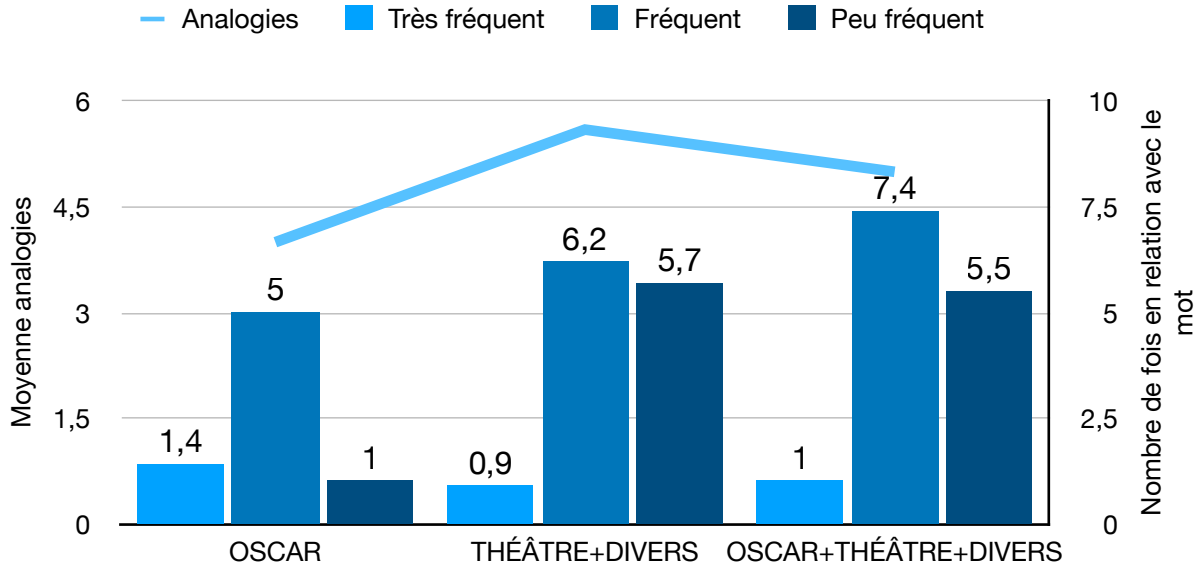

Figure 11a *: Fréquence de mots sur les différents corpus*

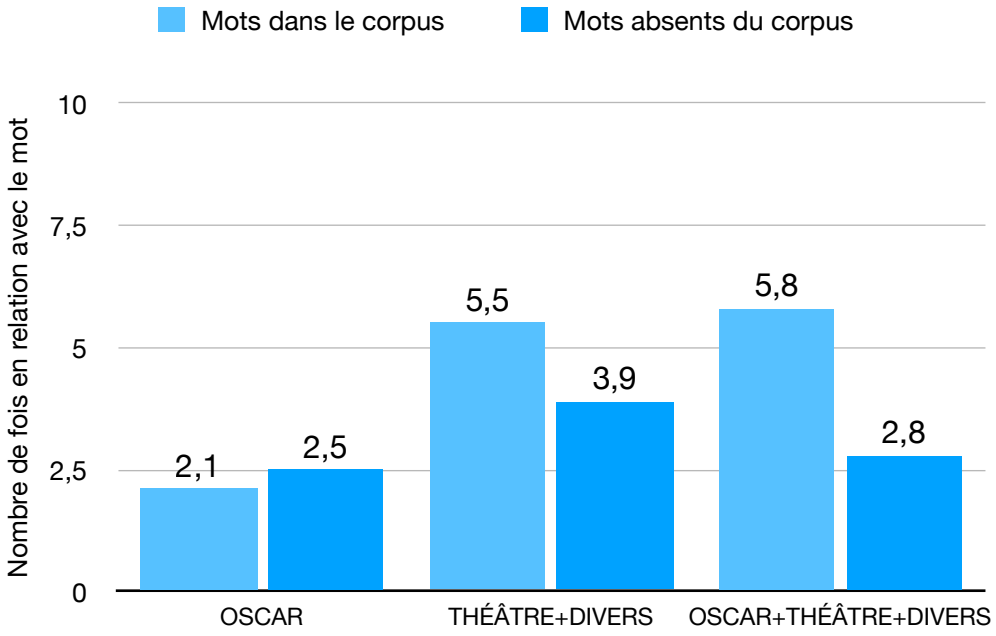

Figure 11b *: Résultats lexique bilingues sur les différents corpus*

5.4.13 Analyse approfondie des mots sur les 3 corpus

Cette étape consistait à voir si le modèle arrivait à trouver des synonymes et non seulement des variations graphiques, car ceci il sait déjà bien faire. Pour faire ceci, j'évalue le modèle sur 30 mots différents en utilisant un fichier Excel créé par l'enseignante Delphine BERNHARD et en utilisant pour l'évaluation deux mesures différentes. Les évaluations sont faites sur les 3 corpus (OSCAR + THÉÂTRE + DIVERS).
La première mesure est la similarité cosinus entre 2 mots. La similarité cosinus est une technique largement utilisée pour la similarité des textes. La similarité cosinus renvoie un score compris entre 0 et 1. 1 correspond à la similarité exacte et 0 à la similarité nulle. Dans la pratique, si le score de similarité est supérieur à 0,5, il est probable qu'il y a une certaine similarité. Mais, cela peut varier en fonction de l'application et du cas d'utilisation, mais aussi du contexte des mots.

La deuxième mesure est la fonctionnalité 10 mots les plus proches. Pour chaque mot, le modèle donne une liste de 10 mots qui représente les mots les plus proches du mot extrait du fichier Excel. J'ai décidé de répartir les 10 mots de la liste dans 6 catégories différentes. Les catégories sont : Synonymes morphologiquement liés, Autres synonymes, Variantes graphiques et flexionnelles, Même famille morphologique, Autre relation sémantique, Aucune relation.

- *ànbiete = offrir ; proposer*
- *Anfaenger = débutant ; inexpérimenté*
- *Ärdbeer = fraise*
- *Bìrkebàüm = bouleau*
- *Blätter = feuillage*
- *dief = profond*
- *Donnerschdi = jeudi*
- *Gugelhupf = kougelhopf ; kougelhof*
- *Kornflocke = flocons de céréales*
- *Kreis = cercle*
- *Metzjerei = boucherie*
- *Noochmiddàà = après-midi*
- *Pàpier = papier*
- *rajerisch = pluvieux*
- *schneien = neiger*
- *Schüalmeischter = instituteur*

- *Hecke = arbres/arbustes*
- *kìrscherot = rouge cerise*
- *Stiefmüeter = belle-mère*
- *Storich = cigogne*
- *Strüss = bouquet*
- *Üsdruck = expression*
- *üssblose = souffler*
- *singe = chanter*
- *spiele = jouer*
- *Vàdder = père*
- *wachselhàft = variable*
- *Wàppa = blason*
- *wärme = chauffer*
- *weich = mou*

Chaque mot ci-dessus a 5 catégories et chaque catégorie contient une liste de mots. La figure 12 affiche le score moyen de la mesure Cosinus pour chaque catégorie sur une évaluation.

Catégorie	Score moyen sur 1
Synonymes morphologiquement liés	0,12
Autres synonymes	0,12
Variantes graphiques et flexionnelles	0,62
Même famille morphologique	0,44
Autre relation sémantique	0,01

***Figure 12** : Score moyen de la mesure Cosinus*

La figure 13 affiche pour chaque mot par catégorie un score. Ce score va de 0 à 10 pour chaque catégorie. Ceci est la deuxième mesure avec les 10 plus proches voisins. Par exemple pour le mot *ànbiete*, une liste des 10 plus proches voisins de ce mot est générée. Ensuite avec l'aide du fichier Excel je compare si un mot de la liste se trouve dans le fichier. Si c'est le cas, je donne un point à la catégorie correspondante et s'il ne trouve pas le mot dans le fichier, j'incrémente d'un point la catégorie 6 *Aucune relation*.

Dans la liste des 10 mots il y a autant de mots pertinents qu'il y a des points dans les catégories 1, 2, 3, 4 et 5. Si ce n'est pas un mot pertinent c'est qu'il n'a aucune relation donc appartient à la catégorie 6. Le mot *ànbiete* à la figure 13 indique que dans 4 catégories on a un score de 0 mais que dans la catégorie 3 il y a 2 points. Ceci veut dire que dans la liste des 10 plus proches voisins, il y a 2 mots qui sont pertinents (relation avec le mot *ànbiete*). Donc, il y a 8 mots qui ne sont pas pertinents. Cette évaluation a été faite sur les 3 corpus.

Les catégories de la figure 13 sont :
- Cat. 1 = Synonymes morphologiquement liés
- Cat. 2 = Autres synonymes
- Cat. 3 = Variantes graphiques et flexionnelles
- Cat. 4 = Même famille morphologique
- Cat. 5 = Autre relation sémantique
- Cat. 6 = Aucune relation

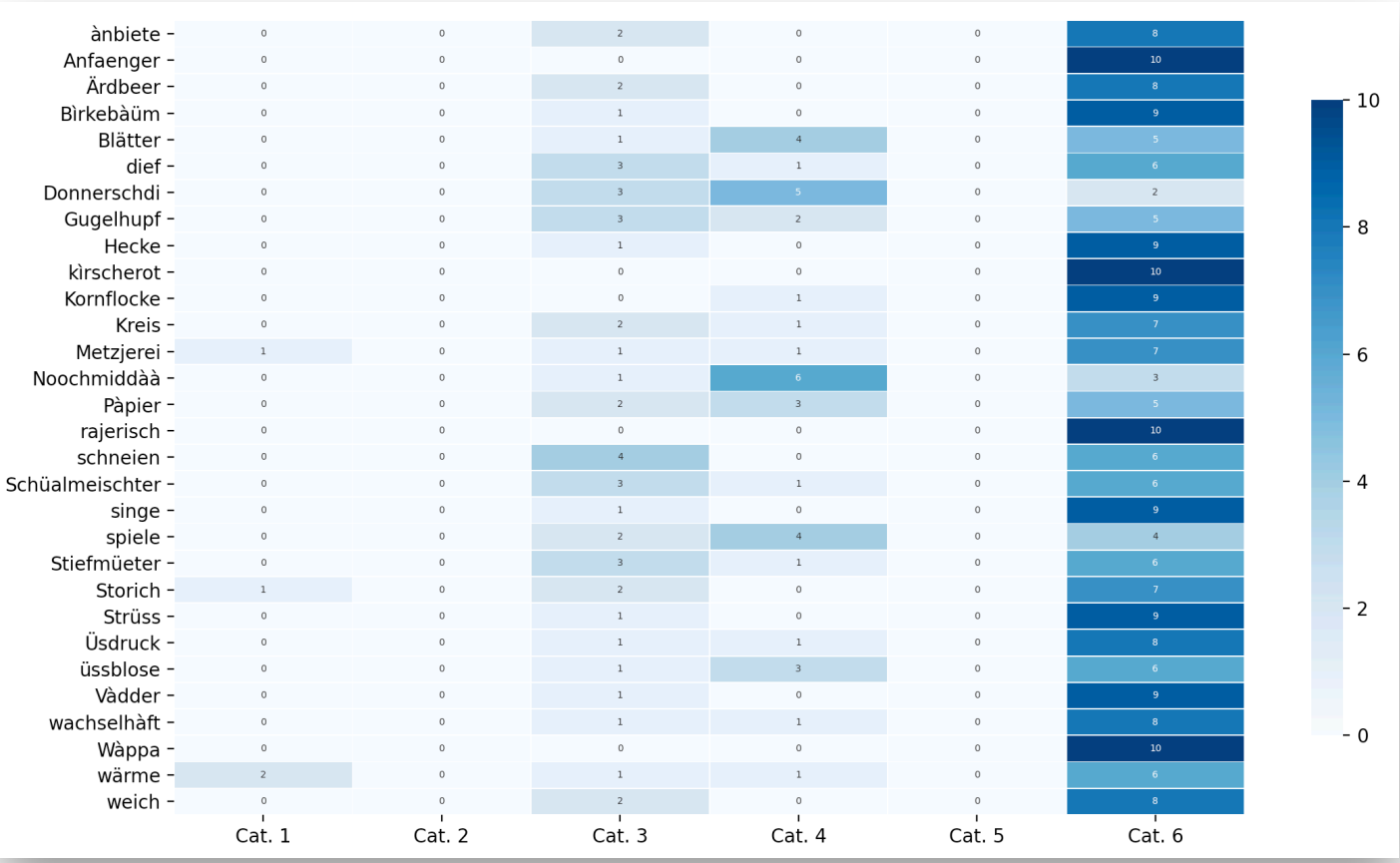

Figure 13 *: Heatmap, résultats de la mesure des 10 plus proches voisins pour les 30 mots*

La figure 14 affiche le score moyen global de 5 évaluations. Cette évaluation est faite une fois sur les corpus mélangés et une fois que sur les corpus alsaciens. On remarque que les résultats sont meilleurs pour les corpus alsaciens. Ceci confirme donc qu'il vaut mieux utiliser un corpus qui a le même dialecte et que le modèle s'en sort bien avec les variations graphiques pour l'alsacien. Ce qui n'est pas le cas pour les synonymes car premièrement FastText n'est pas vraiment fait pour être utilisé sur les synonymes et deuxièmement nous avons un très petit corpus.

Catégorie	Score moyen sur 300 (OSCAR+THÉÂTRE+DIVERS)	Score moyen sur 300 (THÉÂTRE+DIVERS)
Synonymes morphologiquement liés	5,4	7,4
Autres synonymes	0	1,4
Variantes graphiques et flexionnelles	45,2	54
Même famille morphologique	36	57,2
Autre relation sémantique	0	0
Mots avec aucune des relations précédentes	213,4	180,6
Score moyen - (nombre de mots pertinents)	86,6	119,2

Figure 14 *: Score moyen global de la mesure des 10 mots les plus proches*

6. Résultats & Choix effectués

Tout au long du rapport, on peut constater avec les différentes figures que l'architecture à choisir est FastText pour les plongements de mots en alsacien et que Skipgram est meilleur que CBOW. On a pu voir que FastText avec des paramètres définis est meilleur que Word2Vec et que même en utilisant la librairie Magnitude sur Word2Vec les résultats sont de loin pas si bons qu'avec FastText. Ensuite, en rajoutant un corpus alémanique et un corpus alsacien DIVERS, on peut constater que le modèle est toujours stable et que les résultats se sont même un peu améliorés. Ceci nous dit donc que les nouveaux corpus améliorent l'entrainement du modèle et sont de bonne qualité. Tout ceci a été évalué en utilisant mes scripts d'évaluation.

Dans ces scripts d'évaluation, on peut retrouver le lexique bilingue, les fonctionnalités de FastText (10 mots les plus proches, analogies et la similarité cosinus) et la liste d'évaluation avec le fichier Excel. En faisant une analyse plus approfondie, j'ai pu constater qu'il vaut mieux utiliser un corpus de taille petite avec le bon dialecte qu'un large corpus avec différents dialectes. En regardant les résultats avec la similarité cosinus et les mots les plus proches, on suppose que notre modèle est toujours stable et qu'il s'en sort bien avec certaines catégories comme les variations graphiques. On se rend compte que le modèle est aussi robuste car pour certains mots qui ne sont pas dans le vocabulaire il n'a pas de souci à les évaluer. Les choix à retenir sont FastText avec un corpus (petit ou grand) mais qui contient uniquement le dialecte alsacien.

7. Perspectives

Même si je n'ai pas abordé les plongements cross-lingues, cette étude nous montre le rôle important du corpus et de la méthode choisie. Cette étude pourra servir de base à une analyse cross-lingue : des plongements de mauvaise qualité en monolingue ne pourront avoir qu'une répercussion négative en cross-lingue. C'est important de bien comprendre comment on peut avoir les meilleurs plongements de mots. La suite sera donc d'étudier les plongements de mots translingues pour les dialectes alsaciens en se basant sur ces résultats monolingues.

8. Conclusion

Les dialectes alsaciens sont des langues parlées non codifiées cela pose une grosse difficulté, mais j'ai quand même réussi à trouver un modèle stable avec FastText. FastText est donc l'outil à utiliser car il utilise la fonctionnalité n-grammes qui est très importante pour l'alsacien à cause de beaucoup de variations graphiques et qu'il génère un vecteur pour les mots qui ne sont pas dans le corpus.

Après une analyse plus approfondie, j'ai pu répondre à cette question: "Est-il préférable d'utiliser uniquement des textes alsaciens qui a comme conséquence qu'il y a moins de ressources ou vaut-il mieux faire un mélange pour avoir plus de ressources à notre disposition ?"
La réponse, c'est que la tendance qui apparait c'est qu'un corpus gros n'est pas toujours le meilleur et surtout c'est que même avec un petit corpus FastText arrive à trouver des mots pertinents qui sont proches graphiquement et morphologiquement du mot. Les résultats à la figure 11b et et 14 indiquent qu'il vaut mieux utiliser des corpus qui contiennent uniquement le même dialecte alsacien car le modèle performe mieux sur ce

type de corpus. Pendant le projet les résultats étaient meilleurs avec les corpus mélangés parce que la plupart des mots qu'on cherchait se trouvaient dans les corpus ce qui ne représente pas tout à fait la réalité. Ceci montre que ce n'est pas toujours la quantité des données qui compte, mais la qualité.

Pour conclure, dans notre cas il vaut mieux avoir un corpus plus petit, mais de la famille du dialecte qui nous intéresse plutôt qu'un gros corpus avec des dialectes mélangés.

9. Remerciement

Je tiens à remercier mon encadrante du TER Madame Delphine BERNHARD, enseignante-chercheuse à l'Université de Strasbourg, de m'avoir attribué ce sujet pour le TER. Sa disponibilité, sa gentillesse, ses compétences, et ses précieuses directives m'ont été d'une aide inestimable.

10. Bibliographie

BOJANOWSKI Piotr, GRAVE Edouard, JOULIN Armand & MIKOLOV Tomás (2017). Enriching Word Vectors with Subword Information. In : *Transactions of the Association for Computational Linguistics*, 5, 135-146. https://doi.org/10.1162/tacl_a_00051 (cf. p. 7)

DOVAL Yerai, VILARES Jesús & GÓMEZ-RODRÍGUEZ Carlos (2020). Towards Robust Word Embeddings for Noisy Texts. In : *Applied Sciences*, 10(19), 5-6. https://doi.org/10.3390/app10196893 (cf. p. 9)

MIKOLOV Tomás, CHEN Kai, CORRADO Greg & DEAN Jeffrey (2013). Efficient Estimation of Word Representations in Vector Space. In : Proceedings of the International Conference on Learning Representations, 1-12. https://arxiv.org/abs/1301.3781 (cf. p. 7)

MUNDRA Rohit & SOCHER Richard (2016). CS 224D: Deep Learning for NLP. In : *Lecture Notes: Part II,* 1-11. https://www.semanticscholar.org/paper/CS-224D (cf. p. 9)

NGUYEN Dong & GRIEVE Jack (2020). Do Word Embeddings Capture Spelling Variation ? In : *Proceedings of the 28th International Conference on Computational Linguistics*, 870-881. https://doi.org/10.18653/v1/2020.coling-main.75 (cf. p. 2)

PATEL Ajay, SANDS Alexander, CALLISON-BURCH Chris & APIDIANAKI Marianna (2018). Magnitude : A Fast, Efficient Universal Vector Embedding Utility Package. In : *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing : System Demonstrations*, 120-126. https://doi.org/10.18653/v1/d18-2021 (cf. p. 7)

RUDER Sebastian, VULIĆ Ivan & SØGAARD Anders (2019). A Survey of Cross-lingual Word Embedding Models. In : *Journal of Artificial Intelligence Research,* 65, 569-576. https://doi.org/10.1613/jair.1.11640 (cf. p. 6)