UCLouvain
Faculté de philosophie, arts et lettres

Séminaire de linguistique computationnelle

Similarité de documents

Bianca Ciobanica

LCLIG2140

Université catholique de Louvain

Juin 2024

Table des matières

1 Tâche

Pour ce travail, nous avons exploré les plongements lexicaux afin de comparer le contenu sémantique de documents issus de la presse. Les modèles utilisés sont Word2Vec, FastText et Doc2Vec. En premier lieu, nous avons établi un sous-corpus d'articles issus de la RTBF [2] que nous avons prétraité et échantillonné pour créer un jeu d'évaluation. Ensuite, nous avons entraîné les trois modèles sur les articles du sous-corpus prétraité et avons essayé d'offrir une version « améliorée » de Word2Vec. Une évaluation manuelle a été réalisée pour chaque modèle. Une fois le modèle final choisi, nous l'avons utilisé dans le programme final `launch_session.py` pour fournir n documents similaires à partir d'un texte donné en entrée avec plusieurs options disponibles.

2 Méthodologie

2.1 Scripts utilisés

- `01_preprocess_corpus.py` crée nouveau sous-corpus, le prétraite (tokenisation et lemmatisation) et crée un jeu d'évaluation ;
- `02_train_models.py` entraîne les modèles et les sauvegarde ;
- `03_evaluate_models.py` enregistre les résultats pour l'évaluation manuelle ;
- `store_corpus_vectors.py` enregistre les vecteurs pour tout le corpus du modèle final ;
- `get_most_sim.py`, `parser_builder.py` et `preprocess_doc.py` servent pour le programme final `launch_session.py` qui fournit des documents similaires à partir d'une requête.

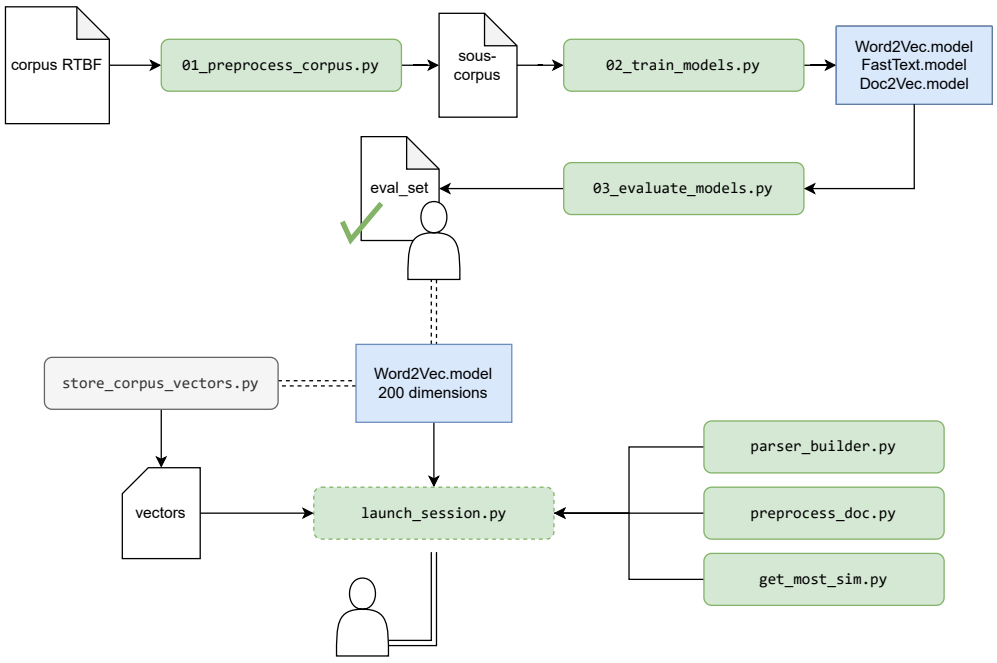

Figure 1: Schéma de la méthodologie

2.2 Prétraitement

Le sous-corpus comprend au total 40 856 articles du feed RTBF_FEED (exclusion de la radio, etc.) publiés entre 2020 et 2021. Le texte des articles est tokenisé, lemmatisé et normalisé. Bien que nous ayons tenté de conserver les noms propres, la détection par *spacy* manquait parfois de précision et les articles sont finalement insensibles à la casse. Afin de mieux capturer les relations sémantiques des mots, certaines classes grammaticales ont été ignorées, toujours à l'aide de *spacy*, telles que les pronoms, auxiliaires, déterminants, ponctuation, espaces et autres symboles (nous n'avons pas voulu nous fier à liste prédéfinie de *stopwords*, car cette liste pourrait ne pas être adapté à notre tâche). Cette stratégie permet de ne pas encombrer le modèle de termes peu significatifs en vue d'obtenir de meilleurs résultats. Il est important de noter que le prétraitement avec *spacy* s'est révélé peu efficace en termes de durée et de précision, mais toutefois suffisant.

2.3 Calcul de la similarité

Pour obtenir les n documents les plus similaires à partir d'une requête, nous procédons comme suit : nous calculons la moyenne des vecteurs des mots présents dans la requête et pour chaque document du sous-corpus. Les vecteurs sont produits par le modèle qui représente la position des mots dans l'espace vectoriel du modèle (embeddings). Lors de l'obtention des vecteurs, nous effectuons une pré-normalison et post-normalisation ; cela permet de maintenir une stabilité numérique lors des calculs. Nous prenons ensuite ces vecteurs pour calculer la similarité cosinus entre la requête et les documents du sous-corpus. Ce score peut être interpreté comme la proximité au niveau du contenu sémantique dans l'espace vectoriel (en gardant tout de même à l'esprit qu'une petite différence dans la valeur de ce score entraîne une grande différence au niveau de leur proximité).

3 Entraînement des modèles

Les modèles ont été entraînés sur 25 epochs et 67 577 mots de vocabulaire ; d'abord une première version avec 100 dimensions et une autre avec 200 pour les trois modèles (Word2Vec [4], FastText [1], Doc2vec [3]). FastText nous servira pour notre version « améliorée » de Word2Vec. L'idéal serait d'identifier les hyperparamètres optimaux à l'aide d'une *grid-search*, mais cela n'a pas été effectué dans ce travail et reste une piste d'amélioration.

3.1 Version améliorée de Word2Vec

Un défaut majeur de Word2Vec est sa capacité limitée à généraliser aux mots qui ne sont pas inclus dans son vocabulaire lors de l'entraînement.

Nous avons donc expérimenté avec une version améliorée de Word2Vec afin de répondre à ce problème. L'idée est de se servir des embeddings de FastText lorsqu'un token n'est pas connu

par le modèle puisque FastText offre une représentation du mot au niveau-infralexical ce qui lui permet de traiter les mots inconnus. Lorsqu'on récupère la moyenne des vecteurs des mots pour un document document et que le document possède un mot inconnu par le modèle, cette nouvelle version se sert de l'embedding de FastText pour ce token. Nous projetons cet embedding dans l'espace de Word2Vec à l'aide de l'algorithme Procrustes orthogonal [5]. Cette méthode est appropriée car les deux modèles partagent la même taille de vocabulaire et le même nombre de dimensions pour l'embedding, un prérequis pour l'algorithme. Pour la version améliorée, nous avons fait la concaténation d'un max pooling ainsi que la moyenne du vecteur pour obtenir le vecteur final qui servira pour les calculs de similarité.

Pour Word2Vec simple et Doc2Vec, on récupère la moyenne des vecteurs comme énoncé en 2.3.

4 Évaluation

Pour évaluer les modèles, nous avons examiné les résultats sur 25 articles échantillonnés au sein du même sous-corpus utilisé pour l'entraînement et 25 articles hors de ce corpus (articles publiés avant 2020 mais toujours du feed RTBF_FEED). Cette approche permet d'évaluer les performances des modèles sur des données jamais vues auparavant. Pour chaque article, nous avons évalué manuellement le top 3 fournit par chaque modèle (Word2Vec, Word2vec « amélioré », Doc2Vec). Dans un premier temps, nous avons évalué les modèles entraînés avec 100 dimensions. Malheureusement, il y a eu une mauvaise manipulation lors de la sauvegarde pour `evaluation_set_v100.csv`, mais au moins il comprend les statistiques finales. Les modèles entraînés avec 200 dimensions ont ensuite été évalués. Les résultats se trouvent dans le dossier `evaluation_sets`, cette fois-ci sans mauvaise manipulation lors de la sauvegarde !

4.1 Résultats

Le modèle le moins performant a été Doc2Vec, peut-être en raison du prétraitement peu efficace. Nous avons en effet utilisé la tokenization simple de *gensim*, car notre tokenization personnalisée avec *spacy* engendrait une forte perte d'information trop importante alors que Doc2Vec nécéssite beaucoup de texte. Les scores de Doc2Vec étaient très bas et les résultats peu convaincants. Par exemple, un article sur la mousse au chocolat avait tendance à ressortir beaucoup alors qu'il n'en était pas question dans le document entrée en requête.

En ce qui concerne notre version « améliorée » de Word2Vec, elle présentait souvent des scores cosinus trop élevés alors que le contenu sémantique n'était pas similaire à la requête. Dans 100 % des cas le document requête se trouvait en première position dans les documents similaires de Word2vec simple et Word2vec amélioré ce qui confirme la fiabilité des modèles. Cependant, les modèles étaient moins performants lorsque le texte n'était pas tiré du même corpus. Ceci n'est pas surprenant étant donné que la taille de notre sous-corpus n'est pas conséquente.

Le modèle le plus performant s'est révélé être Word2vec simple avec 200 dimensions qui est notre choix final; ses résultats étaient convaincants à 90 % comparé à 78 % avec 100 dimensions.

5 Script final

Le script final est divisé en plusieurs petits modules chacun traitant une étape spécifique. Premièrement, le `launch_session.py` lance une session interactive dans l'invite de commande. L'utilisateur peut ensuite choisir les différentes options qui lui sont proposées. Si l'utilisateur souhaite réaliser une requête de similarité, le programme principal se sert de `parser_builder.py` qui traite les arguments donnés en entrée. Nous filtrons le corpus si l'utilisateur souhaite obtenir les articles à partir d'une certaine date. Ensuite, `get_most_sim.py` va tokeniser à l'aide de `preprocess_doc.py` si l'utilisateur fournit son propre texte, sinon nous sélectionnons le document au sein du corpus à l'aide de l'identifiant fournit en entrée (le fichier `all_ids.txt` contient tous les ids du corpus). Une fois le texte prétraité, nous obtenons les vecteurs à l'aide de Word2Vec chargé au début de la session pour obtenir le top n (par défaut 3) des documents similaires à l'aide de la similarité cosinus du vecteur requête et des vecteurs du corpus. Enfin, les résultats sont affichés sur la console et sauvegardés dans le répertoire `resultats_requetes` sous format csv.

6 Pistes d'améliorations

Une première piste d'amélioration serait d'augmenter la taille de notre corpus, car le modèle a été entraîné sur les articles publiés seulement en 2020 et 2021. Certaines thématiques peuvent être donc sous représentées rendant le modèle moins robuste pour la généralisation. Par conséquent, la méthode de prétraitement mériterait également d'être améliorée si nous augmentons la taille du corpus, car pour l'instant elle est assez couteuse en termes de temps. Bien qu'augmenter la taille du corpus permette aux modèles de mieux généraliser sur de nouvelles données, si la qualité ne peut pas être maintenue comme nous l'avons tenté pour un corpus de taille plus petite, un petit jeu de données de qualité pourrait être plus performant que des données bruitées.

L'acquisition des données de qualité constitue la base fondamentale de la tâche et pourtant cette étapte nécessite des ressources importantes. De plus, une fois le modèle sélectionné pour la tâche, il faut également vérifier si celui-ci va de main avec les données que nous utilisons. Pour ce travail, nous aurions pu explorer cet aspect plus en profondeur.

7 Remerciements

Je souhaite vous remercier (et les autres membres du Cental) pour votre dévouement continu à nous fournir des cours de haute qualité. Votre disponibilté et gentillesse est également appréciable. J'apprends toujours quelque chose de nouveau et je ne peux que vous remercier pour cela !

References

[1] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

[2] Escouflaire, L., Bogaert, J., Descampe, A., and Fairon, C. (2023). The RTBF Corpus: a dataset of 750,000 Belgian French news articles published between 2008 and 2021.

[3] Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents.

[4] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

[5] Schönemann, P. (1966). A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10.