Analyse statistique et empirique des modèles de *Word-Embedding* sur Twitter

Kim Antunez, Romain Lesauvage, Alain Quartier-la-Tente
sous l'encadrement de Benjamin Muller (Inria)

Table des matières

Introduction

Grâce à l'évolution des méthodes d'apprentissage profond (*Deep Learning*), l'appréhension du langage naturel est aujourd'hui devenue une discipline à part entière (*Natural Language Processing*). Ce succès s'explique en partie grâce à l'émergence de techniques non supervisées d'apprentissage de représentation de structures linguistiques. Les méthodes de *word embedding* (« plongement lexical » en français) permettent de représenter chaque mot d'un dictionnaire par un vecteur de nombres réels afin que les mots qui apparaissent dans des contextes similaires possèdent des vecteurs correspondants qui sont relativement proches (au sens d'une distance définie). Les modèles *Word2Vec*, développés par une équipe de recherche chez Google (Mikolov *et al* (2013a)), sont parmi les plus célèbres et sont ceux sur lesquels se concentrera notre projet.

Dans ce projet de statistique appliquée, nous étudierons dans un premier temps en détail et implémenterons le modèle *Word2Vec* (partie 1). Dans un deuxième temps, nous évaluerons la validité du modèle implémenté et l'appliquerons sur une base de données composée de plusieurs millions de tweets publiés en France entre 2013 et 2017 (partie 2). Enfin, nous mobiliserons des techniques d'analyse de sentiments afin de créer des indicateurs qui pourront être comparés aux indicateurs produits dans la statistique publique, en particulier concernant l'opinion des ménages (partie 3).

1 Implémentation du modèle *Word2Vec*

1.1 Le modèle *Word2vec*, un modèle de *word-embedding*

Le *Natural Language Processing* (NLP ou « traitement automatique du langage naturel ») est une branche du *machine learning* visant à analyser, traiter et reproduire le langage humain. Les modèles de NLP *Word2Vec*, développés par une équipe de recherche chez Google (Mikolov *et al* (2013a)), sont parmi les plus célèbres et utilisent le *word-embedding* – plongement lexical en français.

1.1.1 Historique : de la sémantique vectorielle à *Word2Vec*

La « sémantique vectorielle » est née dans les années 1950 [1]. C'est une méthode algébrique de représentation d'un document visant à réaliser des tâches diverses (détecter le plagiat, filtrer des articles...). Il est alors nécessaire de capter de nombreux types de proximités entre mots : les synonymes (automobile / voiture), antonymes (froid / chaud), connotations positives *versus* négatives (heureux / triste), etc.

Un modèle répondant à toutes ces exigences ne peut exister. Pour y répondre au mieux, la sémantique vectorielle puise son inspiration des travaux linguistiques des années 1950 et en particulier de l'« hypothèse de distribution » selon laquelle un mot se définit par son environnement. Dit autrement : les mots qui se produisent dans un contexte identique tendent à avoir des significations similaires [2].

Les premiers modèles sémantiques (comme le *term frequency-inverse document frequency* (TF-IDF)) représentaient les relations entre mots grâce à des très grandes matrices, dites *sparses*, dont les dimensions correspondaient à la taille du vocabulaire (contenant donc beaucoup de 0). Les méthodes

1. L'ouvrage Jurafsky & Martin (2019) permet de retracer avec une grande richesse l'évolution des méthodes de NLP.

2. Comme l'a écrit le linguiste britannique John Rupert Firth en 1957, « Vous connaîtrez un mot par ses fréquentations ».

de *word-embedding* qui sont ensuite apparues ont permis de représenter chaque mot d'un dictionnaire par un vecteur de nombres réels denses (peu de 0) de plus faible dimension (en général entre 50 et 1000). Si la réduction de dimension rend les vecteurs-mots moins facilement interprétables, elle a pour grand avantage de faciliter et d'accélérer les tâches d'apprentissage impliquant ces mots.

Mikolov *et al* (2013a) ont mis en avant en 2013 les méthodes de *word-embedding* à travers la création de *Word2Vec*. Ce modèle de réseaux de neurones [3] à deux couches est rapidement devenu une référence grâce à la grande précision des résultats qu'il permet d'obtenir, pouvant être entraîné en un temps record sur un corpus très volumineux.

1.1.2 *Word2Vec*, un modèle d'apprentissage « auto-supervisé »

En sortie du modèle *Word2Vec*, chaque mot est représenté par un vecteur dont la dimension est fixée par la valeur d'un hyperparamètre. Les mots qui apparaissent dans des contextes similaires (« bonjour » et « salut » par exemple) seront représentés par des vecteurs relativement proches dans l'espace vectoriel de définition de ces vecteurs. Dans la même logique, *Word2Vec* permet également de réaliser des opérations vectorielles, comme dans l'exemple, souvent cité : $\overrightarrow{Paris} - \overrightarrow{France} + \overrightarrow{Italie} = \overrightarrow{Rome}$ qui provient de Mikolov *et al* (2013a).

Deux architectures du modèle *Word2Vec* existent (voir graphique 1) :

- L'approche *Continuous bags of words* dont l'objectif est d'estimer la probabilité d'observer un mot, appelé **« focus »**, sachant le contexte dans lequel il apparaît (i.e. : les mots **voisins** qualifiés de **« contextes »**).
- L'approche *Skip-gram* a un objectif inverse : estimer, pour chaque mot du vocabulaire, la probabilité d'être proche du mot focus. C'est cette approche que nous étudions dans ce projet et dans la suite de ce rapport.

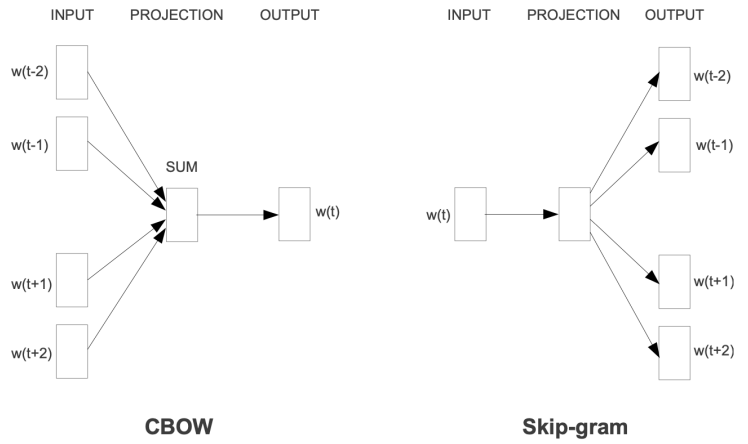

FIGURE 1 – Architecture des modèles Continuous bags of words (CBOW) et Skip-gram.
Source : Mikolov et al (2013a)

Pour transformer chaque mot en un vecteur, au lieu de simplement compter les fréquences d'apparition des mots contextes voisins d'un mot focus [4], nous entraînons un réseau de neurones sur une tâche

3. C'est l'article Bengio *et al* (2003) qui a introduit dix ans avant *Word2Vec* le premier modèle d'apprentissage de représentation de vecteurs-mots à partir d'un réseau de neurone simple.
4. Comme dans les premiers modèles sémantiques dits *sparses*.

annexe : on construit un classifieur dont la tâche de prédiction est binaire pour chacun des mots du vocabulaire et répond à la question (dans le cas Skip-Gram) « Est-ce que ce mot contexte est susceptible d'être voisin du mot focus ? ». La prédiction en elle-même ne nous intéresse pas, c'est plutôt le poids du classifieur en sortie du modèle qui correspondra aux *word-embeddings*.

Les voisins d'un mot focus reposent sur un hyperparamètre : la fenêtre (*window* ou w). Pour $w = p$, les voisins du mot focus sont les p mots précédents et les p mots suivants dans la phrase. Par exemple, dans la phrase :

> “*Le professeur de statistique est strict avec ses élèves.* ”

pour $w = 2$, si le mot focus est « statistique » alors le contexte qui lui est associé est : `[professeur, de, est, strict]` ; si le mot focus est « professeur » alors le contexte qui lui est associé est : `[Le, de, statistique]`.

Pour déterminer les représentations vectorielles des mots, nous entraînons le réseau de neurones en le nourrissant des paires `[focus, contexte]` [5] contenues dans les différentes phrases (ici tweets) du corpus afin qu'il puisse déterminer les probabilités d'apparition d'un mot dans le voisinage d'un autre mot (voir description de l'algorithme en partie 1.2).

Ainsi, la grande force du modèle d'apprentissage *Word2Vec* est qu'il est « auto-supervisé ». En effet, comme nous avons vu plus haut, le corpus est considéré comme une donnée d'entraînement implicitement supervisée, ce qui nous évite d'avoir à mobiliser des corpus annexes annotés.

1.2 L'algorithme Skip-Gram

L'objectif de cette partie est de décrire le fonctionnement de l'approche Skip-gram.

Dans la suite de ce projet nous noterons n la taille du vocabulaire (i.e. : le nombre de mots différents) et dim la dimension retenue pour les *word-embeddings*. Comme décrit dans la partie 1.1.2, l'approche *Skip-gram* peut être vue comme un réseau de neurones à deux couches avec :

— En entrée une matrice W_e de taille $n \times dim$;
— En sortie une matrice W_S de taille $n \times dim$.

Ces deux matrices sont initialisées en générant des lois normale $\mathcal{N}(0, 1)$. Elles sont ensuite mises à jour, grâce aux couples `[focus, contexte]` construits à partir du contexte (voir partie 1.2.1), par un algorithme de descente de gradient. À la fin de l'algorithme, ce sont ces matrices qui donneront la représentation vectorielle des mots du vocabulaire. Ainsi, la ligne i de la matrice $W = \frac{W_e + W_s}{2}$ donnera la représentation du $i^{\text{ème}}$ mot du vocabulaire en dimension dim.

1.2.1 Construction de la base d'entraînement

Peu de traitements sont effectués sur la base initiale : nous mettons tout en minuscule, remplaçons les ponctuations par des espaces, mais laissons tous les chiffres et les accents. Chaque phrase [6] est ensuite *tokénisée* par la chaîne de caractère correspondant à un espace `" "` : on considère qu'il y a autant de mots de que chaînes de caractères séparées par un espace [7]. Par exemple, la phrase :

5. Dans notre exemple : `[statistique, professeur], [statistique, de]`...
6. Dans notre cas une phrase correspond à un tweet, même si ce tweet peut être composé de plusieurs phrases.
7. Les mots composés sont donc considérés comme plusieurs mots distincts.

“*Que pensez-vous de CE projet?(i.e. : qu'avez-vous retenu en 10min ?)* ”

est décomposée en 14 mots `[que, pensez, vous, de, ce, projet, i, e, qu, avez, vous, retenu, en, 10min]`.

Comme décrit dans la partie 1.1.2, les couples `[focus, contexte]` dépendent d'un hyperparamètre : la fenêtre w. Pour éviter que les mots trop fréquents, souvent peu informatifs (comme les pronoms personnels), soient sur-entraînés, deux traitements sont effectués :

1. Pour chaque phrase on effectue un sous-échantillonnage (*subsampling*). Pour chaque mot w_i on note $z(w_i)$ la proportion d'apparition de ce mot, c'est-à-dire le rapport entre le nombre de fois que ce mot apparaît et le nombre total de mots. La probabilité de garder un mot le mot w_i est donnée par :
$$\mathbb{P}(w_i) = \min\left\{\left(\sqrt{\frac{z(w_i)}{q}} + 1\right) \times \frac{q}{z(w_i)}, 1\right\}$$
Le paramètre q appelé « sample » – échantillonnage – contrôle le nombre de mots sous-échantillonnés (plus il est grand, plus la probabilité de garder le mot w_i est grande). Si q vaut 0,001 (valeur par défaut) alors par exemples :
   - $\mathbb{P}(w_i) = 1$ (w_i est toujours gardé) lorsque $z(w_i) \leq 0,0026$, c'est-à-dire si w_i représentent moins de 0,26 % du nombre total de mots.
   - $\mathbb{P}(w_i) = 0,5$ (50 % de chance de garder w_i) lorsque $z(w_i) = 0,00746$.
   - $\mathbb{P}(w_i) = 0,033$ (3,3 % chance de garder w_i) lorsque $z(w_i) = 1,0$ (si le corpus n'est constitué que du mot w_i, ce qui serait bien sûr absurde).

Ce sous-échantillonnage est effectué de manière indépendante pour chaque phrase : un même mot peut donc être sous-échantillonné dans une phrase et ne pas l'être dans une autre.

2. Pour chaque phrase, on tire au hasard (selon une loi uniforme) un mot focus pour lequel on tire un mot *contexte* au hasard dans la fenêtre w, en imposant que les deux mots choisis soient parmi les mots sous-échantillonnés [8]. Par exemple, nous supposons que dans la phrase `[que, pensez, vous, de, ce, projet, i, e, qu, avez, vous, retenu, en, 10min]`, les mots sous-échantillonnés sont les mots en position 2, 5, 6, 8, 9, 10, 11, 12, 13, 14. Pour mieux comprendre, nous remplaçons les mots non échantillonnés par « nonsubsampled ». La phrase devient alors `[nonsubsampled, pensez, nonsubsampled, nonsubsampled, ce, projet, nonsubsampled, e, qu, avez, vous, retenu, en, 10min]`. Si $w = 2$ alors le mot focus tiré ne peut pas être « pensez » puisque dans ce cas il n'y aurait aucun mot contexte associé. Si le mot focus tiré est « qu » alors le mot contexte est tiré au hasard parmi `[e, avez, vous]`.

Ce mécanisme va être répété sur toutes les phrases du corpus et l'ensemble du corpus va être parcouru plusieurs fois. Le nombre de fois que l'ensemble du corpus est parcouru est appelé *epochs*.

8. Si pour une phrase, aucun couple `[focus, contexte]` ne figurent simultanément dans les mots sous-échantillonnés, alors aucun couple n'est retenu pour cette phrase.

1.2.2 Descente de gradient

Pour chaque couple `[focus, contexte]`, les matrice W_e et W_s sont mises à jour par descente de gradient. C'est-à-dire que $\theta^{(t)} = W_e$ et $\theta^{(t)} = W_s$, les matrices obtenues après la $t^{\text{ème}}$ itération de l'algorithme, sont mises à jour par l'équation :

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta Loss(\theta^{(t)})$$

avec η le taux d'apprentissage (un hyperparamètre à fixer) et $L(\theta)$ la fonction de perte.

Le modèle *Word2vec* a initialement été construit en utilisant une fonction de perte dérivée de la fonction *softmax* (voir partie 1.2.2.1 et Mikolov *et al* (2013a)). L'algorithme a ensuite été amélioré en utilisant le *negative sampling* (voir partie 1.2.2.2 et Mikolov *et al* (2013b)).

1.2.2.1 Version softmax Soit $w_1, \ldots, w_T$ les mots utilisés pour entraîner le modèle. L'objectif du modèle Skip-Gram est, étant donné un mot focus, de prévoir quels sont les mots voisins contextes dans une certaine fenêtre w. Mathématiquement, on cherche à maximiser la quantité :

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-w \leq j \leq w,\, j \neq 0} \log \mathbb{P}(w_{t+j} | w_t)$$

où :

- les w_{t+j} sont les mots voisins de w_t (w_t est donc un mot focus et w_{t+j} un mot contexte) ;
- $\mathbb{P}(w_{t+j}|w_t)$ est la probabilité d'observer le mot contexte w_{t+j} sachant que l'on a observé le mot focus w_t. Cette quantité est calculée en fonction des matrices W_e et W_s à partir de la fonction softmax [9]. En notant n la taille du vocabulaire et W_{e,w_i} et W_{s,w_i} les représentations vectorielles du mot w_i respectivement dans la matrice d'entrée et de sortie, cette probabilité est égale à [10] :

$$\mathbb{P}(w_{contexte}|w_{focus}) = \frac{\exp(W_{e,w_{focus}} \times {}^tW_{s,w_{contexte}})}{\sum_{i=1}^{n} \exp(W_{e,w_{focus}} \times {}^tW_{s,w_i})}$$

Maximiser l'équation (1.2.2.1) revient à minimiser la fonction de perte suivante pour chaque couple [focus, contexte] :

$$Loss_1 = -\log \mathbb{P}(w_{contexte}|w_{focus}) = -W_{e,w_{focus}} \times {}^tW_{s,w_{contexte}} + \log\left(\sum_{i=1}^{n} \exp(W_{e,w_{focus}} \times {}^tW_{s,w_i})\right)$$

L'inconvénient de cette méthode est qu'elle est très gourmande en temps de calcul. En effet, pour chaque couple `[focus, contexte]`, la complexité du calcul de $\log \mathbb{P}(w_{contexte}|w_{focus})$ est proportionnelle à la taille du vocabulaire. La taille du vocabulaire pouvant être très grande (par exemple, dans notre base de tweets, cette taille est de 70 330), le temps de calcul peut vite devenir très important.

C'est pourquoi la version *softmax* est très peu utilisée dans les implémentations de Skip-Gram. Une approche alternative, *negative sampling* avec une fonction sigmoïde, moins gourmande en temps de calcul, est alors souvent préférée [11].

9. Étant donné le vecteur $z = (z_1, \ldots, z_n)$) la fonction softmax est la fonction qui à z associe le vecteur dont la $j^{\text{ème}}$ coordonnée est égale à $\frac{\exp(z_j)}{\sum_{i=1}^{n} \exp(z_i)}$.

10. Dans tout le rapport nous utiliserons la notation ${}^tX$ pour désigner la transposée de la matrice X.

11. Une autre alternative à l'approche *softmax* parfois utilisée est l'approche *hierarchical softmax* qui se base sur

1.2.2.2 Version *negative sampling*

Le *negative sampling* est basé sur le concept du *Noise Contrastive Estimation* – estimation contrastée du bruit – où on cherche, à partir d'un modèle logistique, à différencier un vrai signal (un vrai couple `[focus, contexte]`) d'un faux (un bruit qui correspondrait à un faux couple `[focus, contexte]` généré aléatoirement).

Dans cette approche, plutôt que de mettre à jour l'ensemble des représentations vectorielles des mots pour chaque couple `[focus, contexte]`, on tire K mots au hasard du vocabulaire $(w_{neg,i})_{i=1..K}$, selon une loi P (définie plus tard), en considérant que ces mots ne seront pas des mots voisins de `focus` [12].

L'approche *softmax* peut être vue comme un problème de classification multiclasses : étant donné un mot `focus`, on estime la probabilité que les autres mots soient parmi ses voisins (chaque classe étant un mot du vocabulaire). L'idée du *negative sampling* est de transformer ce problème de classification multiclasses en un problème de classification binaire d'une variable D : pour chaque couple `[focus, mot2]`, on cherche à déterminer si `mot2` est dans le contexte de `focus`. Si c'est le cas, alors $D = 1$ (mot2 est positif et est le `contexte`), sinon $D = 0$ (mot2 est négatif, il appartient à $(w_{neg,i})_{i=1..K}$).

On cherche donc à maximiser $\mathbb{P}(D = 1|w_{focus}, w_{contexte})$ et $\mathbb{P}(D = 0|w_{focus}, w_{neg,i})$. Pour estimer ces probabilités, on utilise une fonction sigmoïde plutôt que la fonction softmax :

$$\mathbb{P}(D = 1|w_{focus}, w_{contexte}) = \sigma({W_{e,w_{focus}}}^t W_{s,w_{contexte}}) = \frac{1}{1 + \exp(-{W_{e,w_{focus}}}^t W_{s,w_{contexte}})}$$

et :

$$\mathbb{P}(D = 0|w_{focus}, w_{neg,i}) = \sigma(-{W_{e,w_{focus}}}^t W_{s,w_{neg,i}}) = \frac{1}{1 + \exp({W_{e,w_{focus}}}^t W_{s,w_{neg,i}})}$$

Par rapport à l'approche *softmax*, on cherche toujours à maximiser la quantité de l'équation (1.2.2.1) mais en estimant $\log \mathbb{P}(w_{contexte}|w_{focus})$ par :

$$\log \mathbb{P}(w_{contexte}|w_{focus}) = \log \underbrace{\sigma({W_{e,w_{focus}}}^t W_{s,w_{contexte}})}_{\mathbb{P}(D=1|w_{focus},w_{contexte})} + \sum_{i=1}^{K} \mathbb{E}_{w_{neg,i}\sim P}[\log \underbrace{\sigma(-{W_{e,w_{focus}}}^t W_{s,w_{neg,i}})}_{\mathbb{P}(D=0|w_{focus},w_{neg,i})}]$$

Ainsi, pour chaque couple `[focus, contexte]` et un ensemble $(w_{neg,i})_{i=1..K}$ de mots négatifs tirés, on associe la fonction de perte suivante, à minimiser :

$$Loss_2 = -\log \sigma({W_{e,w_{focus}}}^t W_{s,w_{contexte}}) - \sum_{i=1}^{K} \log \sigma(-{W_{e,w_{focus}}}^t W_{s,w_{neg,i}})$$

La complexité est ici bien plus faible que pour la fonction softmax puiqu'elle est proportionnelle à K.

Mikolov *et al* (2013b) trouvent, empiriquement, que la meilleure distribution P pour générer les mots négatifs est telle que :

$$\mathbb{P}_P(w_i) = \frac{z(w_i)^{3/4}}{\sum_{j=1}^{n} z(w_j)^{3/4}}$$

Avec $z(w_i)$ la fréquence d'apparition du mot w_i.

Ils recommandent également de prendre $K \in \{5, \ldots, 20\}$ pour les petites bases de données et $K \in \{2, \ldots, 5\}$ pour les grandes bases de données. Dans ce projet, nous utiliserons $K = 5$ (pour chaque couple `[focus, contexte]` nous tirons donc 5 mots négatifs).

---

l'utilisation d'arbres binaires de classification. La complexité de cet algorithme est proportionnelle à $\log_2 n$ mais reste plus importante que celle de l'approche *negative sampling*.

12. Il est bien sûr possible que parmi les mots tirés au hasard il y ait des mots qui soient vraiment dans le contexte. Cependant, puisque la taille du vocabulaire est très grande, on considère que cette erreur est négligeable.

mot	similarité cosinus
énorme	0,991
taille	0,991
...	...
vanille	0,061
salissures	0,054

TABLE 1 – Mots les plus proches de « grand » par similarité cosinus

Note : Paramètres utilisés : ep = 50 / lr = 0,01 / w = 5 / dim = 10.

2 Évaluation du modèle implémenté

Malgré l'utilisation généralisée des *word embeddings*, très peu de travaux théoriques expliquent ce qui est réellement capturé par ces représentations de mots.

C'est pourquoi ce modèle est principalement évalué à l'aide de méthodes empiriques. Les méthodes que nous avons retenues pour évaluer, dans les parties qui suivent, la qualité des vecteurs-mots obtenus sont décrites plus précisément dans l'annexe A.

2.1 Évaluation sur un corpus fictif

Avant de nous attaquer au jeu de données complet décrit plus bas, nous avons évalué un premier corpus fictif afin de nous assurer de la robustesse et de la validité du modèle implémenté. Nous avons associé dix couples (du type [voiture, camion]), à dix mots contextes différents ([véhicule, moto...]). Le corpus fictif est formé de 10 000 phrases composées chacune d'un mot d'un couple, de cinq mots du contexte et de trois mots bruits, tous tirés aléatoirement.

Nous avons ensuite mis en œuvre les différentes techniques d'évaluation [13] présentées en partie **??** sur les *word-embeddings* obtenus grâce à ce corpus fictif.

Les résultats semblent concluants : la similarité cosinus montre bien une forte corrélation entre les mots focus et contexte du corpus initial et une faible corrélation avec les mots bruits (tableau 1). L'ACP et l'algorithme t-SNE permettent également de montrer graphiquement cette proximité (figure 2). Les clusters apparaissent de manière plus évidente avec t-SNE.

2.2 Choix des meilleurs hyperparamètres pour le modèle

Une fois nous être assurés de la bonne implémentation du modèle (partie 2.1) grâce au corpus fictif, nous nous sommes attachés à identifier les hyperparamètres les plus pertinents au regard des données dont nous disposons.

Ces données correspondent à un ensemble de 1,3 million de tweets [14] postés en France entre 2014 et 2017, supposés être représentatifs de l'ensemble de tweets nationaux publiés durant cette période.

Le modèle *Word2Vec* version Skip-gram, décrit en partie 1, fait en effet intervenir un certain nombre d'hyperparamètres parmi lesquels :

13. À l'exception de la méthode par « jugement humain » puisque le corpus est ici créé fictivement par ordinateur sans prêter attention au réel sens des mots.

14. Ces tweets, achetés à twitter, sont la propriété de l'Inria.

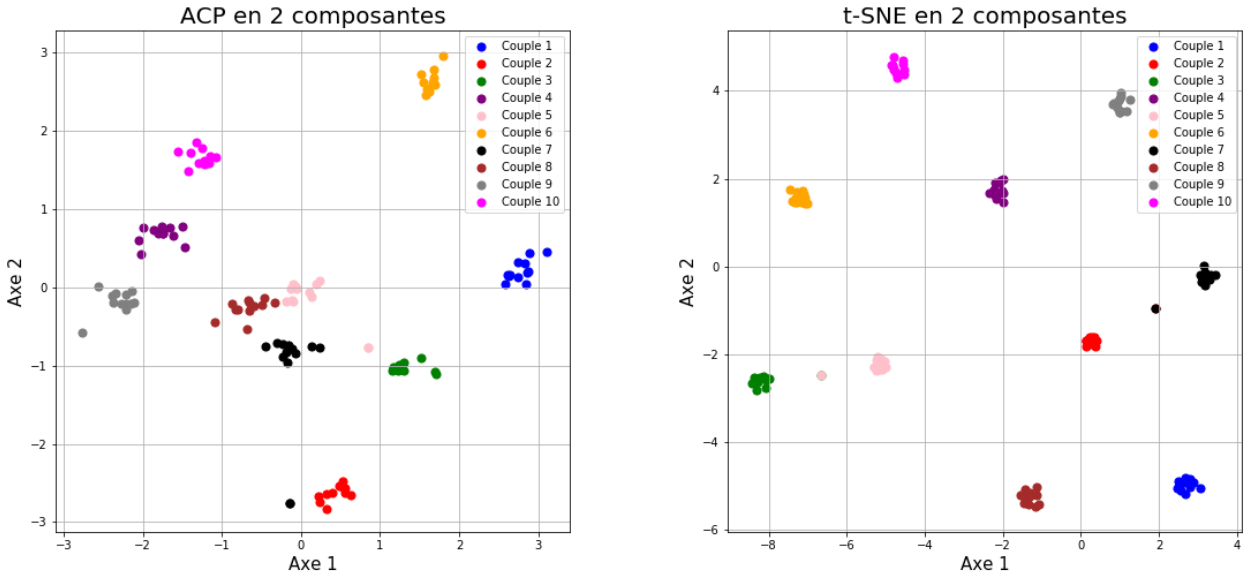

FIGURE 2 – Évaluation du modèle sur données fictives

Note : Paramètres utilisés : ep = 50 / lr = 0,01 / w = 5 / dim = 10.

- *ep* : le nombre d'« *epochs* »
- *lr* ou α : le « *learning rate* », ou taux d'apprentissage
- *w* (*window*): la taille de la fenêtre de sélection des mots contextes
- *dim* : la dimension des vecteurs-mots (ou *word-embeddings*)

Or, la performance de nombreuses méthodes de *machine learning*, dont *Word2Vec*, dépend fortement des valeurs choisies pour ces paramètres, ces valeurs étant elles-mêmes très dépendantes des données mobilisées.

Même si les méthodes d'optimisation bayésiennes deviennent de plus en plus performantes pour optimiser la valeur de ces hyperparamètres en tenant compte de leurs interactions (Hutter, Hoos & Leyton-Brown (2014)), ce choix s'effectue régulièrement de manière empirique, en testant différentes valeurs d'hyperparamètres sur les données mobilisées. C'est l'approche que nous retenons ici.

Le package `Gensim` (« Generate Similar »), dans lequel la méthode *Word2Vec* est implémentée, est un des outils actuels les plus robustes et performants [15] pour la modélisation sémantique non supervisée (Řehůřek & Sojka (2010)).

Nous avons choisi de mobiliser `Gensim` dans la suite de ce rapport, en parallèle du modèle que nous avons implémenté, en raison de son temps d'exécution bien plus rapide [16]. Cette rapidité d'exécution nous a permis de réaliser des tests d'hyperparamètres plus nombreux.

Pour réaliser ces tests, nous avons fait tourner le modèle *Word2Vec* plusieurs fois en modifiant un à un les paramètres. Nous avons ensuite évalué ces différents modèles par la méthode du « jugement humain » (voir partie A.3.1) en comparant la mesure de la similarité cosinus [17] entre deux mots

15. Grâce à sa dépendance à `NumPy`, `Gensim` puise dans des bibliothèques de bas niveau. Ainsi, alors que le code de haut niveau est du Python, c'est en fait du Fortran et du C hautement optimisés qui sont utilisés, ce qui rend `Gensim` bien plus performant que `PyTorch` que nous avons utilisé pour implémenter le modèle décrit en partie 1.

16. À titre d'exemple, alors qu'une epoch sur l'ensemble des tweets met une vingtaine d'heures à tourner pour « notre » modèle, elle met 1 minute via `Gensim`.

17. Nous avons également évalué les modèles en utilisant (l'inverse de) la distance euclidienne à la place de la similarité

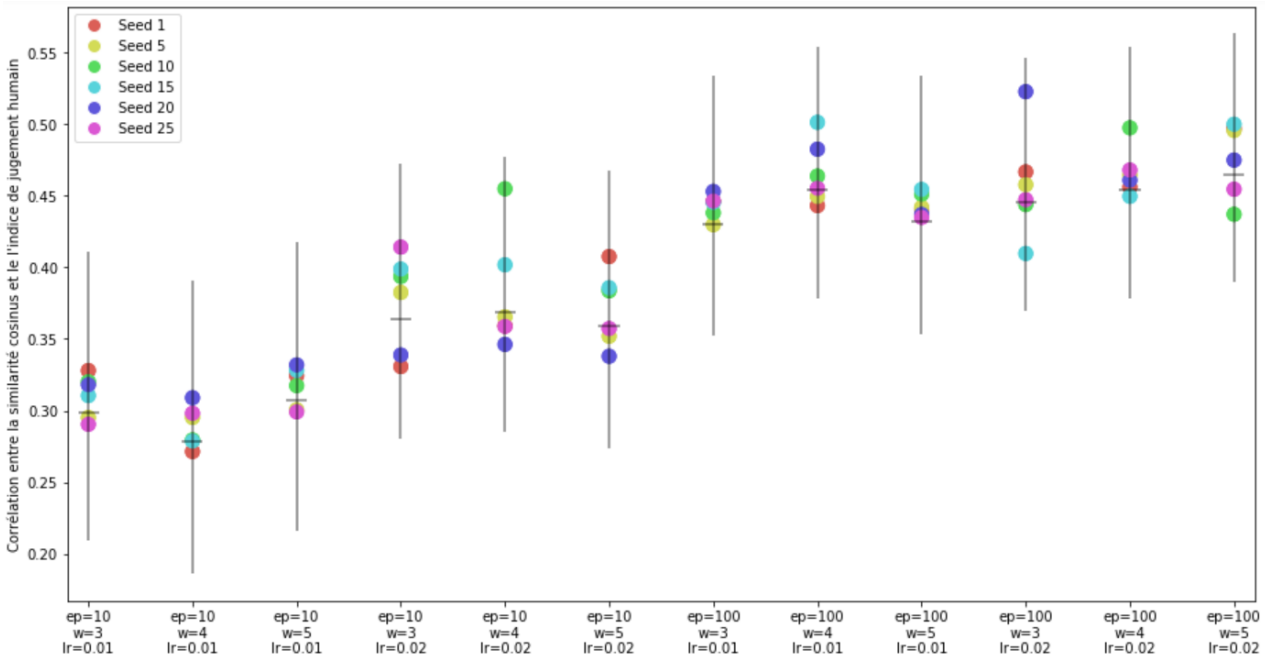

FIGURE 3 – Tests d'hyperparamètres : epochs, fenêtre et taux d'apprentissage

Note : Paramètres utilisés : dim = 50
Le trait horizontal correspond au coefficient de Spearman calculé sur les échantillons empilés des six modèles et la barre verticale à l'intervalle de confiance associé.

obtenue à partir de notre modèle à l'évaluation subjective de cette proximité par des individus. En outre, un même modèle est lancé six fois (six « seeds » différentes) afin de construire des intervalles de confiance de la matière décrite précédemment, en empilant les six échantillons de mesure de proximités correspondant aux six implémentations d'un même modèle [18].

2.2.1 Nombre d'epochs, taille de fenêtre et taux d'apprentissage

Pour cette première série de tests d'hyperparamètres, nous avons fixé la dimension des *word-embeddings* à 50 [19] et évalué l'impact du nombre d'epochs, de la taille de la fenêtre et du taux d'apprentissage (figure 3) .

2.2.1.1 Le nombre d'epochs

Le nombre d'epochs a un effet net. Passer de 10 à 100 epochs fait nettement augmenter le score de corrélation de Spearman entre données subjectives et données en sortie du modèle.

➔ Nous retenons alors le paramètre **ep = 100**.

cosinus. L'effet des paramètres devient alors bien moins clair et la performance du modèle est inférieure, ce va dans le sens de l'utilisation plus fréquente de la méthode de la similarité cosinus dans la littérature.

18. Pour chaque modèle, nous calculons les statistiques de rang des 65 paires de mots de la base de jugement humain ainsi que le rang des similarités cosinus des mots obtenus en sortie du modèle. Nous réalisons ces actions pour les six implémentations du même modèle et empilons les résultats obtenus. C'est à partir de cette base empilée de 6x65 lignes moins les données manquantes que nous calculons chaque intervalle de confiance selon la formule décrite en partie A.3.1.

19. En réalisant les mêmes tests sur uniquement 100 000 tweets, puis en testant une dimension de *word-embeddings* de 20, les effets observés et commentés ici se confirment.

2.2.1.2 Le taux d'apprentissage

La valeur 0,02 semble donner systématiquement de meilleurs résultats que 0,01. En réalisant davantage de tests de taux d'apprentissage en fixant les autres hyperparamètres, les différents taux d'apprentissage présentent des performances similaires [20].

➔ Nous retenons alors le paramètre **lr = 0,02**.

2.2.1.3 La taille de la fenêtre

La taille de la fenêtre ne semble pas jouer un rôle majeur, et dépend beaucoup des autres paramètres choisis.

Certains travaux (Levy & Golberg (2014)) indiquent que, suivant la taille de fenêtre choisie, les informations capturées sont différentes. Cela pourrait expliquer la complexité de choisir la « meilleure » taille de fenêtre. Alors que les « grandes » fenêtres capturent des informations sur le domaine du mot (autres mots de tout type étant utilisés dans des discussions connexes), les « petites » fenêtres saisissent davantage le mot en lui-même (ses extensions, synonymes, lui sont alors proches). La valeur de 4 représente une taille de fenêtre « ni trop grande ni trop petite » et qui présente de bons résultats dans la plupart des tests effectués.

➔ Nous retenons alors le paramètre **w = 4**.

2.2.2 Dimension des vecteurs-mots

On cherche cette fois-ci à évaluer l'effet de la dimension des *word-embeddings*. Selon certains papiers (comme Pennington, Socher & Manning (2014)), la qualité des représentations vectorielles s'améliore à mesure que l'on augmente la taille du vecteur, mais seulement jusqu'à atteindre 300 dimensions [21]. Après 300 dimensions, la qualité des vecteurs commence à diminuer et le temps de calcul augmente considérablement.

En pratique, en comparant l'effet de la dimension des vecteurs (modèle fixé à ep = 100, w = 4 et lr = 0,02), on observe bien une augmentation de l'efficacité du modèle jusqu'en dimension 300 et une efficacité moindre en dimension 500 (figure 4). Bien que l'efficacité du modèle semble meilleure en dimension 300, la dimension 100 améliore la rapidité de l'algorithme, pour des résultats d'une qualité similaire.

➔ Nous retenons alors le paramètre **dim = 100**.

20. En fixant les paramètres dim = 50, ep = 100 et w = 4 (celles du modèle retenu *in fine*), et en testant les taux d'apprentissage 0,005, 0,01, 0,02, 0,03 et 0,04, les valeurs moyennes des corrélations s'échelonnent entre 0,41 et 0,48, soit des valeurs proches.

21. La dimension des vecteurs doit également être adaptée à la taille du vocabulaire. Un des articles fondateurs de word2vec (Mikolov *et al* (2013a)) recommande donc d'augmenter à la fois la dimension des vecteurs et la quantité de données d'apprentissage. Par exemple, avec un vocabulaire d'une centaine de mots, il serait inefficace d'utiliser des projections en grande dimension (risque de surapprentissage).

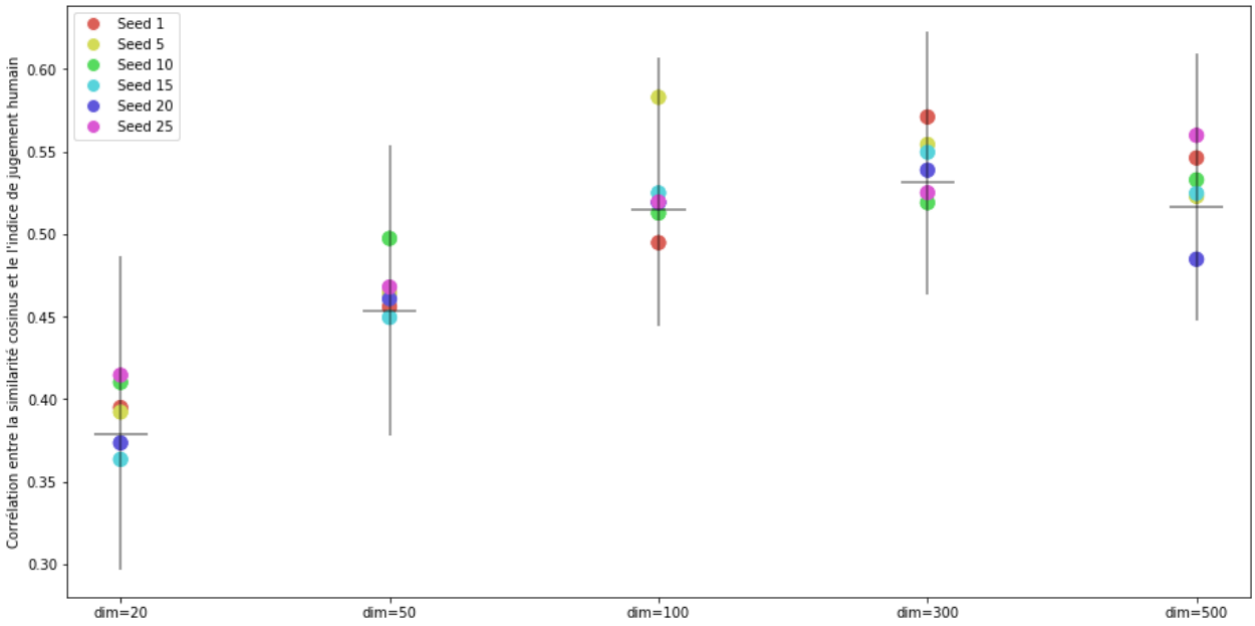

FIGURE 4 – Tests d'hyperparamètres : dimension des *word-embeddings*

Note : Paramètres utilisés : ep = 100 / w = 4 / lr = 0,02.
Le trait horizontal correspond au coefficient de Spearman calculé sur les échantillons empilés des six modèles et la barre verticale à l'intervalle de confiance associé.

2.3 Évaluation sur le corpus final

2.3.1 Avec « notre » modèle

Nous avons ensuite fait tourner le modèle que nous avons implémenté en utilisant les paramètres retenus précédemment [22] mais uniquement sur 100 000 tweets et 80 epochs pour des questions de temps de calcul [23].

Les résultats obtenus semblent relativement satisfaisants. La recherche des plus proches voisins par similarité cosinus (dont quelques exemples sont illustrés en tableau 2) donne des résultats proches de l'intuition.

Par ailleurs, le coefficient de Spearman entre la similarité cosinus des mots obtenus et le jugement humain est de 0,571 (p-valeur : 4,1 %). Toutefois, ce bon résultat est à considérer avec précaution puisque seuls 13 des couples de mots de la base RG-65 ont été reconnus dans le corpus de 100 000 tweets que nous utilisons ici.

Enfin, les représentations graphiques des positions des mots via des ACP et les sommes vectorielles sur les mots [24] donnent des résultats bien moins concluants que le modèle `Gensim` entraîné sur l'ensemble des tweets (voir partie 2.3.2).

22. w = 4, lr = 0,02 et dim = 100
23. Près de 18 heures.
24. Comme l'exemple de $\overrightarrow{Paris} - \overrightarrow{France} + \overrightarrow{Italie} = \overrightarrow{Rome}$ dans Mikolov *et al* (2013a)

bonjour *(669 apparitions)*	**femme** *(264 apparitions)*	**1** *(765 apparitions)*	**samedi** *(203 apparitions)*
😘 (0,59)	quelle (0,49)	5 (0,55)	soir (0,57)
😊 (0,59)	cette (0,46)	mois (0,51)	vivement (0,51)
merci (0,54)	une (0,44)	10 (0,49)	demain (0,50)
nuit (0,48)	vie (0,44)	2 (0,48)	end (0,48)
bisous (0,47)	grippe (0,44)	top (0,48)	weekend (0,47)
bonne (0,47)	belle (0,43)	depuis (0,47)	matin (0,45)
☺️ (0,46)	ma (0,43)	saison (0,46)	jeudi (0,45)
vous (0,46)	magnifique (0,43)	ans (0,44)	prochain (0,43)
plaisir (0,44)	nouvelle (0,43)	jours (0,43)	week (0,43)
allez (0,43)	vidéo (0,39)	3 (0,43)	🎉 (0,42)

TABLE 2 – 10 plus proches voisins par similarité cosinus avec « notre » modèle

Note : Paramètres utilisés : ep = 80 / w = 4 / lr = 0,02 / dim = 100 / base : 100 000 tweets
La similarité cosinus de chaque paire de mots est renseignée entre les parenthèses.

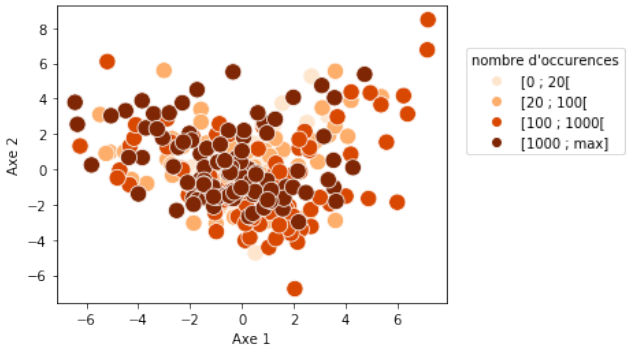

FIGURE 5 – Position des mots en fonction de leur nombre d'occurrences (Modèle `Gensim`)

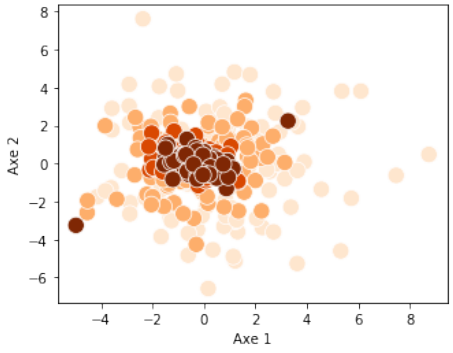

FIGURE 6 – Position des mots en fonction de leur nombre d'occurrences (« notre » modèle)

Note : Paramètres utilisés : ep = 100 (gauche) ou 80 (droite) / w = 4 / lr = 0,02 / dim = 100.
Méthode utilisée : ACP, deux premiers axes.

2.3.2 Avec le modèle `Gensim`

Le modèle `Gensim`[25] donne des résultats encore plus convaincants que précédemment, ayant été davantage entraîné, et sur un corpus plus fourni (ensemble des tweets). En effet, les vecteurs-mots en sortie du modèle `Gensim` sur l'ensemble des tweets (figure 5) sont davantage répartis dans l'ensemble du plan, alors que les mots en sortie du modèle que nous avons implémenté sur 100 000 tweets sont répartis en fonction de leur nombre d'occurrences, les mots les moins fréquents n'ayant probablement pas (ou peu) été entraînés (figure 6).

Le coefficient de Spearman a une valeur semblable à précédemment : 0,495 mais sa p-valeur est proche de 0 % et, cette fois-ci, 52 des couples de mots de la base RG-65 ont été reconnus dans le corpus de tweets.

Les 10 plus proches voisins calculés par similarité cosinus (tableau 3) semblent encore davantage pertinents. Les plus proches voisins de « 1 » contiennent davantage de chiffres, de « samedi » davantage

25. w = 4, lr = 0,02, dim = 100 et ep = 100.

bonjour *(17 043 apparitions)*	**femme** *(6 177 apparitions)*	**1** *(21 055 apparitions)*	**samedi** *(4 917 apparitions)*
bonsoir (0,85)	fille (0,86)	2 (0,65)	vendredi (0,88)
bjr (0,75)	copine (0,74)	3 (0,64)	jeudi (0,86)
hello (0,71)	meuf (0,71)	6 (0,63)	lundi (0,83)
salut (0,66)	demoiselle (0,66)	4 (0,62)	mercredi (0,83)
coucou (0,55)	nana (0,66)	7 (0,60)	dimanche (0,83)
transmets (0,49)	nièce (0,66)	5 (0,58)	mardi (0,76)
désagrément (0,48)	sœur (0,65)	9 (0,58)	demain (0,72)
avezvous (0,48)	barbe (0,65)	8 (0,56)	barathon (0,56)
bettembourg (0,48)	maman (0,64)	1e (0,55)	22h45 (0,55)
hey (0,47)	princesse (0,64)	34 (0,53)	20h (0,54)

TABLE 3 – 10 plus proches voisins par similarité cosinus avec le modèle `Gensim`

Note : Paramètres utilisés : ep = 100 / w = 4 / lr = 0,02 / dim = 100 / base : ensemble des tweets
La similarité cosinus de chaque paire de mots est renseignée entre les parenthèses.

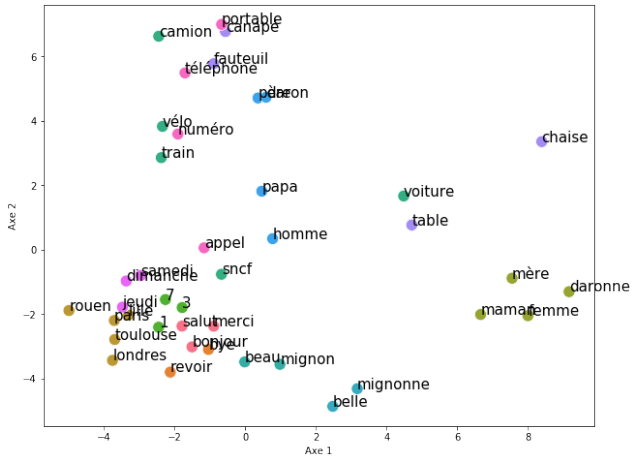

FIGURE 7 – ACP sur un corpus réduit de mots

Note : Paramètres utilisés : ep = 100 / w = 4 / lr = 0,02 / dim = 100 / base : ensemble des tweets

de jours de la semaine et le tableau contient désormais des synonymes de « femme » et de « bonjour ». Certains mots surprenants subsistent toutefois, comme par exemple « transmets », « désagrément » et « betembourg », voisins de « bonjour ». Toutefois, la fréquence d'apparition de ces mots dans le corpus est faible (moins d'une centaine d'occurrences). La projection de certains vecteurs-mots sur les deux premiers axes d'une ACP (figure 7) nous confirme la qualité de l'entraînement du corpus sur l'ensemble de tweets.

Enfin, nous avons réalisé des opérations sur les mots-vecteurs. Si l'opération $\overrightarrow{Roi} - \overrightarrow{Homme} + \overrightarrow{Femme} = \overrightarrow{Reine}$ (figure 8) semble fonctionner [26], l'opération $\overrightarrow{Paris} - \overrightarrow{France} + \overrightarrow{Italie}$ (figure 9) n'identifie pas « Rome »(similarité cosinus de 0,18 seulement) dans les mots les plus proches mais d'autres villes comme « Lyon » (similarité cosinus de 0,62). « Rome » semble effectivement située « trop en haut » dans le plan de l'ACP par rapport aux autres villes. Peut-être ce mot n'a-t-il pas suffisamment été entraîné (246 apparitions dans les tweets contre 46 433 pour Lyon par exemple) pour que le

26. Les similarités cosinus obtenues sont les suivantes : $corr(\overrightarrow{Roi}, \overrightarrow{Homme}) = 0,34$, $corr(\overrightarrow{Homme}, \overrightarrow{Femme}) = 0,35$ et $corr(\overrightarrow{Roi} - \overrightarrow{Homme} + \overrightarrow{Femme}, \overrightarrow{Reine}) = 0,67$. $\overrightarrow{Reine}$ est bien le mot le plus proche de la somme vectorielle calculée.

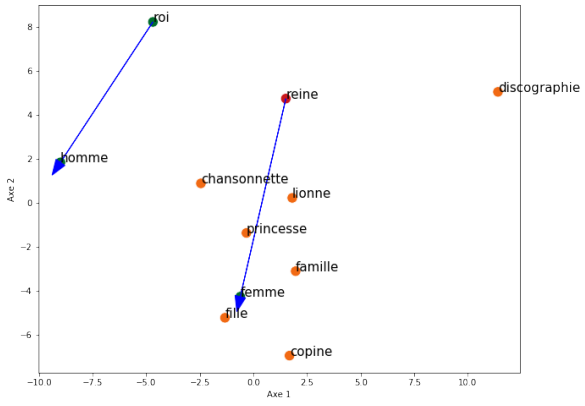

FIGURE 8 – $\overrightarrow{Roi} - \overrightarrow{Homme} + \overrightarrow{Femme} = ?$

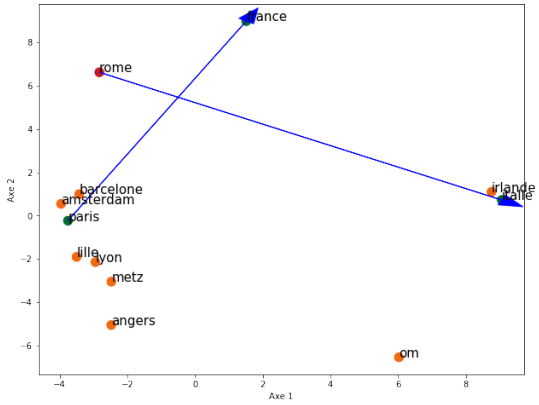

FIGURE 9 – $\overrightarrow{Paris} - \overrightarrow{France} + \overrightarrow{Italie} = ?$

Note : Paramètres utilisés : ep = 100 / w = 4 / lr = 0,02 / dim = 100.
Les mots en vert correspondent à ceux présents dans l'opération, le mot en rouge le mot que l'on serait supposé trouver et les mots en orange les 10 mots les plus proches du résultat de l'opération vectorielle.

vecteur-mot obtenu soit pertinent, ou peut-être que, dans les tweets mobilisés, le mot « Rome » s'utilise dans un contexte différent de l'article de Mikolov.

3 Analyse des sentiments

3.1 Prédire le sentiment d'un tweet à partir des *word-embeddings*

Nous cherchons dans cette partie à prédire le sentiment associé à un tweet. Nous nous intéresserons ici à 2 sentiments possibles : +1 quand le tweet est jugé positif, -1 quand il est jugé négatif. D'une manière générale, on se demande comment, à partir des mots d'un tweet, on peut prédire son sentiment. À partir de la base de tweets de la SNCF, nous allons donc chercher à construire le meilleur modèle de prédiction de sentiments. Nous comparerons alors deux approches : un première basée sur les sentiments moyens de chaque mot des tweets et l'autre basée sur les *word-embeddings*.

3.1.1 Prédiction à partir du sentiment moyen des mots

Un premier modèle de prédiction du sentiment peut être imaginé en utilisant l'information des tweets labelisés pour déterminer un sentiment moyen par mot. En effet, dans un premier temps, on peut regarder pour chaque mot du vocabulaire du corpus le nombre de tweets positifs (nb_+) et négatifs (nb_-) dans lesquels il apparaît et lui associer un sentiment moyen $\frac{nb_+ - nb_-}{nb_+ + nb_-}$, compris entre -1 et 1. Dans un second temps, on peut alors calculer la moyenne des sentiments des mots d'un tweet, et décider du label du tweet en fonction : si la moyenne est positive, le tweet sera jugé positif, sinon, il sera négatif. Autrement dit, considérons un tweet t composé des mots $\{mot_1, \ldots, mot_n\}$ alors $S(t) = 2 \times 1\{\frac{1}{n}\sum_{i=1}^{n} \alpha_i \geq 0\} - 1$ est la prédiction du sentiment du tweet, où α_i est le sentiment moyen du mot_i.

Afin d'étudier l'ajustement du modèle, nous séparons le corpus en une base de *train* composée de 16 004 tweets et une base de *test* composée de 6 858 tweets. Nous entraînons le modèle sur la base de *train*, c'est-à-dire que l'on calcule les sentiments moyens des mots de cette base, puis nous estimons le

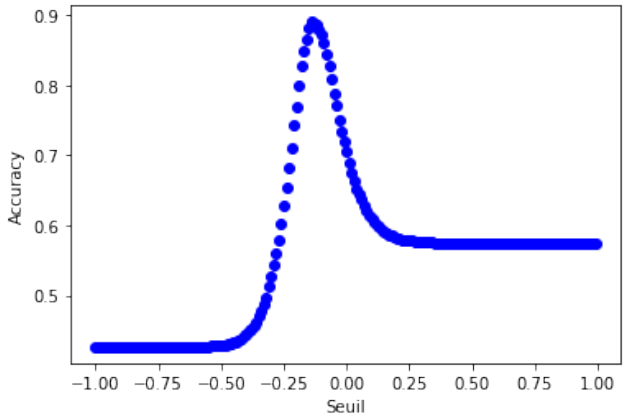

FIGURE 10 – Optimisation du seuil γ pour le modèle à partir des sentiments moyens des mots

modèle sur la base de *test*, c'est-à-dire que l'on prédit pour chaque tweet de cette base un sentiment, que l'on compare au vrai sentiment. Il peut exister des mots présents dans la base de *test* qui ne le sont pas dans la base de *train*, pour ces mots-là on met donc le sentiment moyen à 0. L'application de cette méthode permet d'obtenir une *accuracy*, c'est-à-dire un taux de tweets dont le sentiment est bien prédit, de $70,52\%$.

Ce modèle peut cependant être amélioré. En effet, pour l'instant on compare la moyenne des sentiments des mots de la phrase à un seuil qui vaut 0. Toutefois, on pourrait essayer d'optimiser ce seuil : $\forall \gamma \in [-1;1]$, on définit $S_\gamma(t) = 2 \times 1\{\frac{1}{n}\sum_{i=1}^{n} \alpha_i \geq \gamma\} - 1$ que l'on utilise pour effectuer une prédiction du sentiment, puis on sélectionne le γ qui maximise l'*accuracy* sur la base de *test*.

On effectue cette optimisation (figure 10) et on trouve alors que le γ qui maximise l'*accuracy* est $\gamma^* = -0.14$, pour une *accuracy* de $89,08\%$. On constate donc une nette amélioration par rapport au cas où $\gamma = 0$. Nous avons donc ici une premier modèle qui peut servir de référence pour la suite. Idéalement, nous souhaiterions que le modèle basé sur les *word-embeddings* soit meilleur que celui-ci.

3.1.2 Prédiction à partir des *word-embeddings*

Nous nous intéressons maintenant à un modèle basé sur l'utilisation de nos *word-embeddings*. Afin d'exploiter nos vecteurs, l'idée est d'utiliser un modèle de régression binaire de estimé sur notre corpus, en considérant comme prédicteurs chacune des dimensions des vecteurs. Pour réaliser cette régression, il faut toutefois pouvoir attribuer à chaque tweet un vecteur et effectuer une forme de *sentence embedding*. Pour cela, nous décidons d'attribuer à chaque tweet la moyenne des vecteurs de l'ensemble des mots du tweet. Nous pouvons ensuite réaliser la régression sur ce vecteur.

Si on note Y_i le sentiment du tweet i et $X_{i,1}, \ldots, X_{i,n}$ chacune des coordonnées du *sentence embedding* associé au tweet (nous avons ici $n = 100$), alors on s'intéresse au modèle $Y_i = 1\{\sum_{i=1}^{n} \beta_i X_{i,j} + \epsilon_i \geq 0\}$ où ϵ_i est le résidu de notre modèle. Nous allons chercher à estimer, en notant F_ϵ la fonction de répartition des résidus :

$$\mathbb{P}(Y_i = 1|X_{i,j}) = \mathbb{P}(\sum_{i=1}^{n} \beta_i X_{i,j} + \epsilon_i \geq 0|X_{i,j}) = \mathbb{P}(\sum_{i=1}^{n} \beta_i X_{i,j} \geq -\epsilon_i|X_{i,j}) = F_\epsilon(\sum_{i=1}^{n} \beta_i X_{i,j})$$

Deux modèles binaires peuvent être utilisés, selon la choix de F_ϵ :

1. Le modèle logit : $F_\epsilon(x) = \frac{1}{1+e^{-x}}$.
2. Le modèle probit : F_ϵ est la fonction de répartition d'une loi gaussienne centrée et réduite.

Nous avons donc deux choix de spécification à estimer sur notre corpus, afin de déterminer la meilleure. En réalité, nous pouvons également nous interroger sur d'autres paramètres à intégrer au modèle :

1. Les *stop words* : également appelé « mots vides », ce sont des mots communs dont l'analyse est en général inutile car leur présence n'apporte pas d'information. Dans notre cas, on peut donc penser à estimer le modèle en enlevant du corpus ces *stop-words*.
2. Les mots inconnus : la base qui a servi à entraîner le modèle des *word-embeddings* est différente de celle que l'on utilise pour la prédiction des sentiments. Ainsi, il y a des mots de ce corpus pour lesquels on ne connaît pas la représentation vectorielle. On peut alors décider que la représentation vectorielle des mots inconnus est le vecteur nul. Cependant, lorsqu'on a entraîné le modèle *Word2Vec* un pré-processing a permis de regrouper les mots très peu fréquents au sein d'un même mot clé et le modèle a alors calculé une représentation vectorielle commune à tous ces mots. On peut donc décider d'attribuer ce vecteur aux mots inconnus, en estimant que s'ils sont absents du corpus d'entraînement du modèle *Word2Vec*, ce sont bien des mots très peu fréquents.

Afin de mener une analyse la plus complète possible, nous avons décidé d'estimer tous les modèles possibles selon que l'on choisisse :

1. D'inclure ou non les *stop-words* ;
2. Le vecteur nul ou le vecteur des mots très peu fréquents estimé pour les mots inconnus ;
3. Le modèle probit ou logit.

Nous obtenons ainsi 8 modèles à estimer. Pour comparer ces modèles entre eux, nous avons décidé d'utiliser l'AUC, *Area Under the Curve*, c'est-à-dire l'aire sous la courbe ROC, *receiver operating characteristic*.[27] Nous effectuons alors une validation croisée, pour chacun de nos 8 modèles, en se basant sur l'AUC. Nous nous intéressons également à des critères économétriques afin de déterminer la meilleure représentation entre le modèle logit et le modèle probit.

Nous obtenons alors les résultats suivants :

1. Il ne semble pas utile d'enlever les *stop words* pour améliorer les résultats. Ceci peut s'expliquer car certains *stop-words* peuvent peut-être renseigner sur un sentiment. Ainsi, le mot « pas »est un *stop-word* mais renseigne plutôt sur des tweets négatifs (son sentiment moyen dans le corpus est de -0.25). De même, on peut penser que les phrases contenant du futur ou du conditionnel renvoie à des sentiments différents.
2. Il apparaît meilleur d'affecter aux mots inconnus le vecteur des mots peu fréquents estimé par le modèle *Word2Vec* que de leur attribuer le vecteur nul. On aurait pu intuitivement penser qu'en l'absence d'information sur un mot, il est préférable de lui attribuer un vecteur nul afin qu'il n'influe pas sur la décision du sentiment. Cependant, cela ne se vérifie pas empiriquement. Cela veut peut-être signifier que le vecteur associé aux mots très peu fréquents estimé par notre

27. La courbe ROC permet de représenter pour l'ensemble des seuils possibles l'évolution du nombre de vrais positifs en fonction du nombre de faux positifs. L'AUC, l'aire sous cette courbe, est donc compris entre 0 et 1 (le modèle qui prédirait de façon aléatoire 1 ou -1 a un AUC de 0,5).

modèle *Word2Vec* capture bien la rareté du mot et que les mots peu fréquents ont peut-être tendance à être plus positifs ou plus négatifs et non neutres.

3. Utiliser un modèle logit permet d'obtenir de meilleurs résultats qu'en utilisant un modèle probit. En effet, à la fois les AUC et les critères économétriques sont meilleurs pour les modèles logit, quels que soient les choix faits par ailleurs sur les *stop words* et les mots inconnus.

Finalement, nous utiliserons donc par la suite le modèle estimé par une **régression logit** sur la base en **gardant les *stop-words*** et en affectant aux mots inconnus dans le modèle *Word2Vec* le **vecteur des mots très peu fréquents** estimé par ce dernier.

L'application de ce modèle à la même base *test* que le modèle basé sur les sentiments moyens des mots donne une *accuracy* de $68,71\%$. Nous obtenons de moins bons résultats en utilisant le modèle basé sur les *word-embeddings*. Nous allons maintenant essayer d'analyser ces résultats.

3.1.3 Une cause des mauvaises prédictions : le *domain shift*

Les techniques d'apprentissage automatique sont souvent confrontées à un défi majeur : le fait que les modèles soient entraînés sur des données différentes de celles que l'on va effectivement utiliser : c'est ce que l'on appelle le « *dataset shift* »[28]. Bien que des solutions comme la correction de biais de sélection des échantillons, ou encore celle des données déséquilibrées, soient étudiées depuis de nombreuses décennies dans le monde de la statistique, certains autres problèmes, comme celui du changement de domaine (*domain shift*) émergent depuis plus récemment suite à l'utilisation croissante des méthodes de *machine learning*.

Le changement de domaine se caractérise par un changement de la nature, du « domaine » des données utilisées. Dans le cas de *deep-learning* sur des données de photographie, il pourrait par exemple s'agir de traiter de corpus de photos prises par des appareil photo calibrés de manière différentes (contraste, luminosité...) . Dans notre cas précis, il s'agit de la différence entre les données de tweets publiés en France entre 2013 et 2017 et les données de tweets de la SNCF, qui portent sur un sujet très spécifique et pour lesquels certains mots ont peut-être une interprétation spécifique en termes de sentiments. Modéliser le « domain shift » implique donc d'estimer le passage d'une représentation à une autre en utilisant des informations de distribution.[29] Dans ce qui suit, l'idée n'est pas de modéliser mathématiquement[30] le *domain shift* des deux bases de tweets mais de présenter quelques statistiques descriptives illustrant ce phénomène.

Pour ce faire, ...

3.2 Sentiments des tweets et enquête de conjoncture auprès des ménages [Alain]

Conclusion [sûrement Kim]

28. Les éléments en lien avec le *dataset shift* proviennent de Candela et al. (2009)

29. La correction gamma (représentation paramétrique non linéaire de l'intensité des pixels) est par exemple une manière de pouvoir traiter le « domain shift » lié à l'utilisation d'appareil photos différents.

30. Pour modéliser mathématiquement de manière relativement « grossière » un *domain shift*, il s'agirait de considérer par exemple une variable latente « idéale » x_0 (une base de tweets de référence), jamais observée mais qui influerait sur y (l'indice mensuel). Nous observons uniquement x telle $x = F(x_0)$ avec F qui représente la transformation de la base de tweets de référence à la base de tweets réellement utilisée, qui peut varier en fonction de la base de données x utilisée. La distribution $P(y|x_0)$ (de l'indice mensuel sachant le jeu de tweets idéal utilisé) est considérée comme étant la même pour les deux jeux de tweets utilisée (base « sncf » et base de tweets postés en France entre 2013 et 2017). En revanche, cette distribution est modifiée si F est modifiée.

A Comment évaluer le modèle ?

A.1 Distance entre deux mots

L'un des enjeux principaux du modèle étant de pouvoir estimer la proximité entre deux vecteurs-mots, nous pouvons tout d'abord mesurer cette dernière par des calculs de distance.

Il existe différents types de distances. Chacune d'elles possède des propriétés intéressantes et s'adaptent plus ou moins bien au problème traité. Nous avons ici retenu deux distances classiquement utilisées :

— **la distance euclidienne** : $d_e(\vec{u}, \vec{v}) = \|\vec{u} - \vec{v}\|_2$
Un problème est que la longueur du vecteur mot, captée dans le cas de la distance euclidienne, est positivement corrélée à la fréquence d'apparition du mot (Schakel & Wilson (2015)). Cette information peut s'avérer utile dans l'analyse de la signification des mots, notamment lorsque l'on effectue des opérations sur les vecteurs (comme l'exemple de $\overrightarrow{Paris} - \overrightarrow{France} + \overrightarrow{Italie} = \overrightarrow{Rome}$ dans Mikolov *et al* (2013a)).
Toutefois, cette dépendance à la fréquence d'apparition peut également fausser l'analyse. C'est pourquoi nous avons choisi, par la suite, de normaliser les vecteurs :

$$d_e(\vec{u}, \vec{v}) = \left\| \frac{\vec{u}}{\|\vec{u}\|_2} - \frac{\vec{v}}{\|\vec{v}\|_2} \right\|_2$$

— **la similarité cosinus** : $d_c(\vec{u}, \vec{v}) = \frac{\vec{u}.\vec{v}}{\|\vec{u}\|_2 \|\vec{v}\|_2}$.
La similarité cosinus correspond au produit scalaire entre les deux vecteurs normalisés. Elle mesure ainsi l'angle formé entre deux vecteurs-mots.
C'est la distance que de nombreux papiers fondateurs de la méthode *Word2Vec* (comme Mikolov *et al* (2013a) ou Levy & Golberg (2015)) utilisent, avec l'argument selon lequel les mots apparaissant dans des contextes similaires sont groupés dans la même direction durant l'entraînement. Une similarité est proche de $+1$ si deux mots sont positivement reliés (proches), de -1 s'ils sont négativement reliés (éloignés) et de 0 s'ils ne sont pas « reliés ».
Il est toutefois délicat d'interpréter une similarité proche de -1. On pourrait intuitivement penser à des antonymes, comme « grand » et « petit », mais en pratique, les antonymes sont susceptibles d'apparaître dans des contextes semblables et sont donc bien souvent positivement corrélés.

A.2 Analyse en Composantes Principales

Une fois le modèle *Word2Vec* entraîné, nous obtenons des *word-embeddings* pour chacun de nos mots, représentés par des vecteurs de grandes dimensions (20, 50 ou même supérieures à 100).

Dès lors, il devient complexe de bien observer la proximité entre deux mots. C'est pourquoi il devient utile de mobiliser des méthodes de réduction de dimensions comme l'analyse en composantes principales (ACP). En effet, l'objectif premier de cette méthode est de projeter un nuage de points sur un espace de dimension inférieure. Cela permet de rendre l'information moins redondante et plus visuelle, tout en étant le plus proche possible de la réalité.

Considérons le cas où nous disposons de n individus (dans notre cas les mots) et de p variables (dans notre cas, leurs composantes ou dimensions issues du modèle *Word2Vec*). On note $X = (x_{ij})$ la matrice de taille (n, p) des données brutes, où x_{ij} représente la valeur de la j-ème variable pour le i-ème individu. Mathématiquement, pour définir l'ACP, on définit deux espaces :

— L'*espace des individus*, de dimension p, auquel on associe la métrique M utilisée pour le produit scalaire. Dans la suite nous utiliserons $M = I_p$ la matrice identité. La norme et le produit scalaire associés à M sont donc euclidiens.
— L'*espace des variables*, de dimension n, auquel associe la métrique $N = diag(p_1, ..., p_n)$ avec $\sum_{i=1}^{n} p_i = 1$. La matrice N représente le poids donné à chaque individu. Par simplification nous utiliserons ici des poids uniformes : $N = \frac{1}{n} I_n$. Afin de donner le même poids à toutes les variables, chaque variable est centrée-réduite : cela revient à centrer-réduire les colonnes de notre matrice X. Nous notons $Z = \bar{X} = (z_{ij})$ la matrice des données centrées et réduites [31].

Pour toute métrique D ($D = N$ ou $D = M$), on associe le produit scalaire $\langle x, y \rangle_D = {}^t x D y$. La construction des axes de l'ACP est faite par projection orthogonale. La projection orthogonale d'un individu i (vecteur ligne) z_i sur une droite de vecteur directeur unitaire v vaut $\langle {}^t z_i, v \rangle_M = z_i \times v$ et les coordonnées de projection des n individus valent Zv.

Les vecteurs directeurs des axes sont définis de manière à maximiser la dispersion du nuage (son inertie [32]) des individus projetés et conserver ainsi au mieux les distances entre les individus. L'inertie [33] se définit alors comme :

$$
\begin{aligned}
I(Z) &= \sum_{i=1}^{n} p_i \|z_i - \bar{z}\|_M^2 \text{ avec } \bar{z} = \begin{pmatrix} \bar{z}_1 \\ \vdots \\ \bar{z}_p \end{pmatrix} = \begin{pmatrix} \frac{1}{n}\sum_{i=1}^{n} z_{i,1} \\ \vdots \\ \frac{1}{n}\sum_{i=1}^{n} z_{i,p} \end{pmatrix} (= 0_{\mathbb{R}^p} \text{ dans notre cas}) \\
&= \sum_{i=1}^{n} \frac{1}{n} \sum_{j=1}^{p} (z_{i,j} - \bar{z}_j)^2 \text{ car } M = I_p \\
&= \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} (z_{i,j} - \bar{z}_j)^2 \\
&= \sum_{j=1}^{p} Var(z^j), \text{ avec } z^j = \begin{pmatrix} z_{1,j} \\ \vdots \\ z_{n,j} \end{pmatrix} \\
&= p \text{ car les variables sont réduites}
\end{aligned}
$$

On trouve tout d'abord le vecteur directeur v_1 qui orientera le premier axe de l'ACP grâce au programme suivant :

$$
v_1 = \underset{\|v\|_M = 1}{\operatorname{argmax}} \underbrace{\|Zv\|_N}_{= Var(Zv)} = \underset{\|v\|_M = 1}{\operatorname{argmax}} {}^t v R v
$$

où $R = Var(Z) = \frac{1}{n} {}^t Z Z$ est la matrice des corrélations entre les p variables.

Puis, on choisit v_2 orthogonal à v_1 tel que l'inertie soit toujours maximisée :

$$
v_2 = \underset{\|v\|_M = 1,\, v \perp v_1}{\operatorname{argmax}} Var(Zv)
$$

31. Nous travaillons ici dans le cadre d'une ACP normée où la matrice X a été centrée puis réduite. La réduction de X a modifié les distances initiales entre individus ($\langle z_i, z_{i'} \rangle_M \neq \langle x_i, x_{i'} \rangle_M$). Cela n'aurait pas été le cas si la matrice X avait été uniquement centrée (ACP non normée).

32.

33. La dispersion d'un nuage de points unidimensionnel par rapport à sa moyenne se mesure par la variance. Dans le cadre multidimensionnel, la dispersion du nuage par rapport à son barycentre $\bar{z}$ se mesure par l'inertie, qui généralise la variance.

En procédant de manière séquentielle, on obtient $q < r$ axes orthogonaux avec $r = rg(Z)$ et q choisi par le statisticien [34].

On peut montrer que $\forall k < q$:

- v_k est un vecteur propre associé à la k^{e} valeur propre λ_k de R (les valeurs propres étant rangées par ordre décroissant) ;
- la composante principale Zv_k est centrée et $V(Zv_k) = \lambda_k$;
- les Zv_k ne sont pas corrélés entre eux.

On obtient alors la matrice $F = ZV$ des nouvelles coordonnées factorielles des individus, avec $V = (v_1, \ldots, v_q)$ la matrice des vecteurs propres.

Nous utilisons ici l'ACP en vue d'identifier les individus (ici, nos mots) qui sont proches. Pour ce faire, il suffit de représenter les coordonnées factorielles de la matrice F dans des repères, en général en 2 dimensions pour une question de lisibilité. Deux mots apparaissant dans des contextes similaires seront proches sur ce repère et orientés dans la même direction.

Enfin, pour juger de la qualité de la réduction de dimension, on calcule souvent la proportion de l'inertie totale expliquée par les q premières composantes principales.

$$\frac{V(F)}{I(Z)} = \frac{\sum_{i=1}^{q} \lambda_i}{p}$$

A.3 Algorithme *t-distributed Stochastic Neighbor Embedding*

Bien que l'ACP soit une première manière de résumer l'information contenue dans nos vecteurs, elle présente des limites, notamment dans les vecteurs aux trop grandes dimensions, pour lesquels l'inertie des premiers axes de l'ACP peut se révéler faible.

Pour combler ces lacunes, un autre algorithme de réduction de dimension peut être utilisé, celui dit du *t-distributed Stochastic Neighbor Embedding* (t-SNE). Contrairement à l'ACP, cet algorithme est stochastique et non-linéaire et il favorise l'apparition de groupes de mots proches. Sa philosophie demeure cependant identique : représenter dans un espace à dimension réduite notre nuage de points de manière à repérer les mots proches.

La première étape de l'algorithme consiste à calculer les similarités entre les n vecteurs-mots $(x_i)_{i=1\ldots n}$. La similarité entre x_i et x_j se mesure comme étant la probabilité conditionnelle $p_{j|i}$ de choisir x_j comme voisin de x_i, si les voisins étaient tirés au sort selon une loi $\mathcal{N}(x_i, \sigma_i)$ [35] :

$$p_{j|i} = \frac{\exp\left(-\frac{(d_e(x_i - x_j))^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{(d_e(x_i - x_k))^2}{2\sigma_i^2}\right)}$$

La seconde étape de l'algorithme consiste à trouver le nouvel espace de projection à faible nombre de dimensions. On appellera g_i les x_i projetés dans cet espace que l'on cherche à déterminer. On calcule

34. Différentes méthodes existent afin de déterminer le q optimal, comme la règle de Kaiser ou encore celle du coude.

35. σ_i doit être calculé de manière à adapter la loi conditionnelle aux données. Une faible dispersion autour de x_i entraînera un σ_i faible et réciproquement. Il s'agit de trouver le σ_i qui minimise ce qui est appelé en théorie de l'information la « perplexité », c'est-à-dire un indicateur qui décrit à quel point une distribution de probabilité réussit à prédire un échantillon.

maintenant les probabilité conditionnelles $q_{j|i}$ de choisir g_j comme voisin de g_i en supposant que les $(g_i)_i$ suivent cette fois-ci une distribution de *Student* – d'où le nom de l'algorithme – plutôt qu'une loi gaussienne [36].

$$q_{j|i} = \frac{(1 + (d_e(g_i - g_j))^2)^{-1}}{\sum_{k \neq i} (1 + (d_e(g_i - g_k))^2)^{-1}}$$

Afin d'obtenir les g_i, on minimise, par descente de gradient, la divergence de Kullback–Leibler entre les distributions de probabilité P et Q des p_{ij} et q_{ij} définis par :

$$KL(P,Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \qquad \text{avec} \qquad p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

Comme dans l'algorithme de l'ACP, l'algorithme de t-SNE nous permet d'obtenir une nouvelle projection des x_i. Il faut cependant analyser avec précaution ses résultats. L'algorithme n'étant pas linéaire, l'interprétation de la taille des *clusters* obtenus ou de la distance qui les sépare n'est alors pas directe.

A.3.1 Jugement humain

Les *word-embeddings* obtenus par *Word2Vec* sont censés regrouper les mots qui apparaissent dans un contexte similaire. Une dernière façon de vérifier la qualité de nos vecteurs-mots est de les comparer à un jugement humain. Pour ce faire, nous utilisons la liste de référence RG-65 pour le français [37] (Boumedyen Billami & Gala (2017)). Elle contient 65 paires de noms communs (tableau 4) évaluées sur une échelle de 0 (non liés) à 4 (très liés).

Nous calculons ensuite la corrélation de Spearman entre les similarités cosinus de ces différentes paires issues de notre modèle (notées ici $(X_i)_{i=1..n}$) et les scores proposés ci-dessus par des êtres humains (notés ici $(Y_i)_{i=1..n}$).

La corrélation de Spearman est égale au coefficient de corrélation de Pearson calculé sur les variables de rang.

$$r_s = \text{corr}(\text{rg}_X, \text{rg}_Y) = \frac{\text{cov}(\text{rg}_X, \text{rg}_Y)}{\sigma_{\text{rg}_X} \sigma_{\text{rg}_Y}}$$

La variable de rang rg_{X_i} est définie telle que $\text{rg}_{X_i} = j \iff X_i = X_{(j)}$ (X_i est la $j$ème plus petite variable).

Pour tester la significativité de ce coefficient, nous utilisons la loi sous (H_0) de la statistique de test $z = \text{arctanh}(r_s) = \frac{1}{2} \ln \frac{1+r}{1-r} \overset{H_0}{\sim} \mathcal{N}(0, \frac{1}{n-3})$ et obtenons l'intervalle de confiance suivant :

$$IC_\alpha(r_s) = \left[\tanh\left(z - \frac{q_{1-\frac{\alpha}{2}}}{\sqrt{n-3}} \right), \tanh\left(z + \frac{q_{1-\frac{\alpha}{2}}}{\sqrt{n-3}} \right) \right]$$

avec $q_{1-\frac{\alpha}{2}}$ le quantile d'ordre $1 - \frac{\alpha}{2}$ d'une loi $\mathcal{N}(0,1)$.

36. Dans un espace à faible dimension, la dispersion des vecteurs est réduite. La distribution de Student possède des queues plus épaisses que la loi normale, ce qui permet de mieux différencier les vecteurs distants des vecteurs similaires.

37. Le RG-65 a fait appel à 18 évaluateurs humains. La base, initialement mobilisée dans un article anglophone (Rubenstein & Goodenough (1965)) a été traduite de l'anglais.

mot 1	mot 2	similarité
corde	sourire	0,00
midi	ficelle	0,00
...	...	...
corde	ficelle	3,33
...	...	...
automobile	auto	3,94
coq	coq	4,00

TABLE 4 – Base de données de jugement humain

Références

Bengio, Y., Ducharme, R., Vincent, P., Janvin, C. (2003). A Neural Probabilistic Language Model. JMLR, 3:1137–1155. https://papers.nips.cc/paper/1839-a-neural-probabilistic-language-model.pdf.

Boumedyen Billami, M., Gala, N (2017). Création et validation de signatures sémantiques : application à la mesure de similarité sémantique et à la substitution lexicale. TALN 2017. https://hal.archives-ouvertes.fr/hal-01528117/document.

Candela, J. Q., Sugiyama, M., Schwaighofer, A., & Lawrence, N. D. (2009) Dataset shift in machine learning. The MIT Press, 1, 5. http://www.acad.bg/ebook/ml/The.MIT.Press.Dataset.Shift.in.Machine.Learning.Feb.2009.eBook-DDU.pdf.

Hutter, F., Hoos, H., Leyton-Brown, K., (2014). An Efficient Approach for Assessing Hyperparameter Importance. PMLR 32(1):754-762. http://proceedings.mlr.press/v32/hutter14.pdf.

Jurafsky, D., Martin, J. H. (2019). Speech and Language Processing (3rd ed. draft). Prentice Hall. https://web.stanford.edu/~jurafsky/slp3/edbook_oct162019.pdf.

Levy, O., Golberg, Y. (2015). Neural Word Embedding as Implicit Matrix Factorization. https://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf.

Levy, O., Golberg, Y. (2014). Dependency-based word embeddings. ACL. http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf.

Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781. https://arxiv.org/pdf/1301.3781.pdf.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality, arXiv:1310.4546. https://arxiv.org/pdf/1310.4546.pdf.

Pennington, J., Socher, R., Manning, C. D., (2014). Glove: global vectors for word representation. Proc. of EMNLP,1532 – 1543. https://www.aclweb.org/anthology/D14-1162.pdf.

Řehůřek, R., Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks. p. 46–50, 5 pp. ISBN 2-9517408-6-7. https://is.muni.cz/publication/884893/en.

Rubenstein, H., Goodenough, J. B. (1965). Contextual Correlates of Synonymy. Commun. ACM, 8 (10), 627–633. https://dl.acm.org/doi/10.1145/365628.365657.

Schakel, A. M., Wilson, B. J. (2015). Measuring Word Significance using Distributed Representations of Words. arXiv:1508.02297. https://arxiv.org/pdf/1508.02297v1.pdf.