

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)

1 Partie 1

2 Évaluation du modèle implémenté

2.1 Comment évaluer le modèle ?

Malgré l'utilisation généralisée des *word embedding*, 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. Nous allons décrire dans cette partie 2.1 quelques méthodes que nous avons retenues pour évaluer la qualité des vecteurs-mots obtenus.

2.1.1 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** $d_e(\vec{u}, \vec{v}) = \|\vec{u} - \vec{v}\|_2$

La longueur du vecteur mot, captée dans le cas de la distance euclidienne, est positivement corrélée à la fréquence d'apparition du mot (Bortoli, C., *et al* (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, T., *et al* (2013))

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** $d_c(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|_2 \|\vec{v}\|_2}$

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, T., *et al* (2013) ou Levy, O., Golberg, Y. (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.

2.1.2 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). L'objectif premier de cette méthode est en effet de projeter un nuage de points sur un espace de dimension inférieure afin 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. Afin de donner à chaque individu le même poids, nous centrons et réduisons les colonnes de notre matrice de données. On notera par la suite $Z = (z_{ij})$ la matrice des données centrées et réduites.

La construction des axes de l'ACP est faite par projection orthogonale. Nous utilisons ici le produit scalaire $\langle x, y \rangle_N = x^t N y$ avec la métrique $N = \text{diag}(\frac{1}{n}, \dots, \frac{1}{n})$. Ainsi, la projection orthogonale d'un individu i (vecteur ligne) z_i sur une droite de vecteur directeur v vaut ${}^t z_i v$ 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) des individus projetés et conserver ainsi au mieux les distances entre les individus. L'inertie se définit comme

$$I(Z) = \frac{1}{n} \sum_{i=1}^n d_e^2(z_i, \bar{z}) = \sum_{i=1}^n \text{var}(z^j) = p$$

avec $d_e(z_i, z_{i'})$ la distance euclidienne entre deux individus z_i et $z_{i'}$: $d_e(z_i, z_{i'}) = \sum_{j=1}^p (z_{ij} - z_{i'j})^2$ ¹.

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

$$v_1 = \underset{\|v\|=1}{\text{argmax}} \text{Var}(Zv) = \underset{\|v\|=1}{\text{argmax}} v^t R v$$

où $R = \text{Var}(Z) = \frac{1}{n} Z^t Z$ est la matrice des corrélations entre les p variables. La norme du vecteur v se calcule dans ce nouvel espace comme $\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{v^t v} = \sqrt{\sum_{i=1}^p v_i^2}$

1. 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 ($d_e(z_i, z_{i'}) \neq d_e(x_i, x_{i'})$). Cela n'aurait pas été le cas si la matrice Y avait été uniquement centrée (ACP non normée).

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

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

En procédant de manière séquentielle, on obtient $q < r$ axes orthogonaux avec $r = \operatorname{rg}(Z)$ et q choisi par le statisticien².

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

- v_k est un vecteur propre associé à la k^{e} valeur propre λ_k de R
- 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 $V = (v_1, \dots, v_q)$ 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.

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

2.1.3 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. 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 $(x_i)_{i=1\dots n}$. 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 $\mathcal{N}(x_i, \sigma_i)$ ³

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

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

3. 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.

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. De la même manière que précédemment, on exprime des probabilité conditionnelles $q_{j|i}$ en fonction des g_i mais qui suivent cette fois-ci une distribution de *Student* - d'où le nom de l'algorithme - plutôt qu'une loi gaussienne⁴.

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

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 :

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

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.

2.1.4 Jugement humain

Les *word-embedding* obtenus par *Word2Vec* sont censés regrouper les mots qui apparaissent dans un contexte similaire. Une dernière façon de le vérifier peut être de comparer les résultats obtenus au point de vue des êtres humains sur la proximité qu'il peut y avoir entre différentes paires de mots.

Pour ce faire, nous utilisons une base de données d'une soixantaine de paires de mots auxquelles sont associés des scores de similarité allant de 0 (aucun lien entre les mots) à 4 (mots identiques).

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

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.

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

4. 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.

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

Pour tester la significativité de ce coefficient, nous réalisons un test de Student avec

$$t = r \sqrt{\frac{n-2}{1-r^2}} \stackrel{H_0}{\sim} St(n-2)$$

Speech sur intervalle de confiance

2.2 Evaluation sur un corpus fictif

Avant de nous attaquer au jeu de données complet, nous avons évalué un premier corpus fictif afin de nous assurer de la robustesse du modèle implémenté. Nous avons associé dix couples (du type [voiture, camion]), à dix mots contexte 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[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.] présentées en partie 2.1 sur ce corpus fictif.

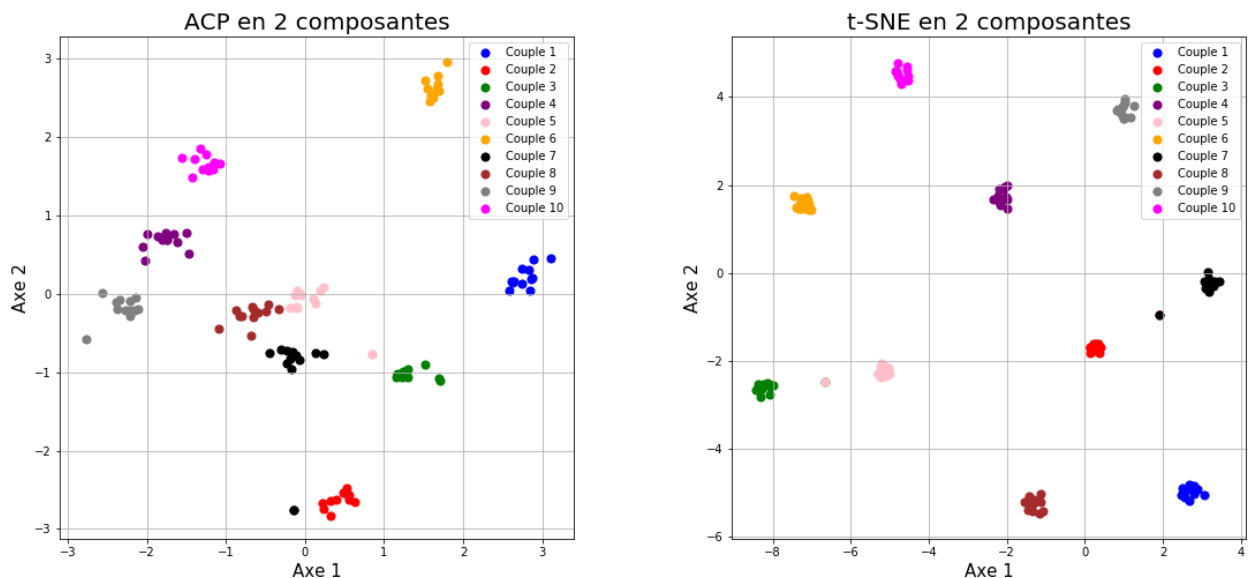


FIGURE 1 – Évaluation du modèle sur données fictives

Note : Paramètres utilisés : $epochs = 50$ / $lr = 0,01$ / $window = 5$ / $dim = 10$.

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. L'ACP et l'algorithme t-SNE permettent également de montrer graphiquement cette proximité (figure 1). L'algorithme t-SNE semble bien discriminer davantage les clusters que l'ACP.

```
mot_plus_proche("grand", n = 50)
#[('énorme', 0.9914256427481623),
# ('taille', 0.9905528713008166),
# [...]]
# ('vanille', 0.06068283530950071),
# ('salissures', 0.0539210063101789)]
```

2.3 Choix des meilleurs hyperparamètres pour le modèle

Une fois nous être assurés de la bonne implémentation du modèle (partie 2.2) grâce au corpus fictif, nous nous sommes attachés à identifier les hyperparamètres les plus pertinents au regard des données dont nous disposons.

Le modèle word2vec version CBOW, décrit en partie 1, fait en effet intervenir un certain nombre d'hyperparamètres⁵ :

- α : le « *learning rate* », ou taux d'apprentissage
- *window* : la taille de la fenêtre de sélection des mots contextes

Compléter après lecture partie Alain et harmoniser le nom des hyperparamètres dans l'ensemble du rapport.

Or, la performance de nombreuses méthodes de *machine learning*, dont Word2Vec, dépend fortement des valeurs choisies pour les hyperparamètres, ces dernières dépendant directement des données mobilisées.

Même si les méthodes d'optimisation bayésiennes deviennent de plus en plus performantes pour optimiser la valeur de ces hyperparamètres et de leurs interactions (Hutter, F., *et al* (2014)), le choix de ces paramètres s'effectue régulièrement de manière empirique, en testant différentes valeurs sur les données mobilisées. C'est l'approche que nous retenons ici.

2.3.1 Utilisation du modèle Gensim

Le package **Gensim** (« Generate Similar »), dans lequel la méthode *Word2Vec* est implémentée, est un des outils actuels les plus robustes et performants⁶ pour de modélisation sémantique non supervisée (Řehůřek, R., *et al* (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 exécution bien plus rapide⁷. Cette rapidité d'exécution nous a permis de réaliser de plus nombreux tests d'hyperparamètres que nous vous présentons ici.

Afin de tester l'effet des différentes valeurs des hyperparamètres, 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 » (cf. partie 2.1.4). En outre, un même modèle est lancé six fois (six « seeds » différentes) afin de construire des intervalles de confiance de la matière décrite précédemment.

5. les paramètres en gras sont ceux dont nous avons évalué l'effet

6. 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 utilisons.

7. A titre d'exemple, alors qu'une epoch sur l'ensemble des tweets met une vingtaine d'heures à tourner pour « notre » modèle, elle met **X minutes** via **Gensim**.

2.3.2 Sur de « notre » modèle

TODO : comparer en espérant que ça donne des résultats à peu près semblables à Gensim.

2.4 Evaluation sur le corpus final

Montrer les 4 évaluation du corpus final de 1 000 000 de tweets. Dire en quoi le modèle semble bon, en quoi il se

Références

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