Sorbonne Nouvelle
université des cultures

Robustesse des Plongements Lexicaux : Analyse Comparative sur GloVe et word2vec à l'aide des Mesures de Distance

Sémantique Computationnelle – LYST001

Master 2 - Traitement Automatique des Langues

Xinhao ZHANG -N°22205750

xinhao.zhang@sorbonne-nouvelle.fr

PluriTAL 2023

Ce devoir maison a pour objectif de comparer la robustesse des word embeddings dans les modèles connexes, tels que GloVe et word2vec, en utilisant plusieurs mesures de distance pour les 20 plus proches voisins de 20 mots de notre choix. Il me semble vraiment intéressant d'explorer en profondeur comment ces modèles, bien qu'entraînés sur le même corpus, peuvent tout de même différer dans la représentation des mots et leurs voisins les plus proches.

Dans le cadre du descriptif de mon travail, la première étape consistera à choisir un corpus, idéalement en anglais, et à en sélectionner les mots les plus fréquents. Ensuite, ce corpus sera utilisé pour entraîner les modèles démo de GloVe et Word2Vec, afin de générer des listes des k voisins les plus proches pour chaque mot sélectionné. On employera ainsi différentes mesures de distance pour évaluer les divergences entre les listes de voisins produites par les deux modèles. Le devoir se conclura par le classement des mots ré-ordonnés en fonction de la distance observée entre leurs listes de voisins dans les modèles GloVe et Word2Vec. Toute les données et les scripts peuvent se trouver dans mon github.

i Brève Présentation sur Word Embedding

S'agissant d'une technique fondamentale du domaine de TAL, les word embeddings(en français plongments des mots) renvoient principalement à transformer les mots en vecteurs numériques. Parmi les outils importants de word embeddings figurent Word2Vec, développé par Google, qui utilise des réseaux neuronaux pour modéliser les relations entre les mots ; GloVe(Global Vectors for Word Representation) de l'Université Stanford, qui analyse les cooccurrences statistiques des mots dans un corpus pour capturer leur signification. C'est avec ces différents modèles des embeddings que nous travaillerons par la suite dans ce devoir.

ii Préparation des Données et Expériences

ii.1 Corpus et Sélection des Mots

Afin d'obtenir vraisemblablement l'aisance avec la manipulation du corpus littéraire dans mon prochain stage, j'ai ainsi sélectionné un gros corpus des littératures anglo-saxonnes sur le site Huggingface. De plus, lorsque la consigne demande l'utilisation des modèles pré-entraînés pour l'anglais, il est préférable pour moi de m'orienter vers le corpus tout en anglais, parce que les modèles pré-entraînés seront spécifiques à la langue sur laquelle ils ont été entraînés.

Le corpus choisi englobe un large éventail d'œuvres littéraires, incluant à la fois des romans et des essais. Dans le dossier du corpus, l'ensemble des textes est conservé dans un seul fichier texte, ce qui rend idéal pour l'entraînement des modèles. Pour plus de détails du corpus, veulliez télécharger à partir de ce lien.

Pour la sélection des mots à analyser dans les modèles GloVe et word2vec, je me concentrerai sur l'extraction des mots fréquents, car après avoir observé le script bash de configuration pour le modèle Glove, j'ai découvert qu'une partie du script **demo.sh** s'occupe de calculer la fréquence des mots du corpus et de sauvegarder ces informations dans un fichier nommé **vocab.txt**.

```
1 $BUILDDIR/vocab_count -min-count $VOCAB_MIN_COUNT -verbose $VERBOSE < $CORPUS
    ↪ > $VOCAB_FILE
```

Avec cette ligne de commande, nous pourrons directement utiliser ce fichier de résultat généré suite à l'application du modèle GloVe pour identifer les mots les plus fréquents du corpus. Donc, je vous montre dans la Table 1 les 20 mots les plus fréquents dans ce fichier, qui peuvent être considérés comme une liste de mots à expérimenter.

TABLE 1 – liste des 20 mots les plus fréquents de mon corpus selon GloVe

Mot	Fréquence	Mot	Fréquence
the	261656	his	52299
and	165884	with	41438
of	141869	it	39038
to	133386	for	37134
a	100140	is	36795
in	79961	had	36309
I	69119	as	36266
that	61594	not	33603
he	59566	you	33597
was	57366	at	30141

ii.2 Obtention des Vecteurs avec Modèles Pré-entraînés

Suite au téléchargement des modèles pré-entraînés en anglais pour GloVe et Word2Vec depuis leurs plateformes officielles, j'ai exécuté les scripts **demo.sh** pour GloVe et **demo-word.sh** pour Word2Vec directement sur mon terminal, sans modifier aucun hyperparamètre, à l'exception du chargement de mon propre corpus plutôt que le corpus démo. (Mais il faudrait faire attention à la taille des dimensions aussi, parce que le résultat d'évaluation sera comparable à condition que la dimension possède la même taille pour les deux modèles. Donc j'ai mis 200 vectors-size pour les deux.) Cette démarche m'a permis d'obtenir avec succès deux séries de vecteurs de mots adaptés à mon corpus. Il est cependant intéressant de noter une différence dans le format des résultats : tandis que GloVe m'a fourni un fichier au format texte (.txt), Word2Vec a généré par défaut un fichier binaire, identifiable par son extension (.bin). Enfin, j'ai réussi à obtenir les plongements lexicaux des deux modèles, qui seront mises en oeuvre dans les prochaines étapes.

Pourtant, le processus de chargement du fichier binaire en Python est très différent et généralement plus complexe. De ce fait, j'ai modifié un peu les arguments dans le script **demo-word.sh** pour convertir le résultat des word embeddings en format texte, par exemple 0 suivi de l'argument -binary. Voici les arguments et les paramètres du modèle Word2Vec que j'ai mis. Comme résultat, nous avons eu deux fichiers textes nommés **vectors_glove.txt** et **vectors_word2vec.txt**.

```
1 make
2 time ./word2vec -train train.txt -output vectors.txt -cbow 1 -size 200 -
    ↪ window 8 -negative 25 -hs 0 -sample 1e-4 -threads 20 -binary 0 -iter 15
3 vectors.txt
```

ii.3 Recherche du Voisinage des Mots

Implémentation en Python

Nous allons maintenant implémenter les plongements de mots en Python pour générer la liste des mots les plus proches. Au lieu de directement utiliser le module *gensim* comme bibliothèque en Python, deux fichiers sont mis à la dispoistion pour nous dans la version démo des modèles, **distance.py** pour GloVe ainsi que **distance.c** pour Word2vec. Inspiré par les deux scripts de version demo, j'ai essayé tout d'abord de définir une fonction *read_embedding()* pour charger et préparer les données de vecteurs de mots pour qu'elles soient prêtes à être utilisées dans des calculs de similarité ultérieurs. Cela peut être réalisé simplement en parcourant le fichier en vecteur en boucle, comme illustré ci-dessous :

```python
import numpy as np

def read_embedding(path):
    vocab = []
    word_embedding = []

    # Ouvrir le fichier vecteur
    with open(path,'r',encoding='utf-8') as f:
        for i in f.readlines():
            cut_data = i.split()
            # Ajouter le premier element (mot) a la liste vocab.
            vocab.append(cut_data[0])
            # Convertir les elements restants (valeurs du vecteur) en float
               ↪ et les ajoute au plongement des mots.
            word_embedding.append(list(map(float,cut_data[1:])))

    # Creation d'un dictionnaire pour mapper chaque mot a son indice dans la
       ↪ liste vocab.
    word_to_idx = {}
    for number, word in enumerate(vocab):
        word_to_idx[word] = number

    return vocab, word_embedding, word_to_idx
```

Selon la version démo, nous pouvons constater que les mots les plus proches voisins sont trouvés par le calcul de **Similarité Cosinus**. Ce score donne la similarité de deux vecteurs à n dimensions en déterminant le cosinus de leur angle.

Soit deux vecteurs A et B, le cosinus de leur angle s'obtient en prenant leur produit scalaire divisé par le produit de leurs normes :

$$\cos\theta = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}.$$

Pour leurs normes, nous utiliserons généralement les normes euclidenne (appelé également norm L2) pour la normalisation. Il s'agit d'une mesure de la longueur ou de la magnitude d'un vecteur dans l'espace euclidien. Elle est définie comme la racine carrée de la somme des carrés de ses composants. Voici la formule de la norme euclidienne pour un vecteur $\mathbf{v}$ ayant n composants :

$$\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2 + \ldots + v_n^2}$$

Alors, j'ai pu implémenter le calcul de similarité cosinus entre un mot donné et tous les autres mots dans un ensemble de plongements de mots (word embeddings), ce qui est indiqué par la fonction *calculate_sim()* :

```python
def simcos(a,b):
    # produit scalaire de A et B
    dot = sum(a*b)
    # normalisation euclidienne des valeurs
    mod_a = sum(a**2)**0.5
    mod_b = sum(b**2)**0.5
    return dot/(mod_a*mod_b)

def calculate_sim(word_embedding, word_to_idx, word_name):
    # Convertir des plongements de mots en une matrice
    np_embedding = np.array(word_embedding)

    # Pour stocker les scores de similarite
    all_sim = []

```

```
16      # Iterer sur chaque vecteur de mots dans word_embedding
17      for i in range(len(word_embedding)):
18          # Calculer la similarite cosinus entre le vecteur du mot donnee
               ↪ chaque autre mot
19          all_sim.append([simcos(np_embedding[word_to_idx[word_name]],
               ↪ np_embedding[i]), i])
20
21      # Trier la liste en ordre decroissant de similarite cosinus
22      all_sim.sort(reverse=True, key=lambda x: x[0])
23
24      # Retourner la liste triee de paires [similarite_cosinus, indice]
25      return all_sim
```

J'ai ensuite défini une fonction assez simple *get_top20_words()* pour obtenir la liste des 20 voisinages du mot donné.

```
1  def get_top20_words(sims, vocab):
2      # Extraire les 20 premiers elements d'une liste de paires [similarite,
          ↪ indice] triee par similarite decroissante.
3      # Nous commencons a l'indice 1 car l'indice 0 serait le mot lui-meme.
4      top20 = sims[1:21]
5      # Creer une liste de mots correspondant aux indices trouves dans 'top20'.
6      words = [vocab[i[1]] for i in top20]
7      return words
```

Démonstration des Résultats

Je vous donne un aperçu pour visualiser le résultat généré dans la Table 7 et Table 8 qui correspondent aux deux modèles respectivement.Par souci de leur taille, ces deux tableaux se mettent dans l'appendice à la fin. En ce qui concerne le résultat, on pourrait remarquer que dans la liste des mots générée par Word2Vec, un bon nombre des mots affichés ne présentent aucun sens, même ils n'existent pas en français. C'est plutôt parce qu'Avec Word2vec, il ne s'agit donc plus de compter les voisins comme pour la matrice de co-occurence dans le GloVe, mais de les prédire. Plus précisément, en prenant l'argument - CBOW 1 lors de l'exécution du modèle préentraîné avant, on crée souvent une tâche de classification auto-supervisée à partir du texte brut : pour chaque token, on prend en entrée son contexte non-ordonné (bag of words) et en cible le token. Le réseau de neurones sera donc entraîné de façon à prédire un token selon son contexte.

iii Évaluation par Mesures de Distance

Après avoir établi les deux listes des mots dont les représentations sont les plus «proches» avec GloVe et word2ves pour 20 mots les plus fréquents dans mon corpus, on va ensuite évaluer les deux listes par plusieurs mesures de distance dans cette partie. Pourtant, ici on ne s'intérèsse pas forcément à la distance définie comme le nombre de mots en commun dans les deux listes. L'idée principale est de calculer la distance avec différents algorithmes pour chaque paire de mots similaires (un de GloVe et l'autre de Word2Vec).

Dans l'évaluation, on choisira 4 mesures de distances, qui sont respectivement distance de cosinus, distance euclidienne, distance de jaccard ainsi que distance manhattan. De manière plus générale, j'ai défini une fonction *calcul_distance()* assez globale pour se diriger vers la distance parmi celle de cosinus, euclidienne et manhattan. La raison pourlaquelle la distance de Jaccard n'est pas implémentée ici, c'est que la logique de cette distance se montre un peu différente. Je vous explique tout de suite.

```
1  def calcul_distance(distance, top20_glove, top20_w2v, glove_to_idx, word2
      ↪ vec_to_idx, glove_embedding, word2vec_embedding):
```

```
    distances = []

    # Iterer sur chaque paire de mots des listes top20_glove et top20_w2v
    for word_glove, word_w2v in zip(top20_glove, top20_w2v):
        # Convertir les embeddings de mots en arrays numpy pour le calcul
        embedding_glove = np.array(glove_embedding[glove_to_idx[word_glove]])
        embedding_w2v = np.array(word2vec_embedding[word2vec_to_idx[word_w2v
            ↪ ]])

        # Calculer la distance en fonction du type specifie
        if distance == 'cosinus':
            dist = cosinus_distance(embedding_glove, embedding_w2v)
            distances.append(dist)
        elif distance == 'euclidienne':
            dist = euclidienne_distance(embedding_glove, embedding_w2v)
            distances.append(dist)
        elif distance == 'manhattan':
            dist = manhattan_distance(embedding_glove, embedding_w2v)
            distances.append(dist)

    # Calculer et retourner la moyenne des distances apres avoir traite
        ↪ toutes les paires
    return sum(distances) / len(distances)
```

iii.1 Distance de Cosinus

Puisque j'ai déjà implémenté une fonction *simcos()* pour calculer la similarité cosinus, il s'avère donc facile de tester dans un premier temps la distance de Cosinus en tant que mesure de comparaison. Cela est dû à la formule du calcul pour les vecteurs A et B :

$$1 - \frac{A \cdot B}{\|A\|\|B\|}$$

Comme on l'a vu, la distance cosinus égale directement à 1 moins similarité cosinus. J'ai pu tout simplement établir une autre fonction *calculate_ cosine_ distance()* pour calculer ce genre de distance.

```
def cosinus_distance(a, b):
    return 1 - simcos(a, b)
```

Le résultat pourrait se trouver dans la Table 2.

TABLE 2 – 20 mots choisis et leurs Distances de Cosinus

Mot	Distance Cosinus	Mot	Distance Cosinus
the	0.9812561689149286	his	1.0159914758984896
and	1.0031730471796991	with	0.9665187201503747
of	1.0276621466686489	it	1.009514686168777
to	1.0164327174269185	for	0.9998368578794761
a	1.0011663647869011	is	0.997775457174637
in	1.0155728483929467	had	1.0681713261321353
I	0.9993834369145878	as	1.0138413326443112
that	1.0091609679623703	not	1.010682822465451
he	1.0288282055259972	you	0.9563079653282507
was	1.0384975363753897	at	1.0222651236301818

iii.2 Distance Euclidienne

La distance euclidienne est la distance en ligne droite entre deux points dans l'espace euclidien. Pour les vecteurs de mots, elle correspond à la distance en ligne droite entre leurs points de coordonnées respectifs. Quant à la formule, généralement si on a deux points $P = (p_1, p_2, ..., p_n)$ et $Q = (q_1, q_2, ..., q_n)$ dans un espace n-dimensionnel, la distance euclidienne d entre ces points est donnée par :

$$d(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

J'ai donc ajouté une fonction pour calculer la moyenne des distances euclidiennes

```
1 def euclidienne_distance(a, b):
2     return np.sqrt(np.sum((a - b) ** 2))
```

Je vous montre le résultat de distance euclidienne dans la Table 3.

TABLE 3 – 20 mots choisis et leurs Distances Euclidienne

Mot	Distance Euclidienne	Mot	Distance Euclidienne
the	11.704704322233525	his	14.400352057252954
and	14.730871758373913	with	9.52147478216178
of	9.45194895021987	it	14.2875354657349
to	14.141409515164261	for	13.683671818583045
a	12.917341133930648	is	15.133401763845942
in	12.747091941956768	had	16.66531701219059
I	17.744667541403665	as	13.873920951705227
that	14.134516354743326	not	15.838583846180393
he	16.159023448664435	you	17.991437712284917
was	19.12972153343509	at	13.450438788431342

iii.3 Distance Manhattan

Pour terminer, j'ai également testé la distance Manhattan pour une mesure de distance. S'agissant de la somme des valeurs absolues des différences entre les vecteurs, la distance de Manhattan, D peut être exprimée comme :

$$D(A, B) = \sum_i |A_i - B_i|$$

où A_i et B_i sont les valeurs des vecteurs A et B à la dimension i.

Voici mon implémentation en Python pour cette distance.

```
1 def manhattan_distance(a, b):
2     return np.sum(np.abs(a - b))
```

Le résultat tabulaire est dans la Table 4.

iii.4 Distance de Jaccard

contrairement aux autres distances, la distance de Jaccard n'est pas nécessaire de calculer une moyenne pour chaque paire de mots. Elle est néanmoins une méthode basant sur le rapport entre la taille de leur intersection et celle de leur union. Elle est calculée avec la formule suivante : d $= 1 - \frac{|A \cap B|}{|A \cup B|}$ où A et B sont deux ensembles distincts pour les deux modèles dans mon cas. $|A \cap B|$

TABLE 4 – 20 mots choisis et leurs Distances Manhattan

Mot	Distance Manhattan	Mot	Distance Manhattan
the	130.84665644999998	his	160.26171645000002
and	165.07689394999997	with	105.6588586
of	105.30939749999996	it	158.78744220000004
to	158.14025435	for	154.2064512
a	143.6115159	is	169.01238169999993
in	142.44725785	had	187.2405591
I	195.91381959999998	as	156.41275455
that	157.62532939999997	not	177.24302185000002
he	179.97922965000004	you	196.80317265
was	213.1321414	at	150.18609675

représente la taille de l'intersection de A et B (c'est-à-dire le nombre de mots communs à A et B). $|A \cup B|$ représente la taille de l'union de A et B (c'est-à-dire le nombre total de mots uniques dans les deux ensembles, sans doublons). Sa particularité est que, contrairement aux autres distances, il n'est pas nécessaire de calculer une moyenne pour chaque paire de mots. La fonction *jaccard_ distance()* est comme ceci :

```
1  def jaccard_distance(set1, set2):
2      intersection = len(set(set1) & set(set2))
3      union = len(set(set1) | set(set2))
4      return 1 - intersection / union
```

Après les avoir testés sur les deux listes, le résultat s'est présenté dans la Table 5.

TABLE 5 – 20 mots choisis et leurs Distances Jaccard

Mot	Distance Jaccard	Mot	Distance Jaccard
the	0.9189189189189189	his	0.9743589743589743
and	0.8235294117647058	with	0.9473684210526316
of	0.9189189189189189	it	0.8888888888888888
to	0.9473684210526316	for	0.9189189189189189
a	0.8888888888888888	is	0.9743589743589743
in	0.8571428571428572	had	0.7878787878787878
I	0.8888888888888888	as	0.75
that	0.8571428571428572	not	0.7878787878787878
he	0.8571428571428572	you	0.7878787878787878
was	0.8235294117647058	at	0.9743589743589743

iii.5 Vers un Résultat Comparable

Pour réaliser un résultat parallèlement comparable, j'ai ajusté un peu le code afin de permettre de présenter les mots triés par leur distance dans l'ordre accroissant dans la Table 6.

À première vue, il semble que l'ordre des mots dans les colonnes de la distance Manhattan et de la distance Euclidienne soient identiques. Cela est évidemment dû à leurs façons semblables de mesurer la distance entre deux points dans un espace, comme les indiqués les formules. La variabilité des mots selon les colonnes met également en évidence la sensibilité des différentes mesures de distance

à divers aspects de la relation entre les mots. Par exemple, la distance de Jaccard, qui se concentre sur les ensembles de caractéristiques, produit des résultats sensiblement différents de ceux basés sur des comparaisons vectorielles comme la distance cosinus.

TABLE 6 – les 20 mots triés par 4 mesures de distance

Distance Cosinus	**Distance Enclidienne**	**Distance Manhattan**	**Distance Jaccard**
you	of	of	as
with	with	with	had
the	the	the	not
is	in	in	you
I	a	a	and
for	at	at	was
a	for	for	in
and	as	as	that
that	that	that	he
it	to	to	a
not	it	it	I
as	his	his	it
in	and	and	the
his	is	is	of
to	not	not	for
at	he	he	to
of	had	had	with
he	I	I	his
was	you	you	is
had	was	was	at

iv Conclusion

En conclusion, ce travail m'a offert une opportunité d'explorer de manière approfondie des méthodes de calcul de distances ainsi que les modèles des word embeddings en TAL, notmmemment word2vec et GloVe. J'arrive à compprendre comment fonctionnement les plongements des mots générés par ces deux modèles-là et les appliquer dans le domaine du calcul des proximités du mot.

Au début, je me suis vraiment galéré dans quelques notions de la consigne du travail, tels que 'version démo entraîné par le corpus non-comparable' 'modèles pré-entraînés pour l'anglais' etc, c'est pourquoi j'ai essayé de lire et faire beaucoup de recherches sur ce sujet-là. Au fur et à mesure, j'ai pu presque comprendre l'idée de ce devoir. Pour aller plus loin, personnellement, il serait véritablement intéressant d'intégrer d'autres méthodes de mesure de la similarité entre les tokens, telles que la distance de Levenshtein et la distance SimHash.

i Appendice

TABLE 7 – liste des 20 mots les plus proche de mot selon GloVe

Mot	**20 voisins les plus proches du modèle GloVe**
the	'of', 'place', 'other', 'in', 'out', 'world,', 'way', 'edge', 'at', 'put', 'through', 'if', 'whole', 'rest', 'could', 'world', 'near', 'a', 'found', 'back'
and	'to', 'in', 'could', 'of', 'even', 'looked', 'them', 'then', 'have', 'had', 'time,', 'went', 'the', 'put', 'him,', 'felt', 'now', 'it,', 'back', 'if'
of	'the', 'sort', 'midst', 'presence', 'instead', 'life.', 'in', 'kind', 'out', 'those', 'life,', 'knowledge', 'and', 'think', 'one', 'rest', 'life', 'it.', 'pleasure', 'of,'
to	'able', 'seemed', 'give', 'to,', 'forced', 'going', 'show', 'unable', 'him', 'determined', 'and', 'take', 'speak,', 'tried', 'listen', 'have', 'him,', 'see', 'say', 'trying'
a	'sort', 'such', 'man', 'have', 'look', 'one.', 'kind', 'single', 'him', 'place', 'if', 'man,', 'the', 'fancy', 'with', 'another', 'person', 'gave', 'mere', "'a"
in	'the', 'of', 'and', 'midst', 'place', 'case', 'order', 'spite', 'found', 'faith', 'put', 'front', 'back', 'way', 'that', 'In', 'world', 'with', 'company', 'he'
I	'am', 'know', 'see', 'have', 'if', 'think', 'say', 'want', 'believe', 'wish', 'do', 'think,', 'tell', 'me.', 'mean', "don't", 'thought', 'say,', 'know.', 'should'
that	'that,', 'not', 'is', 'fact', 'know', 'this,', 'should', 'to', 'there', 'might', 'would', 'nothing', 'could', 'so', 'same', 'think', 'place', 'felt', 'case', 'also'
he	'said,', 'if', 'him', 'could', 'moment', 'saw', 'went', 'knew', 'when', 'man', 'him,', 'He', 'looked', 'replied.', 'thought', 'though', 'had', 'say', 'But', 'ought'
was	'It', 'There', 'was,', "wasn't", 'knew', 'there', 'felt', 'found', 'man', 'did', 'once', 'She', 'it', 'moment', 'had', 'that', 'am', 'thought', 'evidently', 'already'
his	'hand', 'head,', 'hands', "master's", 'him', "father's", "wife's", 'head', "brother's", 'own', 'way', 'But', 'arms,', "Levin's", 'hands.', 'own,', 'them', 'coat', 'heart.', 'pipe'
with	'filled', 'covered', 'mingled', 'delight', 'smile.', 'blood.', 'accordance', 'a', 'satisfied', 'compared', 'horror.', 'of', 'sort', 'pleasure,', 'angry', 'acquainted', 'those', 'have', 'met', 'up'
it	'think', 'is.', 'it,', 'put', 'it.', 'if', 'It', 'was,', 'thought', 'way', 'all.', 'did', 'matter', 'know', 'it ?', 'only', 'not', 'impossible', 'them', 'wish'
for	'only', 'sake', 'purpose', 'him', 'food', 'me', 'reason', 'if', 'see', 'it,', 'take', 'another', 'ever,', 'prepared', 'left', 'do', 'ever.', 'get', 'I', 'it.'
is	'It', "'This", 'is,', "'It", 'only', 'say,', 'that', 'not', 'seems', 'nothing', "'That", 'it', 'name', 'what', 'For', 'when', 'there', "'He", 'think', 'truth'
had	'been', 'already', 'having', 'just', 'gone', 'lately', 'given', 'him.', 'brought', 'taken', 'could', 'knew', 'come', 'entered', 'left', 'once', 'him,', 'had,', 'felt', 'discovered'
as	'possible.', 'regarded', 'well', 'soon', 'though', 'could,', 'possible,', "'As", 'usual,', 'As', 'could.', 'if', 'them', 'inasmuch', 'always', 'may', 'far', 'usual.', 'might', 'thought'
not	'did', 'could', 'do', 'does', 'think', 'know', 'will', 'wish', 'But', 'not,', 'should', 'must', 'if', 'it', 'have', 'would', 'are', 'see', 'understand', 'afraid'
you	'do', 'Do', 'know', 'tell', 'think', 'you,', 'You', 'know,', "'Do", "don't", 'if', 'wish', 'If', 'give', 'want', 'know.', 'see', 'you ?', 'think,', 'you.'
at	'once.', 'once,', 'once', 'looked', 'all.', 'least', 'looking', 'last.', 'least,', 'stared', 'glanced', 'At', 'look', 'moment,', 'all,', 'arrived', 'him,', 'wonder', 'him', 'near'

TABLE 8 – liste des 20 mots les plus proche de mot selon Word2Vec

Mot	20 voisins les plus proches du modèle Word2Vec
the	'of', 'in', 'and', 'which', 'a', 'by', 'their', 'his', 'from', 'this', 'on', 'The', 'Tokugawa', 'traverse', 'farthest', 'ladders', 'absorbed.', 'uninhabited', 'Shgun', 'Shgunate,'
and	'the', 'with', 'when', 'then', 'in', 'but', 'their', 'by', 'as', 'while', 'his', 'them', 'of', 'to', 'they', 'and,', 'for', 'her', 'which', 'him'
of	'the', 'in', 'which', 'preserves', 'affords', 'by', 'collective', 'regulating', 'analogous', 'primitive', 'production', 'extermination', 'namely,', 'sexes,', 'and', 'geographical', 'himin', 'unanimous', 'Sparta.', 'failures'
to	'would', 'that', 'for', 'him', 'should', 'but', 'not', 'it', 'might', 'and', 'he', 'if', 'could', 'me', 'dissuade', 'further.', 'I', 'so', 'them', 'him.'
a	'the', 'of', 'this', 'in', 'another', 'A', 'some', "'a", 'little', 'an', 'stout,', 'and', 'was', 'with', 'cookshop', 'every', 'duck.', 'uncomfortably', 'built.', 'his'
in	'of', 'the', 'and', 'by', 'into', 'with', 'In', 'a', 'which', 'from', 'his', 'chilling', 'on', 'customs,', 'namely,', 'constructed', 'befitting', 'Shgun', 'wherein', 'sumptuous'
I	'you', 'me', 'my', "'I", 'me,', '(I', 'me.', 'you,', 'am', 'myself,', 'myself', 'myself.', 'it', 'You', 'do', 'Ellen,', 'if', 'you.', 'not', 'your'
that	'it', 'but', 'this', 'to', 'not', 'he', 'so', 'when', 'if', 'beforehand,', 'would', 'even', 'for', 'what', 'only', 'because', 'have', 'be', 'which', 'should'
he	'him', 'his', 'him,', 'himself', 'she', 'him.', 'it', 'He', 'that', 'himself,', 'to', 'Smerdyakov', 'himself.', 'opponent', 'had', 'beforehand,', 'man', 'Lothario', 'fit.', 'and'
was	'had', 'seemed', 'felt', 'were', 'saw', 'was,', 'been', 'could', 'knew', 'appeared', 'remained', 'came', 'was.', 'became', 'disliked', 'arrival.', 'recollected', 'found', 'evidently', 'she'
his	'he', 'himself', 'him', 'him,', 'her', 'girded', 'and', 'with', 'the', 'His', "latter's", "son's", 'himself,', 'He', "Alyosha's", 'clenched', 'ached,', 'his,', 'him.', 'twitching'
with	'and', 'his', 'in', 'sarcastic,', 'knobby', 'the', 'tawny', 'undisguised', 'irrepressible', 'wrathful', 'beaming', 'clenching', 'especial', 'a', 'flushed,', 'sparkling', 'unmoved,', 'mourning,', 'of', 'willow,'
it	'that', 'it,', 'he', 'it.', 'I', 'not', 'so', 'if', 'me', 'to', 'but', 'him', 'be', 'she', 'this', 'what', 'would', 'should', 'beforehand', 'somehow.'
for	'to', 'but', 'that', 'and', 'so', 'for.', 'only', 'would', 'if', 'not', 'because', 'should', 'ransomed', 'any', 'I', 'all', 'reward', 'it', 'faithfully', 'sorrow.'
is	'is,', 'are', 'has', 'NOT', 'may', 'considers', 'will', 'can', 'does', 'rests', 'argues', 'requires', 'exists', 'is,', 'belongs', 'proves', 'just.', 'becomes', 'Impossible.', 'denies', 'explains'
had	'was', 'been', 'having', 'arrival.', 'had,', 'were', 'arrival,', 'could', 'knew', 'recollected', 'felt', 'previously', 'seemed', 'lately', 'would', 'before,', "hadn't", 'he', 'visit,', 'that'
as	'so', 'and', 'though', 'that', 'but', 'if', 'could,', 'when', 'could.', 'as,', 'to', 'readily', 'nevertheless', 'possible.', 'thoroughly,', 'possible,', 'well', 'it', 'very', 'inasmuch'
not	'that', 'but', 'would', 'it', 'to', 'never', 'be', 'if', 'not.', 'not,', 'I', 'only', 'should', 'did', 'otherwise.', 'so', 'what', 'you', 'do', 'no'
you	'I', 'you,', 'your', 'do', 'You', 'you.', 'you...', 'me', 'myself?', "don't", "'I", 'yourself?', 'you?', 'yourself', 'yourself,', 'papa', 'yourselves', 'tell', 'yourself.', 'am'
at	'At', 'intently', 'stupidly', 'uneasily', 'at.', 'after', 'at,', "'At", 'on', 'fearfully', 'intently,', 'saw', 'was', 'anxiously', 'inquisitively', "'at", 'blankly', 'wildly', 'undisguised', 'with'