Mastère spécialisé IA

Semantic Textual Similarity Project

Réalisé par

Lily Daganaud
Sarah Garcia
Inès Lalou
Candice Bouquin-Renoux

Sous la direction de : Aina Garí Soler

2024/2025

Table des matières

Introduction

La Semantic Textual Similarity (STS) est une technique essentielle en traitement automatique du langage naturel (NLP), visant à mesurer le degré de similitude entre deux textes. Les systèmes STS produisent un score continu allant de 0 à 1. Un score de 0 indique une absence totale de lien sémantique et à l'inverse un score de 1 indique une équivalence parfaite entre deux phrases.

Les prédictions issues de modèles STS ont des applications variées en NLP, telles que la synthèse automatique, la traduction automatique ou encore la vérification de l'équivalence sémantique entre une requête et une correspondance potentielle. Nous trouvions donc intéressant de découvrir de manière plus approfondie ces outils et techniques.

L'un des principaux défis du projet, mais aussi du traitement des langues en général, réside dans la bonne compréhension de la sémantique des phrases : une même idée peut être exprimée de manière très différente, tandis que des formulations presque identiques peuvent véhiculer des significations distinctes. Une simple analyse lexicale, basée sur les mots isolés ou leur fréquence, est donc insuffisante pour capturer ces nuances complexes.

Dans ce projet, nous avons choisi de nous concentrer sur le domaine biomédical, un champ riche en enjeux : l'importance cruciale de l'exactitude sémantique dans ce contexte. Tout au long de ce travail, nous avons cherché à expérimenter, évaluer et combiner différentes approches, dans le but d'obtenir des modèles capables de noter la similitude des phrases, et de les comparer aux scores initiaux. Afin d'organiser notre travail de manière efficace, nous avons procédé en plusieurs étapes. Dans un premier temps, nous avons travaillé ensemble pour nous approprier le sujet, explorer le corpus et le manipuler à l'aide d'outils d'analyse textuelle. Ainsi, nous avons toutes participé à la phase de prétraitement des données. Chacune a développé et analysé individuellement une ou deux caractéristiques (features) spécifiques. Sarah a pris en

charge l'étude des combinaisons de ces caractéristiques. Par la suite, Candice et Lily ont travaillé sur la deuxième partie (Représentation distributionnelle et distribuée des mots), Inès et Sarah sur la troisième partie (Réseaux de neurones), et Inès, Candice et Lily sur la quatrième partie (Biomedical Dataset).

Ce rapport détaille l'évolution de notre travail, les méthodologies explorées et les résultats obtenus, tout en mettant en lumière les défis rencontrés et les enseignements tirés de cette expérience.

Partie 1 : Caractéristiques linguistiques et textuelles simples

1 Présentation du jeu de données

Le jeu de données utilisé dans ce projet est conçu pour évaluer la similarité sémantique entre paires de phrases. Chaque paire est associée à un score de similarité compris entre 0 et 5, où 0 indique une absence totale de similarité sémantique et 5 correspond à une équivalence parfaite. Les scores ont été évalué par des personnes puis une moyenne a été réalisé pour obtenir le score final. Quelques exemples tirés du jeu de données sont présentés dans la Table 1.

Phrase 1	Phrase 2	Score de similarité
A plane is taking off	An air plane is taking off.	5.0
A man is playing a large flute.	A man is playing a flute	3.8
Three men are playing chess.	Two men are playing chess.	2.6

TABLE 1 – Exemple de 3 phrases tirées du dataset *train* avec leur score de similarité.

Le jeu de données est divisé en trois sous-ensembles pour faciliter l'entraînement, la validation et le test des modèles. Voici la répartition des paires de phrases :
- Train : 5,749 paires
- Dev : 1,500 paires
- Test : 1,379 paires

Dans un premier temps nous avons exploré le dataset. Les paires de phrases sont relativement courtes, avec les longueurs moyennes suivantes :
- Train : S1 = 9.95 S2 = 9.94
- Dev : S1 = 11.42 S2 = 11.34
- Test : S1 = 9.82 S2 = 9.80

où S1 représente la longueur moyenne des phrases 1 et S2 la longueur moyenne des phrases 2.

Ces résultats mettent en évidence l'importance de préserver un équilibre dans le prétraitement des données. Étant donné que les phrases sont courtes, l'information essentielle y est condensée dans un nombre restreint de mots. Un prétraitement trop restrictif pourrait donc entraîner une perte d'information critique, compromettant ainsi la qualité des analyses et des modèles développés.

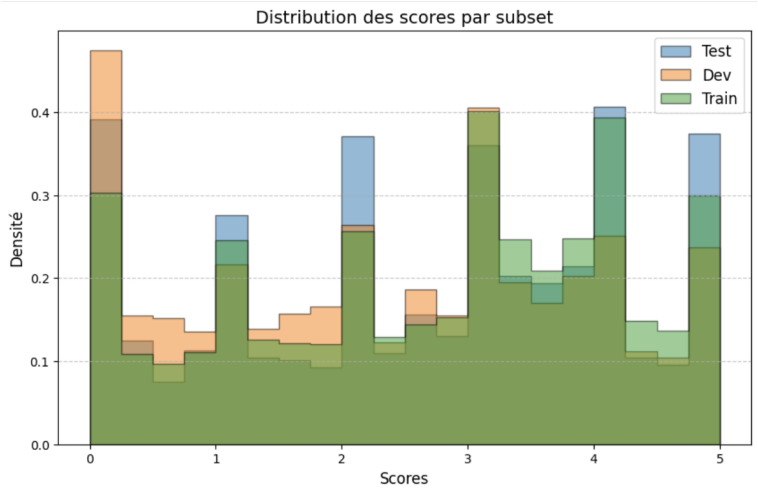

FIGURE 1 – Histogramme représentant la distribution des scores de similarité pour les subset *train*, *dev* et *test*.

La figure 1 révèle que les scores de similarité sémantique suivent une répartition non uniforme, avec une concentration marquée autour des scores clés tels que 0, 3, 4 et 5. Les trois subsets (*Train*, *Dev*, et *Test*) présentent des pics cohérents à ces mêmes valeurs, ce qui indique une répartition comparable des niveaux de similarité dans chaque subset. Ainsi, on pourra réaliser l'étude sans devoir accorder d'attention (poids) particulière aux classes moins représentatives.

Cette similarité des distributions garantie que les données d'entraînement et d'évaluation proviennent de la même distribution. C'est un aspect important pour évaluer correctement la performance des modèles sans introduire de biais lié à une mauvaise représentativité des données. De plus, cette observation est cruciale pour concevoir des modèles capables de capturer ces variations et de traiter efficacement

les cas intermédiaires.

2 Modèles et évaluations

Afin d'évaluer l'ensemble de nos modèles et leurs caractéristiques, nous avons décidé d'utiliser à la fois des régressions linéaires et des random forests. Certaines caractéristiques présentent des tendances plus ou moins linéaires, ce qui rend essentiel l'analyse de la dispersion des résultats de nos caractéristiques par rapport aux scores de similarité (via des scatter plots). Cela nous permet d'adapter notre analyse en fonction des relations observées.

Nous avons sélectionné les métriques suivantes pour évaluer les performances de nos caractéristiques et modèles :
- **Coefficient de Spearman** : Mesure la corrélation monotone entre les prédictions du modèle et les valeurs réelles, utile pour évaluer si le classement relatif des paires est cohérent.
- **Coefficient de Pearson** : Évalue la corrélation linéaire entre les prédictions et les valeurs cibles, pertinent pour détecter des relations proportionnelles.
- **MSE (Mean Squared Error)** : Fournit une mesure globale de l'erreur quadratique moyenne entre les prédictions et les valeurs réelles, permettant de quantifier directement l'écart entre les deux.

Ces métriques ont été choisies pour leur complémentarité. Spearman et Pearson permettent d'analyser respectivement les relations structurelles et linéaires [1] et le MSE offre une évaluation quantitative de l'erreur des prédictions. Cette combinaison nous a permis d'évaluer nos modèles de manière exhaustive, en tenant compte à la fois des corrélations et des erreurs absolues.

D'après les résultats (Figure 2), les deux coefficients sont très proches l'un de l'autre. C'est pourquoi, le choix a été fait pour la suite du projet d'évaluer les différents modèles avec le coefficient de Pearson et le MSE.

```
Results obtained on the dev set:
#Pearson coefficient :
[PearsonRResult(statistic=0.6668107015744721,
pvalue=1.507176205929719e-193),
#Spearman coefficient :
SignificanceResult(statistic=0.6741599806340367,
pvalue=2.234780556855881e-199)]

Results obtained on the test set:
#Pearson coefficient :
[PearsonRResult(statistic=0.5997660486084755,
pvalue=1.7373027828255158e-135),
#Spearman coefficient :
SignificanceResult(statistic=0.5911318709155592,
pvalue=1.0230591304169313e-130)]
```

FIGURE 2 – Résultats de l'évaluation entre la baseline et le gold standart avec les coefficients de Pearson et de Spearman

3 Modèle de référence

Pour commencer, nous avons utilisé un modèle de baseline basé sur le comptage de mots, servant de point de départ pour évaluer nos approches plus complexes par la suite. L'objectif est de trouver un modèle pouvant surpasser la baseline en capturant des relations sémantiques plus riches qu'avec un simple chevauchement de mots.

Le modèle baseline repose sur un modèle de régression linéaire qui utilise une seule caractéristique mesurant le chevauchement des mots entre deux phrases, et se basant sur la similarité de Jaccard. Pour chaque paire de phrases (s1, s2), nous utilisons un CountVectorizer avec l'option binary=True. Cette option encode uniquement la présence ou l'absence de mots dans chaque phrase, ce qui est essentiel pour calculer la similarité de Jaccard. Les phrases sont ainsi converties en représentations binaires, où chaque mot est soit présent, soit absent dans une phrase.

La similarité de Jaccard est définie comme le rapport entre la taille de l'intersection et celle de l'union des ensembles de mots des deux phrases.

$$J(A, B) = \frac{\mid A \cup B \mid}{\mid A \cap B \mid} \tag{1}$$

Ce score mesure dans quelle proportion les deux phrases partagent des mots communs par rapport au total des mots uniques présents dans les deux phrases. Les scores de similarité de Jaccard du jeu de donnée *train* sont utilisés pour entrainer un modèle de régression linéaire afin de prédire les scores de similarité annotés dans le jeu de données. Puis, nous avons évaluer les performances de ce modèle à partir des scores de similarité Jaccard du *dev* set puis du *test* set. Les résultats sont présentés dans la Table 2.

	dev	test
Score de Pearson	0.66	0.59
MSE	0.05	0.06

TABLE 2 – Résultats pour les parties *dev* et *test* du dataset.

Les corrélations sont modérées sur les deux subsets d'évaluation (*dev* et *test*) et le MSE est faible. Cela indique que la feature de chevauchement des mots a une certaine capacité à prédire les scores de similarité, mais reste loin d'être parfaite. La relation entre cette feature et les scores de référence est quasi-linéaire. La légère diminution des performances sur le set de test (Pearson = 0.5998) par rapport au set de validation est attendue. Cet écart est naturel, en raison de la variabilité des données, mais reste faible, ce qui montre une bonne généralisation.

Les résultats montrent que la feature de chevauchement entrainé avec le modèle de regression linéaire des mots constitue une baseline efficace pour la tâche de similarité sémantique textuelle. Cependant, bien que ces résultats soient acceptables, cette approche repose uniquement sur le chevauchement lexical et ne capture pas les relations sémantiques plus complexes, telles que la synonymie, les paraphrases, ou les structures syntaxiques différentes exprimant un même sens. Par conséquent, les phrases qui ne partagent aucun mot en commun obtiendront un score de similarité de Jaccard égal à 0, même si elles sont sémantiquement liées.

Il est donc essentiel d'améliorer le modèle pour qu'il prenne en compte non seulement les mots identiques, mais également des relations plus profondes, comme les

similarités sémantiques et les contextes lexicaux. Pour ce faire, nous allons explorer des features supplémentaires afin d'intégrer des informations sémantiques plus riches et mettre en place un prétraitement pour préparer les donnéess.

4 Prétraitement des données textuelles

Dans le cadre de notre projet, nous avons réfléchi à la meilleure méthode pour prétraiter les données textuelles afin d'optimiser les résultats des modèles de traitement automatique du langage naturel (NLP). L'apprentissage automatique repose fortement sur la qualité des données qui y sont introduites et le prétraitement des données joue donc un rôle crucial pour garantir la précision et l'efficacité du modèle.

Nous avons testé plusieurs types de prétraitements intégrant des techniques différentes. Le but était de trouver celui qui serait le plus adapté à notre tâche. Nous avons ainsi exploré plusieurs configurations : avec ou sans post-tagging, en utilisant la lemmatisation avec post-tagging, en appliquant le stemming, en conservant ou non la négation. Ces prétraitements ont été testés sur certaines de nos variables, et des modèles de régression linéaire ou de Random Forest ont été construits afin de déterminer le modèle le plus adapté.

Dans un premier temps, un prétraitement a été réalisé en retirant uniquement les stopwords et la ponctuation (incluant les caractères particuliers tels que "ë" ou "ä") et en tokenisant les phrases. Enfin, tous les caractères majuscules ont été transformés en minuscules. Dans un deuxième temps, la lemmatisation et le stemming des mots ont été ajoutés au prétraitement. Toutes les variables ont été testées avec et sans lemmatisation.

4.1 Nettoyage des données textuelles

La première étape consiste à nettoyer le texte afin de le normaliser et d'éliminer les éléments sémantiquement inutiles. Pour cela nous avons exploré les paires de phrases afin de déterminer s'il y avait des majuscules, noms propres, quels caractères spéciaux étaient présent etc... Ces actions incluent :
- Conversion en minuscules : Toutes les phrases ont été converties en minuscules pour éviter des doublons (par exemple, "Free" et "free") et garantir la cohérence. Cependant il serait peut-être intéressant de les garder dans le dataset biomedical,

qui doit comporter des noms de molécule ou de maladie.
- Suppression des caractères non textuels : Les caractères non alphabétiques, tels que la ponctuation, les symboles ou les chiffres, ont été supprimés. Cela permet de se concentrer sur les mots porteurs de sens et d'éliminer le bruit dans le corpus.
- Suppression des stopwords : Nous avons retiré les mots fonctionnels (par exemple, "the", "is", "and"), car ils n'apportent généralement pas de valeur ajoutée à l'analyse sémantique. Pour cela nous avons utilisé stopwords.words('english'). De plus, nous avons regardé l'impact des négations dans la sémantique d'une phrase. Pour cela nous avons appliqué un preprocess qui gardait les termes de négation (not, no, never, n't), et un autre qui ne les gardait pas.

Nous avons remarqué qu'il y avait un quart des phrases qui comportait une négation :
- Nombre de paires avec une négation : 989
- Nombre de paires sans négation : 4760

Par la suite nous avons étudier aux travers de nos features l'impact quand nous prenions ou non en compte la négation.

5 Lemmatisation ou stemming ?

Pour simplifier les analyses textuelles et rendre nos modèles plus performants, il est nécessaire de réduire les mots à une forme commune. Nous avons comparé deux techniques populaires : le stemming et la lemmatisation.
- **Stemming** : Cette méthode tronque les mots à leur racine de manière algorithmique, sans tenir compte du contexte grammatical. Par exemple, "running", "runner" et "ran" seraient tous réduits à "run". Bien que rapide, cette approche manque souvent de précision et peut produire des formes de mots qui n'existent pas dans le langage naturel.
- **Lemmatisation** : Contrairement au stemming, la lemmatisation repose sur une analyse linguistique approfondie pour transformer un mot en sa forme canonique (lemme). Par exemple si le mot "running" est taggé en tant que verbe son lemme est "run". Pour "better" (qui est un adjectif) sont lemme est "good".

La lemmatisation offre une meilleure précision, surtout lorsqu'elle est combinée avec le POS tagging, qui identifie le rôle grammatical de chaque mot (nom, verbe, adjectif, etc.). Cela garantit que le bon lemme est sélectionné en fonction du contexte.

Pour la suite du projet, nous avons opté pour un prétraitement utilisant la lemmatisation avec POS tagging, avec et sans négation, car ces approches garantissent une représentation plus fidèle des données textuelles. Ce choix a permis d'améliorer la qualité des analyses et des modèles construits, comme mentionné précédemment. Après avoir analysé l'ensemble des caractéristiques (features), nous avons finalement retenu le prétraitement incluant la négation, car il offrait les meilleurs résultats. Cependant, le choix entre le stemming et la lemmatisation reste dépendant du contexte et des contraintes de performance spécifiques à chaque projet.

6 Développement de 5 features et analyse

La sélection et la conception de caractéristiques (features) représentent une étape importante car elles nous ont permis de parcourir un certain nombre de techniques différentes afin d'étudier la vraisemblance entre deux phrases. L'importance des caractéristiques dans le NLP réside dans leur capacité à capturer des relations complexes entre les éléments textuels tels que le vocabulaire, les structures syntaxiques, et les relations sémantiques. Nous voulions explorer le plus de techniques diverses afin de capturer un maximum de relations complexe qui sont des éléments clefs aux performances du modèle.

Pour répondre à ces objectifs, nous avons décidé de développer cinq caractéristiques. Nous avons choisi TF-IDF car il est utile pour capturer les similarités lexicales globales, WordNet pour enrichir les représentations au-delà des correspondances lexicales exactes, tout comme word Alignment qui évalue leur correspondance sémantique au niveau lexical. N-Gram nous permet d'ajouter de la contextualisation dans notre comparaison phrases car il prend en compte la succession de mot. Et enfin POS Tag nous permet de prendre en compte les similarités grammaticales.

6.1 WordNet

En exploitant WordNet, une base de connaissances lexicales reconnue, nous avons conçu des caractéristiques permettant d'évaluer les relations sémantiques entre les phrases. Nous nous sommes concentrés sur la présence de synonymes et d'hyperonymes. Ce choix repose sur l'hypothèse que la prise en compte des hyperonymes, en plus des synonymes, pourrait enrichir la représentation sémantique et améliorer

les performances de notre modèle.

Nous avons estimé qu'il était essentiel d'aller au-delà des simples correspondances synonymiques. C'est pourquoi nous avons cherché à combiner les informations issues des synonymes et des hyperonymes, afin de fournir davantage de contexte et à capturer des relations sémantiques plus complexes. Cela pouvait conduire à de meilleurs scores sur les tâches envisagées.

Nous avons donc developpé la fonction *compute_wordnet_scores()* qui évalue et attribue des scores en fonction des relations sémantiques identifiées dans WordNet. Elle va calculer deux types de scores : le score de synonymie, qui évalue la proportion de mots ayant des significations similaires, et le score d'hyperonymie, qui identifie les relations entre un mot et ses catégories générales. Pour chaque paire de phrases, les mots sont comparés pour détecter des correspondances via leurs synonymes et hyperonymes. Ces scores sont ensuite normalisés par le nombre total de mots uniques dans les phrases, offrant une évaluation relative de leur similarité.

Phrase 1	Phrase 2	Syno* score	Hyper** score	Syno count	Hyper count
['plane','take']	['air','plane','take']	0.66	0.33	2	1
['man','play','flute']	['man','play',large','flute']	0.75	0.5	3	2

TABLE 3 – Exemples des scores et nombre de synonymes et d'hypernymes tirés du dataset *train*. *Synonyme, **Hypernyme

Par la suite, l'objectif a été d'identifier visuellement si une relation linéaire ou une autre forme de dépendance existe, ce qui pourrait guider le choix d'un modèle approprié pour exploiter ces caractéristiques dans les tâches de traitement du langage (figure 3). On observe une relation globalement croissante et plutôt linéaire pour les scores de synonymes, ce qui est logique : plus le nombre de synonymes entre deux phrases est élevé, plus leur score de similarité est important. En revanche, cette relation semble moins évidente pour les hyperonymes, et ne paraît pas réellement linéaire. Nous avons donc décidé de tester deux modèles, une régression linéaire et un random forest. Les performances ont été évaluées en considérant les scores individuellement ainsi que combinés.

Une première phase exploratoire a été réalisée afin de voir s'il était plus intéressant de combiner les hyperonymes et les synonymes ou de garder seulement l'un ou l'autre

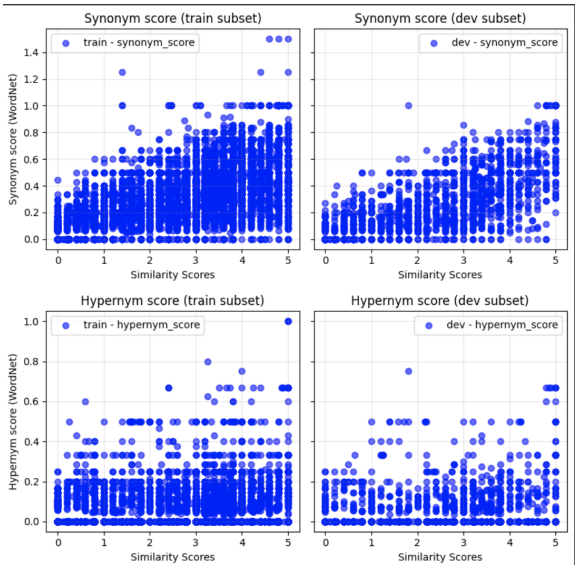

FIGURE 3 – Plot des scores de similarité (gold standart) en fonction des scores obtenus pour les synonymes et hypernymes, pour les parties *dev* et *train* du dataset.

(figure 4). Les résultats nous montrent que les scores obtenus en utilisant uniquement les synonymes sont significativement meilleurs que ceux obtenus avec les hypernymes. Cela peut s'expliquer par le fait que les synonymes capturent directement la similarité lexicale entre les phrases. En revanche, les hyperonymes, bien qu'importants pour des tâches plus techniques (classification conceptuelle) n'apportent pas suffisamment d'informations spécifiques pour mesurer des similarités lexicales directes. Cette faiblesse peut être due à plusieurs raisons :
- Les hypernymes ne montrent pas directement la "proximité" lexicale entre deux phrases mais plutôt une relation hiérarchique (ex. : "chien" est un hypernyme de "animal").
- Dans les phrases courtes ou spécifiques, les hypernymes peuvent être trop généraux pour apporter une valeur discriminante dans la similarité.

Résultats de la régression linéaire (corrélation de Pearson) :

	df_sans_negation			df_negation		
	synonym	hypernym	combined	synonym	hypernym	combined
dev	0.692	0.204	0.701	0.692	0.203	0.700
test	0.612	0.126	0.615	0.610	0.125	0.613

Résultats de la forêt aléatoire (corrélation de Pearson) :

	df_sans_negation			df_negation		
	synonym	hypernym	combined	synonym	hypernym	combined
dev	0.716	0.161	0.704	0.714	0.169	0.701
test	0.619	0.130	0.621	0.617	0.136	0.620

FIGURE 4 – Résultats de la comparaison entre les synonymes, les hyperonymes ainsi que leur combinaison pour le dataset avec ou sans la négation et pour un modèle de regression linéaire et un Random Forest.

Cependant les résultats combinés des synonymes et des hypernymes offrent de légères améliorations par rapport à l'utilisation des synonymes seuls. Cela suggère que, bien que les hypernymes aient une valeur limitée individuellement, ils peuvent compléter les synonymes pour améliorer les prédictions du modèle. Nous allons donc nous orienter vers une combinaison des caractéristiques hyperonymes/synonymes pour poursuivre les tests avec l'ensemble des caractéristiques et comparer chaque feature.

Nous constatons que WordNet produit des résultats légèrement meilleurs que notre modèle de référence, ce qui est attendu. Cela apporte une information sémantique supplémentaire au modèle de base, enrichissant ainsi l'analyse.

6.2 WordAlignement

Pour évaluer la similarité sémantique entre deux phrases, nous avons développé une fonction appelée *align_ word_ bidirectional()*. Cette fonction exploite un modèle d'embeddings lexicaux pour analyser les relations entre les mots des deux phrases. Elle commence par filtrer les lemmes pour ne conserver que ceux présents dans le modèle d'embeddings. Les vecteurs d'embeddings des mots sont ensuite extraits et, si activé, normalisés pour faciliter les comparaisons. La fonction calcule une matrice de similarité en appliquant le produit scalaire entre les vecteurs des mots des deux phrases. Cette matrice est utilisée pour identifier, dans les deux directions, les alignements les plus proches entre les mots des phrases. Elle vérifie si la similarité entre

chaque paire de mots dépasse un seuil prédéfini, ce qui permet de différencier les alignements significatifs des correspondances faibles. À partir de ces alignements, la fonction calcule quatre métriques principales :
- **Alignment Score Thresholded** : proportion de mots alignés avec une similarité dépassant le seuil, reflétant le degré global d'alignement.
- **Average Similarity No Threshold** : mesure la similarité moyenne maximale entre tous les mots, offrant une vue globale sans appliquer de seuil.
- **Average Similarity Thresholded** : évalue la qualité des alignements en se concentrant sur les correspondances dépassant le seuil.
- **Weighted Alignment Score** : intègre la quantité et la qualité des alignements, pondérant les similarités alignées par le nombre total de mots.

Nous avons fait une approche bidirectionnelle, car nous avions remarqué que cela capturait mieux les asymétries potentielles dans la structure des phrases. Enfin, bien que nous ayons initialement envisagé d'utiliser une solution existante trouvée sur GitHub, des incompatibilités liées aux versions de Python et des bibliothèques nous ont conduits à concevoir cette fonction sur mesure, mieux adaptée à nos besoins spécifiques.

Pour l'analyse, nous avons testé plusieurs paramètres (figure 5). Le premier concernait la normalisation des vecteurs d'embeddings. Nous nous attendions à ce que les performances soient meilleures avec la normalisation, car elle élimine l'influence de la magnitude des vecteurs et permet de calculer des similarités cosinus cohérentes. Sans normalisation, les normes différentes des vecteurs faussent les calculs de similarité, dégradant ainsi les performances du modèle.Le second paramètre étudié était le seuil de similarité (threshold). Nous avons testé des seuils de 0.6 et 0.8 pour évaluer leur impact sur les métriques. Un seuil élevé conserve uniquement les alignements les plus significatifs, améliorant la qualité des mesures, mais peut limiter la généralisation si trop restrictif. Les tests ont été effectués sur tous les jeux de données, y compris ceux avec et sans négations. Les résultats obtenus avec la régression linéaire confirment que la normalisation des vecteurs est essentielle. Sans normalisation, le produit scalaire est influencé par la longueur des vecteurs au lieu de leur orientation, ce qui dégrade les calculs de similarité. Pour la comparaison des modèles, nous avons choisi la combinaison normalisant les vecteurs et utilisant un threshold de 0.8. Les métriques *weighted_ alignment_ score()* et *alignment_ score_ thresholded()* se sont avérées les plus performantes. Ces métriques équilibrent le nombre de lemmes alignés et la force de ces alignements, offrant une mesure robuste et fiable de la similarité entre les phrases.

	Dataset	lormaliz	rity Thre	Feature	#	Pearson Dev	P-value Dev	#	Pearson Test	P-value Test
5	with_negations	TRUE	0.6	average_similarity_no_threshold		0,587	1.0236355427927546e-139		0,465	7.657415020291057e-75
13	with_negations	TRUE	0.8	average_similarity_no_threshold		0,587	1.0236355427927546e-139		0,465	7.657415020291057e-75
1	preprocessed	TRUE	0.6	average_similarity_no_threshold		0,588	4.564472074854202e-140		0,472	2.58668132327306e-77
9	preprocessed	TRUE	0.8	average_similarity_no_threshold		0,588	4.564472074854202e-140		0,472	2.58668132327306e-77
4	with_negations	TRUE	0.6	alignment_score_thresholded		0,613	3.189587416746936e-155		0,473	6.678230458242783e-78
0	preprocessed	TRUE	0.6	alignment_score_thresholded		0,612	7.335518910904 1025e-155		0,482	3.0913499410060303e-81
7	with_negations	TRUE	0.6	weighted_alignment_score		0,636	6.78849506743253e-171		0,507	3.807305518223055e-91
3	preprocessed	TRUE	0.6	weighted_alignment_score		0,635	4.380447988201287e-170		0,514	7.465503344240853e-94
12	with_negations	TRUE	0.8	alignment_score_thresholded		0,656	5.635602598723067e-185		0,539	1.3235982834317802e-104
8	preprocessed	TRUE	0.8	alignment_score_thresholded		0,656	2.0749258795772314e-185		0,542	2.6994916308092623e-106
15	with_negations	TRUE	0.8	weighted_alignment_score		0,657	5.335393852132867e-186		0,546	6.019700639502849e-108
11	preprocessed	TRUE	0.8	weighted_alignment_score		0,657	3.134816701892596e-186		0,548	9.849962398098342e-109

FIGURE 5 – Résultat sur les différents paramètres

6.3 TF-IDF

Le TF-IDF est une feature simple qui permet de savoir à quel point un mot est important dans un texte. Un mot très fréquent dans l'ensemble du corpus de texte ne permet de différencier les textes entre eux, c'est pourquoi on prend l'inverse qui permet de donner plus d'importance aux mots moins présents (et donc plus représentatifs d'un texte en particulier).

Nous avons obtenus les scores suivant avec cette feature :

	dev	test
Score de Pearson	0.714	0.608
MSE	0.04	0.05

TABLE 4 – Résultats pour les parties *dev* et *test* du dataset sur un modèle de régression linéaire.

	dev	test
Score de Pearson	0.712	0.605
MSE	0.04	0.05

TABLE 5 – Résultats pour les parties *dev* et *test* du dataset sur un modèle de Random Forest.

On peut voir que les résultats sont identiques entre les deux modèles. De plus, on peut voir que les résultats sont bons

6.4 POS tag Overlap

Cette fonctionnalité va identifier les rôles grammaticaux des mots dans chaque phrase et calculer le taux de overlap des étiquettes (POS tags), c'est-à-dire, la proportion d'étiquettes communes parmi toutes celles trouvées.

Voici les scores qu'on obtient lorsqu'on utilise cette feature individuellement sur un modèle de régression linéaire :

	dev	test
Score de Pearson	0.23	0.14
MSE	0.08	0.09

TABLE 6 – Résultats pour les parties *dev* et *test* du dataset sur un modèle de régression linéaire.

Et voici, le résultat sur un modèle random forest :

	dev	test
Score de Pearson	0.21	0.20
MSE	0.08	0.08

TABLE 7 – Résultats pour les parties *dev* et *test* du dataset sur un modèle Random Forest.

On observe de meilleurs résultats avec le modèle linéaire sur le *dev* set et avec le Random Forest sur le *test* set. On observe également que les performances ne sont pas très élevées, ce qui semble logique, car le POS tagging ne prend en compte que les similitudes grammaticales entre phrases, n'apportant pas d'informations sémantiques.

6.5 N-gram

En utilisant une méthode basée sur n-gram overlap : (ex : bigram overlapp, trigram overlap...) nous pouvons obtenir de bons indicateurs de similarité sémantique entre deux phrases.

N-gram overlap consiste à comparer le nombre de séquences de n tokens consécutifs commun aux deux phrases. Puis, nous conservons comme caractéristique d'intérêt le rapport entre la somme des n-grams communs aux deux phrases et le maximum de n-grams possibles (au vu de la taille des phrases). Cette caractéristique est d'autant plus pertinente lorsqu'elle est utilisée sur un jeu de données lemmatisé car nous pouvons ainsi avoir une comparaison entre les deux phrases qui généralise mieux (moins de variabilité du vocabulaire) et qui permet ainsi de mieux comparer le sens sémantique des phrases. Dans la création de cette caractéristique, il a été important de prévenir le cas où la phrase n'est pas de longueur proportionnelle à la taille de n-gram. Par exemple, le cas où une des phrases à comparer est composée de 3 tokens et qu'on s'intéresse aux bigram. La solution a été de rajouter en début de phrase n-1 token '<sos>' et en fin de phrase n-1 token '<cos>'. Ainsi, tous les éléments de la phrase seront pris en compte.

Etant donné la taille des phrases du corpus, nous avons choisis de nous intéresser uniquement aux bi-gram (séquence de 2 tokens consécutifs communs) et aux tri-gram (séquence de 3 tokens consécutifs communs). Cette feature est d'un grand intérêt car intuitivement, on s'attend à ce que plus deux phrases aient une proportion importante de n-gram commun, plus ces deux phrases sont proches sémantiquement.

La première étape dans l'évaluation des features n-gram a été de représenter les labels à prédire en fonction de ces features afin d'avoir une première idée intuitive de leur lien (Table 6).

On remarque une légère tendance linéaire. Ainsi, nous testerons deux modèles : un premier modèle de régression linéaire et un deuxième modèle plus complexe de Random Forest Regression.

Afin d'étudier les performances de cette feature, les performances des features bigram et trigram overlap combinées et prises individuellement pour prédire le score de similarité ont été étudiées sur différents modèles pour le test set.

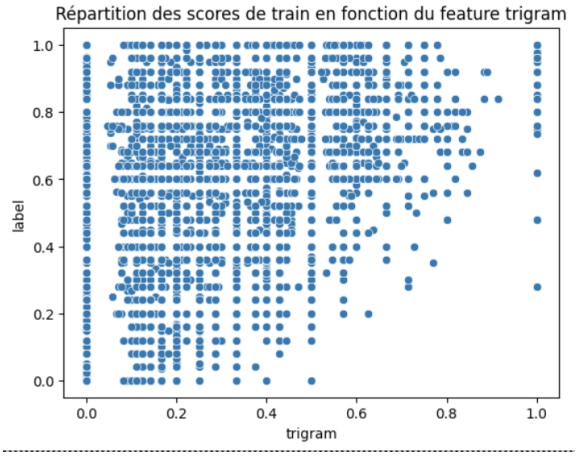

FIGURE 6 – Répartition des scores de *train* en fonction de la feature tri-gram.

	bi-gram	tri-gram	bi-tri-gram	baseline (regression linéaire)
Score de Pearson	0.523	0.467	0.534	0.600
MSE	0.068	0.073	0.066	0.060

TABLE 8 – Comparaison des performances de prédiction pour le model linéaire pour les n-gram feature et baseline feature.

	bi-gram	tri-gram	bi-tri-gram	baseline (regression linéaire)
Score de Pearson	0.541	0.492	0.558	0.600
MSE	0.066	0.071	0.064	0.060

TABLE 9 – Comparaison des performances de prédiction pour le model random forest pour les n-gram feature et baseline feature.

Ainsi, on peut tirer plusieurs conclusions :
- La combinaison de la feature bigram et trigram (bi-trigram) prédit le mieux le score de similarité.
- Bien que les résultats de cette feature seuls ne surpasse pas la baseline, ses performances s'en rapprochent.
- Les meilleures performances pour la feature n-gram sont obtenues en prenant la combinaison de features bi-gram et tri-gram pour entraîner le model Random Forest Regressor.

6.6 Résumé des analyses individuelles

Pour conclure, nous voyons sur la figure 7 que les features les plus performantes sont WordNet ainsi que TFIDF. En effet, WordNet prend en compte les scores hypernymes et synonymes ce qui permet d'augmenter le champ lexical de chaque mot et ainsi d'avoir plus sur la sémantique des phrases.

Linear model evaluation TEST SET -dataset without negations- :

	ngram	wordnet_feature	tf_idf	word_alignment	postag_overlapp
pearson	0.535638	0.615013	0.613307	0.551435	0.146257
mse	0.066359	0.058063	0.058060	0.065410	0.091285

Random Forest Regressor model evaluation TEST SET -dataset without negations-

	ngram	wordnet_feature	tf_idf	word_alignment	postag_overlapp
pearson	0.558327	0.619857	0.604947	0.564006	0.202380
mse	0.064458	0.057426	0.059648	0.063634	0.089458

FIGURE 7 – Tableau récapitulatif de toutes les features

Parallèlement, nous avons réalisé les mêmes tests avec le dataset prétraité pour lequel nous n'avons pas retiré les négations des phrases. Cette étude est présente dans le Google Colab. Nous avons observé que les performances sont identiques sur le dev set. Ceci s'explique car il n'y a très peu de paires de phrases où la première phase présente des négations et l'autre non.

Dans un autre corpus de texte, cette nuance aurait pu être très importante et influer fortement la feature ngram.

7 Analyse multiple feature

Dans cette partie, l'objectif est de tester plusieurs combinaisons de features et d'entrainer puis tester différents modèles afin de prédire le score de similarité entre deux phrases. Ces différentes approches seront comparées aux performances obtenues dans la baseline.

Afin d'avoir une première idée intuitive des features qu'il pourrait être utile de combiner, nous nous sommes intéressées à la matrice de corrélation (figure 8). Grâce à cette matrice de correlation, nous pouvons observer la corrélation entre les features seules et le label.

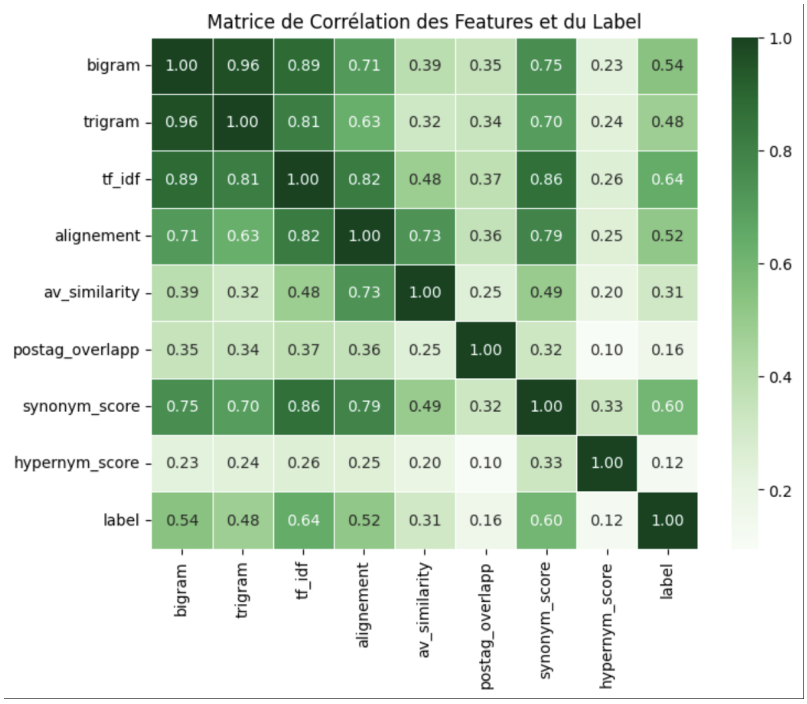

FIGURE 8 – Matrice de corrélation des features et du label.

On remarque notamment que les features bi-gram, tri-gram, tf-idf, synonyme_score (WordNet) sont fortement corrélées avec le label. "À partir de cette analyse et des tests effectués sur chaque features individuellement, nous avons défini différentes combinaisons de features à évaluer."

On s'est intéressées à différents modèles : un modèle de regression linéaire, un modèle de random forest regressor ainsi que des modèles plus complexes xgboost et gradient boost. Ces modèles ont été notamment choisis car ils constituent une référence dans les challenges de SemEval.

Les résultats des évaluations sur le jeu de données de développement sont résumés ci-dessous.

Les combinaisons testées sont les suivantes :
- Combinaison 1 : bi-trigram, tf_idf
- Combinaison 2 : bi-trigram + word alignment (alignment score (threshold=0.8) + average similarity (threshold =0.8))
- Combinaison 3 : Combinaison 2 + tf_idf
- Combinaison 4 : Combinaison 3 + wordnet_feature
- Combinaison 5 : Combinaison 4 + POS Tag overlap (toutes les features combinées)
- Combinaison 6 : bi-trigram + word alignment + WordNet

	Comb_1	Comb_2	Comb_3	Comb_4	Comb_5	Comb_6	Baseline
Pearson	0.721	0.669	0.724	0.733	0.737	0.711	0.667
MSE	0.043	0.050	0.043	0.041	0.041	0.045	0.50

TABLE 10 – Evaluation sur le model linéaire.

	Comb_1	Comb_2	Comb_3	Comb_4	Comb_5	Comb_6	Baseline
Pearson	0.713	0.674	0.712	0.732	0.744	0.711	0.667
MSE	0.046	0.050	0.047	0.045	0.043	0.048	0.50

TABLE 11 – Evaluation sur le modèle Random Forest Regressor.

	Comb_1	Comb_2	Comb_3	Comb_4	Comb_5	Comb_6	Baseline
Pearson	0.706	0.702	0.719	0.746	0.743	0.725	0.667
MSE	0.047	0.047	0.046	0.043	0.044	0.045	0.50

TABLE 12 – Evaluation sur le modèle XGBoost Regressor.

	Comb_1	Comb_2	Comb_3	Comb_4	Comb_5	Comb_6	Baseline
Pearson	0.724	0.710	0.728	0.752	0.753	0.742	0.667
MSE	0.044	0.0449	0.0439	0.0412	0.0409	0.0412	0.50

TABLE 13 – Evaluation sur le modèle Gradient Boost Regressor.

Plusieurs constats peuvent être tirés de cette étude. Tout d'abord on constate que toutes les combinaisons des différentes features que nous avons testé surpassent les performances du baseline model. Les deux combinaisons qui performent le mieux sont les combinaisons 4 et 5. Finalement, on constate que les meilleures performances

sont obtenues avec le modèle Gradient Boost Regressor en utilisant toutes les features retenues combinées. On obtient alors une MSE = 0.0409 et un score Pearson = 0.753 sur le *dev* set.

En testant ce modèle sur le *test* set on obtient les metriques suivantes :

	Meilleure combinaison : Comb_4	Baseline
Score de Pearson	0.649	0.667
MSE	0.054	0.50

TABLE 14 – Résultats obtenus sur le *test* set avec la combinaison 4.

Bien que dans le test set les résultats de notre meilleure combinaison de paramètre et modèle soient moins bonnes que celles obtenues en dev set, elles continuent de surpasser les performances du model baseline.

Par la suite, il serait intéressant de finetuner les hyperparamètres de ce modèle, en testant puis entraînant sur le *dev* set afin de trouver la meilleure combinaison puis en réentrainant sur le *train* set avec les hyperparamètres retenus. On pourra ainsi espérer obtenir des meilleurs résultats.

Partie 2 : Modèles basés sur des représentations distributionnelles et distribuées des mots

8 Représentation distributionnelle

Pour représenter de manière distributionnelle les mots, il a d'abord fallu constituer un vocabulaire. Pour cela, nous avons pris toutes les phrases de la partie *train* du dataset puis nous avons retiré la ponctuation. Nous avons réalisé deux types de preprocessing : l'un retire les stopwords tandis que l'autre les conserve (Table 20). Dans les deux cas, nous avons fait le choix de lemmatizer certains mots susceptibles de prendre différentes formes en fonction de leur utilisation. Ainsi, tous les mots ayant un tag 'NOUN' (nom), 'ADJ' (adjectif), 'ADV' (adverbe) et 'VERB' (verbe) ont été lemmatizés. Le vocabulaire généré possède un vecteur UNK (unknown) pour les mots qui ne sont pas présents dans la matrice.

Phrase originale	Avec stopwords	Sans stopwords
A woman picks up and holds a baby kangaroo.	['a', 'woman', 'pick', 'up', 'and', 'hold', 'a', 'baby', 'kangaroo']	['woman', 'pick', 'hold', 'baby', 'kangaroo']
The man hit the other man with a stick.	['the', 'man', 'hit', 'the', 'other', 'man', 'with', 'a', 'stick']	['man', 'hit', 'man', 'stick']
A panda dog is running on the road.	['a', 'panda', 'dog', 'be', 'run', 'on', 'the', 'road']	['panda', 'dog', 'run', 'road']

TABLE 15 – Exemple de 3 phrases tirées du dataset *train* et ayant été préprocessées soit en conservant les stopwords, soit en les retirant.

La matrice de co-occurence est ensuite créée en comptant le nombre de fois où deux mots apparaissent en même temps dans la même phrase. On calcule également la matrice PMI (Pointwise Mutual Information) qui permet de quantifier l'interdépendance entre deux mots à partir de la matrice de co-occurence.

On calcule les features de différentes façons en fonction des arguments de la fonction *assign_distributional_vectors()*. Elle prend en paramètres 4 booléens :
- *sim_or_dist* : ce paramètre permet de savoir quel type de distance attribuer. S'il est False, une simple concaténation est faite des deux vecteurs de phrases. Autrement, la distance entre les deux vecteurs est calculées soit en prenant en compte la distance cosinus soit la distance euclidienne.
- *postag* : lorsque ce paramètre est False, seule la ponctuation est retirée du texte. A l'inverse, s'il est True, alors il y a un postagging des mots de chaque phrase et une lemmatization est faite sur ces derniers.
- *lowercase* : Lorsqu'il est True, tous les mots sont tranformés en minuscule. Dans cette partie, ce paramètre est toujours True car le vocabulaire est créé en modifiant tous les mots en minuscules.
- *stopword* : Lors de la phase de préprocessing, si ce paramètre est False, tous les stopwords sont conservés. A l'inverse, s'il est True alors les stopwords sont retirés.

	Stopwords conservés	Stopwords retirés
postag = False stopword = False	0.28	0.22
postag = False stopword = True	0.28	0.22
postag = True stopword = False	0.41	0.49
postag = True stopword = True	0.51	0.48
Distance euclidienne	0.32	0.19
Concaténation	0.10	0.11

TABLE 16 – Résulats sur la partie *dev* du dataset en fonction des paramètres. Lorsque la distance n'est pas précisée, cela signifie que la distance cosinus qui est utilisée.

D'après les résultats 16 sur la partie *dev*, lorsque les stopwords sont conservés au moment de la création du vocabulaire, on obtient le meilleur score (0.54) lorsque les phrases du dataset *dev* ont été lemmatizées et que les stopwords sont retirés. A

l'inverse, si au moment de la création du vocabulaire les stopwords sont retirés, alors on obtient le meilleur score (0.52) lorsque les stopwords dans les phrases de la partie *dev* sont conservés et lemmatizé.
Dans tous les cas, on peut voir que la concaténation ne donne pas de bons résultats, cette méthode à donc très rapidement été mise de côté. De manière similaire, si le calcul de la distance est fait avec la méthode euclidienne, les résultats ne sont pas optimaux. Dans la Table 16, un seul exemple de résultats avec la distance euclidienne est montré car tous les résultats étaient semblables.

Score de Pearson	MSE
0.47	0.06

TABLE 17 – Résulats sur la partie *test* du dataset avec les paramètres optimaux : distance = cosinus ; sim_or_dist=True, postag=True, lowercase=True, stopwords=True

Pour tester le modèle sur la partie *test* du dataset, nous avons réunis les parties *train* et *dev* afin d'entrainer le modèle sur 80% des données. On obtient les résultats présentés dans la table 17. On peut voir que ce modèle donne des résultats assez éloignés de ceux de la baseline. Créer notre propre vocabulaire ne semble donc pas être la bonne méthode pour déterminer les similitudes entre deux phrases.

9 Représentation distribuée

Dans cette partie, nous exploitons des représentations de mots pré-entraînées au lieu de créer nos propres embeddings. L'objectif principal est de transformer les phrases en représentations vectorielles en combinant les embeddings des mots individuels après un prétraitement adéquat. Ce processus est similaire à celui utilisé pour les représentations distributionnelles, où les vecteurs des mots sont combinés pour capturer la signification globale des phrases.

Lors de notre projet, nous avions initialement prévu d'utiliser pymagnitude, mais nous avons rencontré des problèmes de compatibilité liés à la version de Python sur Colab, ainsi que des conflits entre certains packages. En conséquence, nous avons décidé d'utiliser Gensim comme alternative. Cependant, ce choix nécessitait de prendre en compte certains aspects spécifiques, notamment la normalisation des vecteurs et le traitement des mots inconnus, qui étaient gérés différemment dans ce framework.

Dans un premier temps nous avons fait comme choix de prendre le modele d'emebdiing de google : *GoogleNews-vectors-negative300*. Une idée que nous avions eu au cours du projet était d'utiliser un autre modèle d'embedding comme FastText qui, bien qu'il soit plus volumineux, à une meilleure gestion des mots inconnus des sous-mots. Cela nous aurait permis également de ne pas avoir à lemmatizer (suite à la gestion des sous mots). Après avoir exploré le fichier pour voir à quoi ressemblait les sorties pour un mot nous avons exploré la Word Mover's Distance (WMD). Il s'agit d'une métrique avancée qui exploite les vecteurs de mots pour mesurer la distance sémantique entre deux phrases. Contrairement à des mesures plus simples comme la similarité cosinus, le WMD calcule la distance de "transport optimal" entre les vecteurs des mots dans deux phrases, cherchant le chemin le plus court pour relier leurs significations. Cela permet de capturer des relations sémantiques plus subtiles entre les phrases. Cette étape marque une avancée importante dans notre projet, car l'objectif est désormais d'obtenir une représentation vectorielle globale de la phrase, plutôt qu'une approche mot par mot. Le WMD et le Word Alignment sont deux méthodes de mesure de similarité entre phrases, mais elles diffèrent dans leur approche. Le WMD calcule une distance globale en optimisant le "coût de transport" pour transformer une phrase en une autre, capturant des relations sémantiques complexes.

En revanche, le Word Alignment se concentre sur des alignements locaux entre mots, utilisant un seuil pour identifier les correspondances significatives. Le WMD, étant plus adapté aux analyses globales, nous semblait particulièrement intéressant à explorer dans le cadre de notre projet.

Dans un premier temps, nous avons filtré les jeux de données prétraités afin de ne conserver que les phrases contenant au moins un mot présent dans le fichier des embeddings. Cela a conduit à une réduction des jeux de données de la manière suivante (exemple pour pour le dataset prétraité sans négation) :
- Train : 3 paires supprimées sur 5749 (0.05% supprimées).
- Dev : 0 paires supprimées sur 1500 (0.00% supprimées).
- Test : 16 paires supprimées sur 1379 (1.16% supprimées).
Nous nous sommes permis de faire cela car le nombre de cas n'était pas important.

La fonction *apply_wmd_to_dataset* calcule la Word Mover's Distance (WMD) entre chaque paire de phrases. Pour gérer les mots qui ne sont pas présents dans le modèle de vecteurs pré-entraînés (mots hors-vocabulaire ou OOV), la fonction commence par calculer le vecteur moyen de tous les vecteurs du modèle. Ce vecteur moyen est ensuite utilisé comme représentation générique pour les mots OOV. Pour

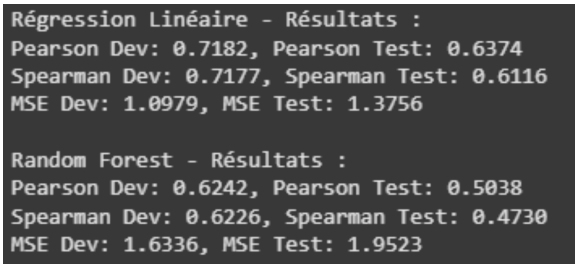

FIGURE 9 – Caption

chaque paire de phrases, la fonction crée un dictionnaire de vecteurs où chaque mot unique des deux phrases est associé à son vecteur normalisé : si le mot est présent dans le modèle, son vecteur réel est utilisé ; sinon, le vecteur moyen est assigné. Les vecteurs sont ensuite normalisés avant de calculer la distance WMD entre les deux phrases. Enfin, pour analyser les résultats obtenus, nous avons entraîné un modèle de régression linéaire ainsi qu'un modèle de random forest, permettant de comparer leurs performances et de mieux comprendre l'impact des distances WMD sur la tâche ciblée. L'idée était de voir la performance de la "feature" seule.

Nous remarquons que les score sont plutot bons nottament pour la régression lineaire, qui donne de meilleure resultats que notre modele de référence. Mais nous devons prendre ne compte que d'enlever des phrases et de remplacer des vecteurs absents par des vecteurs moyens a pu avoir un impact positif mais biaisé sur le modele donnant ainsi de si bon resultat.

Les résultats montrent que la régression linéaire surpasse le random forest dans ce projet de mesure de similarité sémantique des paires de phrases. Avec une corrélation de Pearson de 0.718 sur l'ensemble de validation et un MSE de 1.097, la régression linéaire capture efficacement la relation linéaire entre les distances WMD et les scores de similarité cibles. En revanche, le random forest obtient des performances inférieures, suggérant que les relations présentes dans les données sont principalement linéaires. Cependant, le prétraitement des données, incluant le filtrage des phrases non représentées et la substitution des mots OOV par des vecteurs moyens, pourrait avoir introduit un biais positif en simplifiant la tâche. Ces résultats confirment l'efficacité de la WMD comme indicateur de similarité, mais peut être nécessitent une validation supplémentaire sur des données moins biaisées.

Nous avons ensuite exploré d'autres moyens d'obtenir une représentation vecto-

rielle des phrases, en utilisant des techniques alternatives à la WMD. La fonction *assign_ distributed_ vectors()* a été conçue pour calculer des caractéristiques à partir de ces représentations. Selon les paramètres spécifiés, cette fonction peut calculer soit une mesure de similarité soit en concatenant les vecteurs. Si la mesure de similarité est juste, nous allons soit calculer la WMD distance, soit la distance cosine ou euclidienne (apres une agrégation des vecteurs de chaque phrase de chaque paire) .

Nous avons étudié l'impact de la normalisation, notamment à quel moment elle devait être appliquée (avant ou après l'agrégation). Après plusieurs tests, nous avons conclu que la normalisation des vecteurs était cruciale (nous avions de meilleurs resultats). Car nos resultats de modèles etaient moins bon. Pour chaque sous-ensemble des données (train, dev et test), la fonction traite chaque paire de phrases individuellement. De plus nous avons fait le choix que si un mot n'est pas trouvé dans le modèle, il est remplacé par le vecteur moyen calculé précédemment. Cela a pu introduire un biais positif dans note modèle. Nous aurions pu également affecter un vecteur 0, utiliser des modèle BERT ou recherche de mots similaires en utilisant un algorithme comme la distance de Levenshtein ou un dictionnaire pour trouver un mot proche et utiliser son embedding.

Enfin, nous avons développé une fonction permettant de tester plusieurs combinaisons de paramètres. L'objectif était d'identifier les caractéristiques ayant le plus d'impact sur le modèle, ainsi que les méthodes d'agrégation et de calcul des distances à privilégier. Nous avons également étudié l'influence de l'exclusion de certains types de mots (par exemple, les adjectifs) à l'aide du POS tagging, pour évaluer si cela affectait la performance du modèle.

Voilà un extrait des resultats obtenu sur l'ensemble test :

Les résultats obtenus montrent que les configurations utilisant la distance cosinus sans Word Mover's Distance (WMD) se démarquent par leurs performances, en particulier avec une agrégation par moyenne ou par somme. La configuration 5 (agrégation par moyenne, distance cosinus, sans postagging) atteint un Pearson de 0,6646 et une MSE de 0,0517, ce qui représente les meilleures performances globales. Cette configuration illustre que la moyenne, en tant que méthode d'agrégation, préserve bien les relations sémantiques globales des mots dans une phrase tout en atténuant les variations extrêmes, ce qui rend les vecteurs agrégés particulièrement adaptés au calcul de la distance cosinus.

La configuration 13, qui utilise une agrégation par somme avec la distance cosinus (sans WMD), obtient des performances quasi identiques à celles de la configuration 5. C'est sûrement dû au fait que la distance cosinus compense l'amplification de la somme sur les mots les plus marquants. Cela montre que, dans le cadre des distances directionnelles comme le cosinus, la somme peut être une alternative viable à la moyenne.

En ce qui concerne les configurations utilisant la WMD (comme la configuration 3, avec un Pearson de 0,6358 et une MSE de 0,0551), il est important de noter qu'elles ne s'appuient ni sur une agrégation de vecteurs ni sur des distances simples comme le cosinus ou l'euclidienne. Cette méthode est particulièrement puissante dans des scénarios où les mots importants ne sont pas alignés de manière simple entre les deux phrases. Nous sommes légèrement étonnés de ces résultats, car nous pensions que la WMD serait la plus optimale au vu de sa stratégie de "distance minimale".

Nous supposons ici que les performances légèrement inférieures de la WMD peuvent être attribuées à sa sensibilité aux variations dans les distributions de mots et à son coût computationnel plus élevé. Elle excelle davantage lorsque les phrases sont longues ou contiennent des relations contextuelles complexes, ce qui semble être moins le cas dans ce dataset. Dans les configurations sans WMD, la distance euclidienne reste une alternative robuste, bien qu'elle ne capture pas aussi bien les relations directionnelles que le cosinus.

Un autre point intéressant est l'impact du postagging. Les configurations sans postagging (par exemple, config 5 et config 13) surpassent systématiquement celles avec postagging (config 10 et config 2). Cela suggère que conserver toutes les catégories grammaticales, y compris les adjectifs et les adverbes, enrichit la représentation sémantique des phrases.

Enfin, les configurations avec une agrégation par maximum (configurations 8 et 9) montrent des performances nettement inférieures. Cela est attendu, car le maximum ignore les contributions globales des mots en se concentrant uniquement sur les valeurs extrêmes, ce qui peut être inadéquat pour modéliser des similarités complexes. Les résultats mettent en évidence que la distance cosinus combinée à une agrégation par moyenne ou par somme est la stratégie la plus efficace pour cette tâche.

Partie 3 : Modèle neuronal

Nous allons désormais adopter une approche par réseau de neurones. Notre objectif sera de créer un modèle neuronal capable de représenter sous forme de vecteurs les 2 phrases en entrée. L'idée est que, plus deux phrases sont sémantiquement similaires, plus leurs vecteurs respectifs seront proches dans cet espace vectoriel.

Le modèle, déjà implémenté, est un réseau Siamois bi-directionnel LSTM, conçu pour traiter les phrases de manière séquentielle dans les deux sens et pour comparer les deux phrases en entrée.

Le modèle étant déjà implémenté, notre tâche sera d'entraîner le modèle selon plusieurs configurations de paramètres et de l'évaluer sur le dev set. Et finalement, d'évaluer le meilleur modèle sur le test set.

Pour configurer notre modèle, nous allons jouer avec 3 paramètres : le nombre de couches, le nombre de neurones et la taille des embeddings. Pour comparer nos configurations, nous calculons la loss (MSE) et le coefficient Pearson r. Les résultats sur la partie *dev* du dataset sont présentés dans la Table 18.

Emebedding_dim	Hidden_dim	Num_lstm_layers	Loss	Coefficient de Pearson
70	150	1	0.23	0.47
100	150	1	0.23	0.51
70	200	1	0.22	0.50
70	150	3	0.25	0.41
100	200	3	0.25	0.39

TABLE 18 – Résulats sur la partie *dev* du dataset en fonction des paramètres

Nous observons que nos meilleurs résultats sont obtenus lorsque nous augmentons soit la dimension des embeddings à 100, soit la dimension des couches cachées

à 200. Cela peut s'expliquer par le fait qu'une plus grande dimension des embeddings permet de mieux capturer les relations sémantiques des phrases, offrant ainsi une représentation plus riche des mots. De même, en augmentant la dimension des couches cachées, le modèle devient plus expressif et peut saisir des dépendances plus complexes entre les mots au sein d'une phrase.

En revanche, les performances diminuent lorsque nous augmentons uniquement le nombre de couches LSTM à 3 ou lorsque nous augmentons simultanément les trois paramètres. Cela suggère que le modèle devient trop complexe par rapport à la taille et à la nature de notre jeu de données, ce qui conduit à un sur-apprentissage, où le modèle mémorise trop de détails spécifiques aux données d'entraînement, au détriment de sa capacité à généraliser sur de nouvelles données.

Pour la suite de cette partie nous allons donc partir sur cette configuration : embedding_dim = 100 ; hidden_dim = 150, num_lstm_layer = 1.

Pour évaluer notre modèle, nous analyserons les paires de phrases ayant les meilleurs et les pires scores de similarité cosinus.

Meilleurs scores	
Sentence 1 : a woman is cutting on onion	Sentence 2 : a man is cutting onions
Sentence 1 : the man cut down a tree with an axe	Sentence 2 : a man chops down a tree

TABLE 19 – Exemple de 3 phrases tirées du dataset *train* et ayant été préprocessées soit en conservant les stopwords, soit en les retirant.

Pires scores	
Sentence 1 : a woman peels a potato onion	Sentence 2 : a woman is peeling
Sentence 1 : a girl is riding a horse	Sentence 2 : a girl is riding a h
Sentence 1 : a man is lifting weights is riding a in a garage	Sentence 2 : a man is lifting w
Sentence 1 : a puppy plays with a plastic container	Sentence 2 : the dog is playing with a p

TABLE 20 – Exemple de 3 phrases tirées du dataset *train* et ayant été préprocessées soit en conservant les stopwords, soit en les retirant.

En observant les phrases que le modèle considère comme les plus proches sémantiquement, on remarque qu'il parvient à capturer efficacement la signification principale des actions décrites et à reconnaître des synonymes. Cependant, il tend à

négliger certaines différences, comme les variations en nombre ("onion" vs. "onions") et en genre ("woman" vs. "man").

En revanche, l'analyse des paires de phrases avec les pires scores met en évidence des faiblesses notables. En effet, on remarque que le modèle a du mal à gérer les différences de temps verbal. Cela est particulièrement évident dans la première paire de phrases, où les deux ont la même sémantique, mais diffèrent uniquement par le temps utilisé entre le présent simple ("peels") et le présent progressif ("is peeling"). De plus, le modèle semble éprouver des difficultés à prendre en compte des informations contextuelles supplémentaires (par exemple, "in a garage") et à reconnaître les relations sémantiques comme les hyperonymes (par exemple, "puppy" et "dog"). Néanmoins, il parvient à distinguer des différences sémantiques fortes, comme celles entre "horse" et "bicycle".

Les faiblesses du modèle pourraient être dues à un jeu de données insuffisant ou trop limité en exemples de paraphrases. De plus, chaque mot du vocabulaire est représenté par un embedding statique (Partie 3.2 "Implementing the Model"), ce qui signifie qu'un mot a un seul vecteur, quel que soit le contexte du mot dans la phrase.

Pour améliorer notre modèle actuel, plusieurs pistes de solutions ont été envisagées :
- Augmenter et diversifier notre jeu de données afin de mieux couvrir les variations grammaticales.
- Intégrer une phase de lemmatisation permettrait de réduire les problèmes liés aux variations grammaticales, comme les différences de temps verbal.
- Enfin, après des recherches, on pense qu'utiliser des embeddings contextuels pré entrainé pourrait être envisagé. On pourrait avec un BERT générer des vecteurs dynamiques qui s'adaptent en fonction du contexte de chaque phrase (il faudrait remplacer nn.Embedding par un modèle BERT avec la bibliothèque transformers).

Partie 4 : Base de données biomédicales

Le dataset se compose de 100 paires de phrases possédant un vocabulaire spécifique au milieu médical. Deux types d'analyses ont été faites : une analyse avec l'embedding Word2Vec et une avec l'embedding BioWordVec. Ce dernier a été réalisé par une équipe de chercheurs qui se sont appuyés sur des corpus de texte provenant de PubMed et sur le vocabulaire biomédical recensé dans Medical Subject Headings (MeSH) [2].

Les embeddings traditionnels tels que Word2Vec se basent sur l'intégralité du mot, ignorant ainsi la structure interne des mots. Néanmoins, les mots sont parfois composés d'informations importantes telles que les préfixes et les suffixes. De plus, il est fréquent dans le domaine de la biologie que les mots soient en réalité un assemblage d'autres mots (par exemple 'deltaprotéosome' est composé de 'delta' et de 'protéosome'). Afin de prendre en compte cette structure interne, chaque mot est décomposé en sous-mots grâce à la méthode des n-gram. Puis, tous les vecteurs des sous-mots sont additionnés pour créer le vecteur final représentant le mot d'origine. Cette méthode permet d'une part d'avoir des embeddings proches pour les mots qui se ressemblent mais également de pouvoir intégrer facilement au modèle des mots rares ou nouveaux.

Dans un premier temps, nous avons fait la comparaison entre les deux embeddings afin de voir les différences. Word2Vec possède plus de mots que BioWordVec (3 000 000 contre 2 324 849) mais les mots sont très variés (noms de personnalités publiques, mots communs...) tandis que BioWordVec contient principalement des mots spécialisés ainsi que quelques mots communs (Table 21). On peut malgré tout noter qu'il y a un certain nombre de mots médicaux déjà présents dans l'embedding Word2Vec (principalement des mots courants du domaine médical comme 'diabète' ou 'tachycardie').

Word2Vec	BioWordVec
Anticonvulsant	sphinterotomy
Strategic	antifeeding
trespassed	cytohistologically
Jim_McNerney	hogdkin
reaccommodate	influencial
Sugary	n-methyl-4-nitroaniline
Richard_Solem	immunology
reconcilation	bookmarks
modèle	prequalified
Mindful	contain

TABLE 21 – Comparaison de 10 mots tirés du vocabulaire de Word2Vec et BioWordVec et choisis aléatoirement

Word2Vec & BioWordVec
occurrences
artificially
microbiologists
clarithromycin
correlates
bacteriophage
antihypertension
biotechnologists
cardiomyopathies
proteasomes

TABLE 22 – Liste de 10 mots présents à la fois dans Word2Vec et dans BioWordVec

Si on regarde le nombre de mots de notre dataset présents dans les deux embeddings, on constate qu'il y a un meilleur coverage avec l'embedding Word2Vec (71,2%) qu'avec BioWordVec (67.6%). On pourrait à première vue penser que Word2Vec sera plus performant pour calculer la similarité entre deux phrases car il contient plus de mots vectorisés du dataset. Mais lorsque l'on regarde le score de similarité entre deux mots du domaine médical, on constate que l'on obtient un meilleur score avec BioWordVec qu'avec Word2Vec (Table 23).

Mot 1	Mot 2	Word2Vec	BioWordVec
virus	infectionis	0.66	0.73
protein	molecules	0.68	0.70
disease	disorderis	0.47	0.55

TABLE 23 – Comparaison du score de similarité (cosinus) avec l'embedding Word2Vec et BioWordVec pour des mots du domaine médical

Nous avons ensuite analysé la similitude entre les paires de phrases de notre dataset avec les deux embeddings. Nous avons repris notre travail fait dans la 2ème partie en utilisant l'un ou l'autre des embeddings. Pour le preprocessing, nous avons choisi de lemmatiser et de supprimer les stopwords. Pour les paramètres du modèle, nous avons testé les deux combinaisons qui fonctionnaient le mieux dans la partie 2. Les résultats présentés dans la Table 24 montrent seulement le meilleur score de Pearson obtenu entre les deux.

embedding	sim_or_dist	wmdistance	postag	aggregation	R^2	Pearson
Word2Vec	True	True	True	sum	0.099	0.54
BioWordVec	True	True	True	sum	0.39	0.76

TABLE 24 – Résulats sur la partie *test* du dataset biomédical en fonction de l'embedding utilisé.

Les résultats montrent que les modèles utilisant l'embedding BioWordVec obtiennent des performances bien supérieures (score de Pearson = 0,76) par rapport à un embedding plus généraliste (score de Pearson = 0,54). Le dataset contient des mots très spécifiques qui ne sont pas toujours présents dans l'embedding général. La capacité de BioWordVec à découper les mots en sous-mots permet une analyse plus fine, ce qui conduit à des scores de similarité se rapprochant des gold standards. Cependant, nous sommes conscients que ces résultats sont probablement surestimés. En effet, ils suggéreraient que notre modèle performe mieux avec un dataset à la fois plus technique et beaucoup plus petit que le dataset initial.

Dans le Google Colab, vous trouverez une section intitulée "exploration" à la fin. Nous avons choisi de la conserver afin d'expliquer pourquoi nous n'avons pas adopté exactement la même stratégie entre les parties 2 et 4 (les configurations exploitées ne sont pas identiques) et pourquoi les résultats de la partie 4 sont si élevés.

Dans un premier temps, nous avons obtenu de très bons résultats dans la partie 2, ce qui nous a incités à appliquer une approche similaire dans la partie 4. Ces résultats sont ceux présentés ci-dessus. Toutefois, en approfondissant notre réflexion, nous avons exploré d'autres méthodes pour calculer la distance (notamment la distance euclidienne) et corrigé des erreurs de normalisation. Suite à ces ajustements, les résultats de la partie 2 restaient cohérents. En revanche, appliqués au dataset médical, les résultats étaient moins satisfaisants (score de Pearson d'environ 0,3) lorsque nous agrégions les vecteurs et calculions la distance cosinus ou euclidienne, alors que la distance WMD (Word Mover's Distance) produisait des scores corrects.

Nous avons également testé différentes normalisations à plusieurs étapes et recalculé les distances. Afin d'illustrer notre démarche, nous avons conservé les scores issus de notre première méthode. Par ailleurs, nous remettons en question la sous-performance de notre deuxième approche. Plusieurs facteurs pourraient expliquer ce phénomène :
- Taille du dataset : Un dataset médical plus petit ou contenant des paires de phrases avec des similarités sémantiques difficiles à évaluer (par exemple, relations complexes ou synonymes rares) peut affecter la performance. Mots hors vocabulaire (OOV) : La présence de termes importants non couverts par l'embedding.
- Méthode d'agrégation : L'utilisation du vecteur moyen, de la somme ou du maximum peut être limitée dans sa capacité à capturer les relations complexes entre mots, particulièrement dans des phrases médicales où le contexte joue un rôle crucial. - Enfin, il est possible que des erreurs de code ou de raisonnement soient à l'origine de ces résultats. Nous continuons donc à explorer ces différentes hypothèses et à évaluer de nouvelles approches.

Conclusion

Ce projet nous a permis d'explorer en profondeur différentes approches de mesure de la similarité sémantique textuelle, tout en explorant différents types de datasets. En traversant les étapes de prétraitement, de développement de caractéristiques et d'expérimentation de modèles avancés, nous avons non seulement approfondi notre compréhension des techniques de NLP, mais également identifié des points d'amélioration et des axes de recherche prometteurs.

Nos résultats démontrent que les approches classiques, bien que performantes en termes de capture lexicale de la similarité, atteignent rapidement leurs limites face à des défis plus complexes, tels que la gestion des synonymes contextuels ou des structures grammaticales différentes. L'intégration d'approches basées sur des embeddings distribués et des modèles neuronaux a montré un potentiel significatif, en particulier pour capturer des relations plus subtiles et contextuelles. Nous pourrions approfondir nos analyses sur les features individuelles, développer des modèles basés sur des embeddings contextuels, tels que BERT ou BioBERT. Nous aurions pu également optimiser les paramètres des modèles afin de maximiser leur performance sur des tâches spécifiques ou effectuer de la cross-validation.

En somme, ce projet nous a non seulement permis de consolider nos compétences en traitement automatique du langage naturel, mais il a également ouvert la voie à des recherches futures enrichissantes. Les enseignements tirés ici serviront de base à des investigations plus approfondies.