Re-embedding via Locally linear embedding

Attali Hugo

6 janvier 2022

Table des matières

Résumé

Dans ce projet, nous proposons d'implémenter une méthode de re-embedding via manifold learning. Nous nous appuierons en partie sur le papier de **Souleiman Hasan and Edward Curry** [1]. Après avoir implémenté notre algorithme nous l'appliquerons à des données de thread Reddit. Nous choisissons le thème de la présidentielle 2022 en France pour observer l'effet de notre re-embedding. A travers ce papier nous utilisons la méthode de LLE proposée par Sam **T. Roweis1 and Lawrence K. Saul** en 2000 (Locally Linear Embedding) [2] comme dans le papier original, pour réaliser notre re-embedding. En réalité d'autres approches de manifold learning seraient envisageables. Tous nos résultats sont reproductibles en utilisant google colab ou les versions des packages spécifiés au début du code. Vous trouverez en détails mon code ici

1 Introduction

Les données textuelles constituent une source d'information qui permet d'extraire de la connaissance (détecter des régularités, recherche des similarités...). L'analyse textuelle est une branche précise du data minning. Le *word embedding* est une méthode indispensable dans ce domaine. Il s'agit d'une méthode d'apprentissage permettant de représenter chaque mot (ou plus globalement une phrase, un document) par un vecteur. Cette méthode est basée sur le contexte des mots, ainsi les mots apparaissant dans des contextes similaires possèdent des vecteurs qui sont proches. Les méthodes qui proposent des représentations vectorielles les plus connues sont Word2Vec par Mikolov et al. [3] en 2013 et Glove par Pennington et al. [4] en 2014. Cette représentation permet de réaliser des calculs mathématiques sur ces représentations, indispensable dans un contexte de machine learning.

1. https ://aclanthology.org/D17-1033.pdf
2. https ://www.robots.ox.ac.uk/ az/lectures/ml/lle.pdf
3. https ://arxiv.org/pdf/1301.3781.pdf
4. https ://nlp.stanford.edu/pubs/glove.pdf

La problématique que nous tenterons de résoudre est :

Est-il possible d'obtenir une meilleure représentation des mots en modifiant les embeddings ? Et si oui comment ?

Évidemment c'est une question très vaste et nous allons tenter d'y répondre partiellement en l'appliquant a une méthode de manifold bien précise. Dans la suite de notre rapport nous expliquerons le papier sur lequel nous nous sommes appuyés pour réaliser ce travail en détaillant certaines méthodes. Nous présentons enfin les résultats obtenus puis nous ferons une analyse critique de notre méthode et des potentielles travaux à venir.

2 Word Re-Embedding via Manifold Dimensionality Retention

Les techniques d'embedding de mots ont donc pour objectif de représenter les mots d'autant plus proches dans un espace métrique qu'ils sont sémantiquement semblables. Ce papier relève que les représentations de mots peuvent sous-estimer la similitude entre les mots proches et la surestimer entre les mots distants dans cet espace. Dans cet article, les auteurs présentent une méthode de re-embedding de mots via du manifold learning. En conservant la dimensionalité ils montrent que leur méthode peut améliorer les représentations de mots en réalisant des expériences dans les tâches de similarité des mots de 0,5 à 5,0 points.

2.1 Locally Linear embedding

Habituellement la LLE est utilisée pour travailler sur des données de haute dimension. Ce sont des données qui sont difficilement visualisables et interprétables. L'objectif est de produire des variables latentes qui condensent l'information permettant de réduire la dimension.

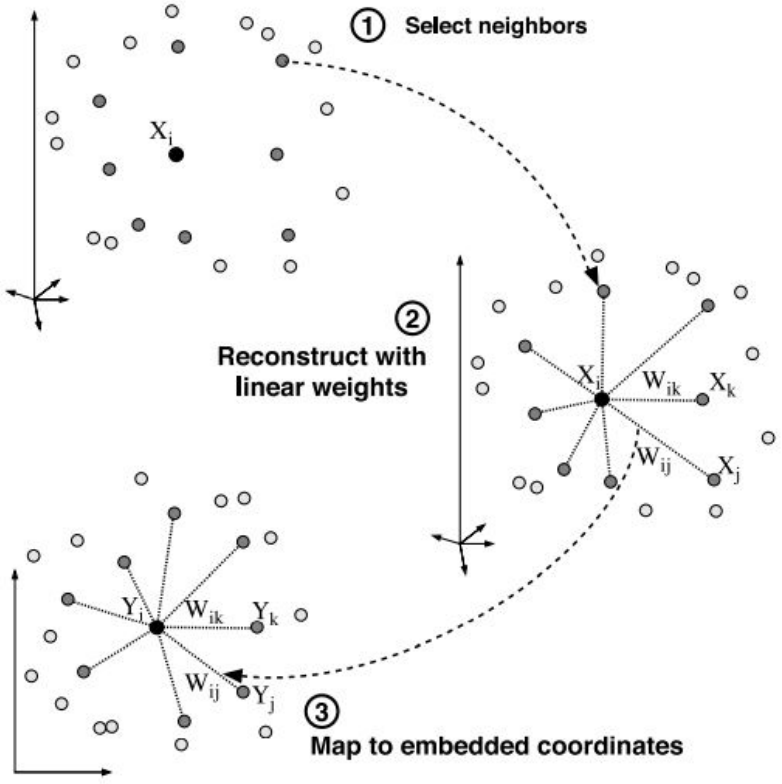

FIGURE 1 – Shéma de la LLE.

2.1.1 Méthode

LLE trouve d'abord les k-voisins les plus proches de chaque point. Ensuite, il approxime chaque vecteur de données comme une combinaison linéaire pondérée de ses k-plus proches voisins. Enfin on résout le problème d'optimisation préservant les distances locales en les projetant dans un espace à plus faible dimension.

Dans notre contexte nous travaillons avec des embeddings donc nous devons minimiser la distance entre deux embeddings de mots proches dans notre espace de grande dimensions avec notre nouvel espace. Nous sommes face à un problème d'optimisation sous contrainte ou les lignes de la matrice de poids, donc de chaque Xi sont égales à un. Pour résoudre ce type de taches on utilise la méthode du Lagrangien.Un bon tutoriel est ici et ici.

Voici le pseudo code de la *LLE* :

Pour i allant de 1 à N : ;
Calculer la matrice de distance entre Xi et tous les autres points;
Trouver les K plus proches voisins ;
Fin Pour;
Pour i allant de 1 à N ;
Créer une matrice Z composée de tous les voisins de Xi ;
Soustraire Xi de chaque colonne de Z;
Calculer la matrice de Gram correspondant au produit scalaire des embedding des voisins de Xi;
Définir $Wij=0$ si j n'est pas un voisin de i;
Fin Pour;
Créez une matrice $M = (I\text{-}W)'*(I\text{-}W)$;
Calculer les $d+1$ vecteurs propres inférieurs de M (correspondant aux $d+1$ plus petites valeurs propres) ;

Dans le papier original le choix des dimensions ne changent pas. Ils réalisent uniquement une transformation de l'embedding de départ. C'est ce que nous allons reproduire.

3 Embedding

Afin de réaliser notre re-embedding, nous devons dans un premier temps, obtenir l'embedding de chaque mot de notre corpus. Pour cela un travail préalable sur nos données sont indispensables. Dans un premier temps il faut nettoyer puis "tokeniser" notre corpus. Dans la phase de nettoyage, nous pouvons effectuer de nombreux traitements selon le contexte de la problématique (retrait de certains signes de ponctuation, harmonisation de la langue, de la casse..). En plus de ces traitements nous devons enlever les mots qui portent peu d'informations, comme par exemple les pronoms, pronoms personnels etc... Les mots comportant peu d'informations dans un corpus est plus communément appelés "stopword". Des librairies de traitement du langages naturel regroupent les stopwords dans une liste. Ils sont ainsi facilement traitables. Nous utilisons dans notre code la librairie NLTK. [5]

Un autre aspect de nettoyage utilisé est la phase de lemmatisation dont l'objectif est de réduire le vocabulaire en regroupant les synonymes, en regroupant les verbes conjugués sous une même racine.

La phase de nettoyage est importante mais elle n'est en réalité jamais parfaite.
Dans ce projet pour créer l'embedding de notre corpus nous utilisons la librairie Gensim [6]. Deux modèles sont proposés, le modèle de sacs de mots continus (CBOW : continuous bag of words) et le modèle skip-gram. Le CBOW vise à prédire un mot étant donné son contexte, c'est-à-dire étant donné les mots qui en sont proches dans le texte. Le skip-gram a une architecture symétrique visant à prédire les mots du contexte étant donné un mot en entrée.

En pratique, le modèle CBOW est plus rapide à apprendre, mais le modèle skip-gram donne généralement de meilleurs résultats. [7]

5. https ://www.nltk.org/
6. https ://radimrehurek.com/gensim/models/word2vec.html
7. https ://arxiv.org/pdf/1301.3781.pdf

Nous utilisons le modele skip-gram. Pour plus de détails un bon tutoriel est ici. Nous créons des embeddings à 300 dimensions et nous comptons un mot dans notre modèle uniquement si le mot appairait au moins deux fois dans notre corpus. Nous obtenons ainsi l'embedding d'environ 1200 mots. Nous pouvons dés lors calculer des similarités entre mots grâce à la similarité cosinus. On l'obtient en en prenant le produit scalaire de nos deux embeddings divisé par le produit de leurs normes. La valeur est comprise dans l'intervalle [-1,1]. La valeur de -1 indique des vecteurs opposés,et la valeur de 1 des vecteurs colinéaires de coefficient positif. Les valeurs intermédiaires permettent d'évaluer le degré de similarité.

Dans notre corpus, la similarité entre "fêtes" et "libertés" de 0.2. Dans un contexte de fêtes de fin d'année et en période covid, obtenir une similarité supérieur à 0 n'est pas surprenant. Pour citer un autre exemple si on calcule la similarité entre fêtes et politique on obtient un résultat de -0.05. Résultat plutôt logique.

Une autre manière d'obtenir les représentations d'un mot est de récupérer des embeddings pré-entraîné. Ces représentations sont déjà apprises sur un corpus conséquent. Néanmoins pour de nombreuses taches il faut "fine-tunner" les représentations pour la tache que l'on veut effectuer. On peut récupérer et obtenir plus d'informations sur les représentations pré-entraîné Glove ici [8].

4 Données et organisation du code

Nous testerons notre algorithme sur des données liées a la présidentielle 2022 en France. Nous scrappons nos données sur *REDDIT* grâce à son API. [9] Nous prenons les threads sur les sujets les plus "hot" avec comme mots clefs : *politique/2022/élection/presendentielle/Melenchon/Macron/Lepen/Bertrand/Zemmour/Jadot*. Ces choix peuvent bien évidemment être encheris de d'autres mots clefs. Notre corpus possède environ 350 documents, et après avoir supprimé les doublons et supprimé les threads possédant peu de mots, nous obtenons prés de 300 documents. Évidemment notre corpus change selon le temporalité de chargement des données. Afin de mieux visualiser nos documents nous créons une classe Thread pour que notre code soit bien organisé, et plus lisible. Cela nous permet de pouvoir rapidement voir un Thread ou un auteur.

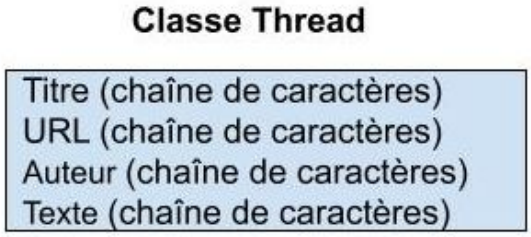

FIGURE 2 – Classe Thread.

En utilisant cette classe nous pouvons facilement avoir accès a des informations clefs sur notre corpus.

8. https ://nlp.stanford.edu/projects/glove/
9. https ://towardsdatascience.com/scraping-reddit-data-1c0af3040768

Nous proposons aussi une classe Glove pour les embeddings pré-entraîné.

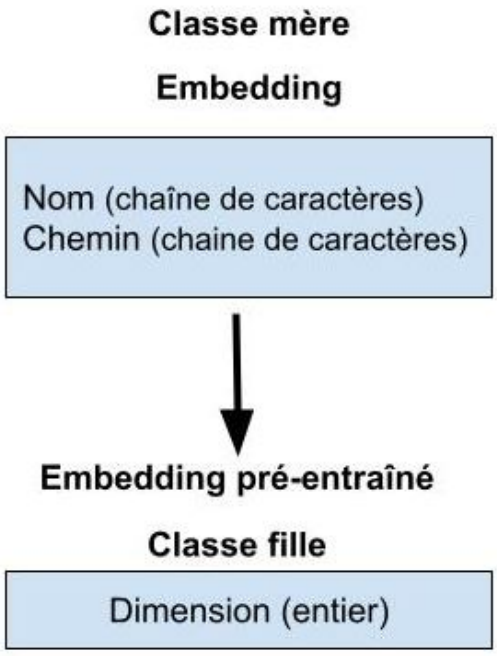

FIGURE 3 – Classe Glove.

Cette classe nous permet de récupérer les embeddings pré-entrainés de Glove en spécifiant leurs dimensions. Les dimensions possibles sont 50, 100, 200 ou 300. Évidemment prendre des vecteurs de dimensions élevés améliorent la précision mais à un coût complexité et mémoire évident. Les mots des représentations pré-entraînés sont en anglais. Il faut donc l'utiliser dans un corpus en anglais. On pourrait éventuellement utiliser les embeddings pré-entrainé "fastext" qui sont un autre manière d'apprendre des représentations de texte dans plusieurs langues. [10]

Dans notre projet nous décidons d'instancer nous même un modèle Word2Vec expliquer en amont afin de manipuler des données textuelles.

Plus globalement notre code s'articule autour de différentes fonctions, nous présentons ici les plus importantes :

Def Glove(chemin vers le repertoire des representations) :
Récupération et traitement des embeddings pré-entraîné
Def clear-thread(Entièreté de notre corpus) :
Nettoyage et tokenisation de nos données
Def Word2Vec(Token de notre corpus) :
Instanciation de notre modèle skip-gram renvoyant les représentations des mots de notre corpus.
Def LLE(Matrice embedding, Nombres de composantes, Nombres de voisins) :
Transformation de nos représentations via la LLE
Def Re-embedding(Mot1, Mot2) :
Compare la similarité entre deux mots écrits par l'utilisateur (mais présent dans le vocabulaire!!) avant et après la LLE.

5 Expériences

On rappelle que le but du re-embedding est d'augmenter la similarité entre les mots proches et la réduire pour les mots éloignés sémantiquement.

10. https ://fasttext.cc/docs/en/supervised-tutorial.html

Pour tester les résultats de notre algorithme nous allons calculer la similarité de deux mots avant puis après la LLE. Dans nos expériences nous prenons 50 dimensions et nous travaillons avec les 6 plus proches voisins.

Dans notre code l'utilisateur est libre de comparer deux mots du moment qu'ils sont présent dans le vocabulaire. On va l'exécuter sur notre exemple précédent à savoir entre les mots fêtes et libertés. On s'attend à une similarité supérieur à notre 0.2 trouvé précédemment. Après exécution de algorithme on trouve une similarité de 0.7 De la même manière on compare les mots "politique" et "fêtes". Peu similaire, on trouve une similarité de 0.05 avant la LLE et de -0.13 avec.

```
Re_embedding("fêtes","libertés")
```

```
utiliser les embeddings pre-entrainés glove ?n
La similarité avant la LLE est de: 0.1896488517522812
La similarité aprés la LLE est de 0.6877135270426364
```

FIGURE 4 – Comparaison de similarity entre les mots fêtes et libertés.

```
Re_embedding("fêtes","politique")
```

```
utiliser les embeddings pre-entrainés glove ?n
La similarité avant la LLE est de: -0.04669519141316414
La similarité aprés la LLE est de -0.12503719271550046
```

FIGURE 5 – Comparaison de similarity entre les mots fêtes et politique.

On voit bien que notre algorithme accorde une similarité encore plus importante aux mots proches sémantiquement et accorde à contrario une similarity moindre aux mots éloignés. Pour faire une analyse plus détailléé, il serait intéressant de regarder la répartition des distances entre les embeddings avant et après la LLE. En effet, l'objectif est d'obtenir une variance plus élevée après la LLE pour mieux différencier les mots proches et les mots éloignés.

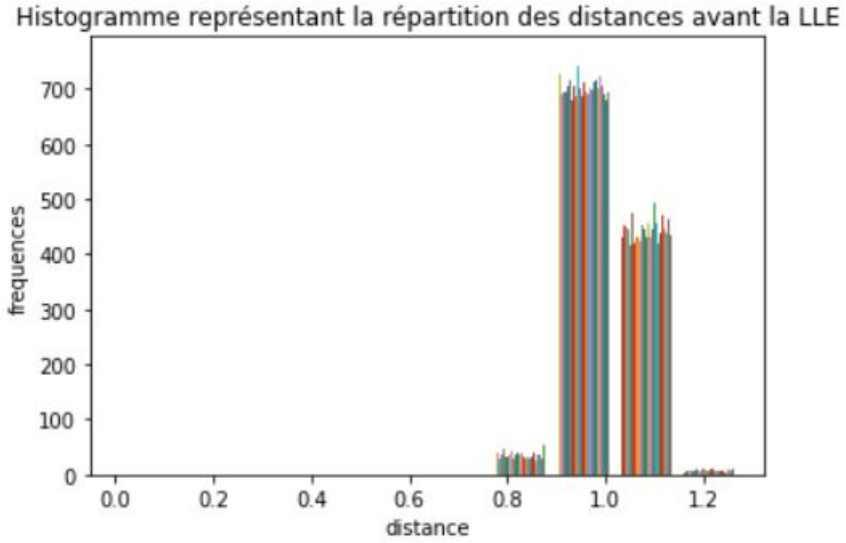

FIGURE 6 – Répartition des distances cosinus avant la LLE.

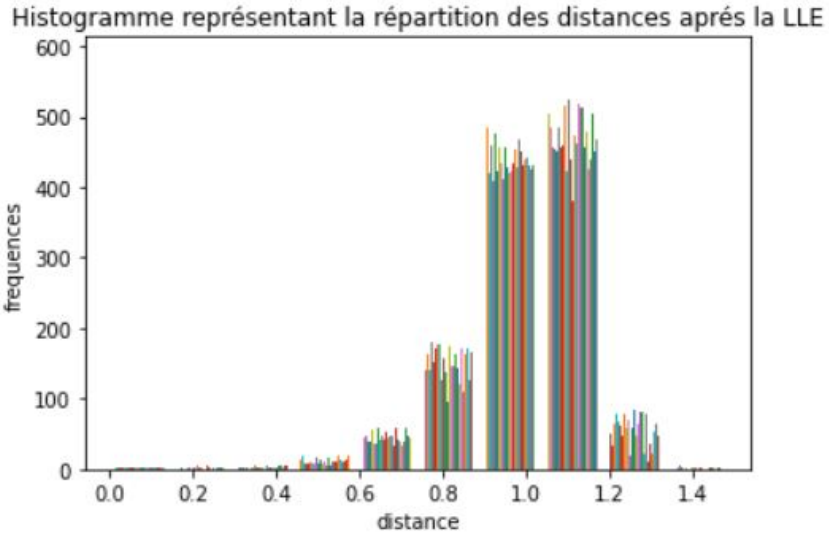

FIGURE 7 – Répartition des distances cosinus après la LLE.

On observe effectivement une meilleur répartition des distances.

6 Analyse critique

Pour faire une analyse critique de notre travail, il faut évidemment prendre du recul sur les résultats. En effet les résultats sont très dépendants du corpus de départ. Ainsi les résultats qui sont montrés dans ce rapport ne seront à priori pas exactement les mêmes. De plus les résultats viennent de commentaire Reddit, qui ne reflètent assurément pas la verité. Par ailleurs nous possédons un corpus plutôt léger.

De plus, la LLE possède une complexité trés coûteuse notamment avec son calcul de la matrice des distances pour trouver les k-plus proches voisins de chaque point.

En nous replongeant dans un contexte NLP, réaliser un re-embedding via du manifold learning peut sembler moins intéressant avec les évolutions rapides du domaine. En effet pour la plus part des taches que l'on peut réaliser en NLP (Predictions de liens, classification...) on est amené à "fine-tunner", spécifier les embeddings en fonction de notre tache. Les derniers modèles de langages comme BERT [11] en 2018 garantissent des représentations très intéressantes. Ainsi faire du re-embedding avant d'effectuer du "fine-tunning" parait lourd et peu intéressant.

Pour notre projet néanmoins il pourrait être intéressant de regarder si nous obtenons une plus grande robustesse si nous avons un corpus plus important ou encore tester notre algorithme sur d'autres données. De plus on pourrait regarder les sorties d'un algorithme de clustering avant et aprés la LLE.

7 Conclusion du projet

Pour faire une courte conclusion, ce projet m'a permis d'implémenter une méthode de re-embedding en introduisant des notions de programmation objet. De plus j'ai dû mener ce projet de la manière la plus complète possible en abordant de nombreuses notions abordées : API, dictionnaires, héritage, manipulation de texte...
J'ai aussi pu en savoir davantage sur la local linear embedding, comment elle est construite, quelle est la théorie sous-jacente de cette méthode, et son application a des données textuelles.

8 Références

https ://aclanthology.org/D17-1033.pdf
https ://www.robots.ox.ac.uk/ az/lectures/ml/lle.pdf
https ://arxiv.org/pdf/1301.3781.pdf
https ://nlp.stanford.edu/pubs/glove.pdf
https ://www.nltk.org/.
https ://radimrehurek.com/gensim/models/word2vec.html
https ://arxiv.org/pdf/1301.3781.pdf8.
https ://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b9.
https ://nlp.stanford.edu/projects/glove/10.
https ://towardsdatascience.com/scraping-reddit-data-1c0af3040768
https ://arxiv.org/pdf/1810.04805.pdf
https ://towardsdatascience.com/lle-locally-linear-embedding-a-nifty-way-to-reduce-dimensionality-in-python-ab5c38336107
https ://datascientest.com/le-word-embedding
https ://www.robots.ox.ac.uk/ az/lectures/ml/lle.pdf
https ://github.com/drewwilimitis/Manifold-Learning
https ://medium.com/analytics-vidhya/locally-linear-embedding-lle-data-mining-b956616d24e9
https ://en.wikipedia.org/wiki/Nonlinear$_d imensionality_r eduction$

11. https ://arxiv.org/pdf/1810.04805.pdf