Nom et Prénom : AMAR Papa
Matricule ULB / Filière : 000540605 / MA-STICS
Identifiant Github : Amar-ulb

STIC-B545 – Rapport TP3

Introduction

Ce document se veut un rapport du TP3 qui porte sur le clustering et l'entrainement d'un modèle Word2Vec sur un corpus.
Nous effectuerons d'abord un clustering des documents CAMille pour une décennie choisie et interpréterons les résultats. Ensuite il s'agira d'entraîner un modèle word2vec sur l'ensemble du corpus et d'explorer les relations entre vecteurs.

1. Clustering et interprétations des résultats obtenus

1.1. Clustering :

Decade = 1960 et Nombre de clusters désiré : K= 3
Une des principaux paramètres à régler dans le clustering K-means est le « n_clusters ».
J'aurais aimé utiliser la méthode Elbow pour déterminer le nombre optimal de clusters. Ne pouvant pas y parvenir (erreur sur mon code) , j'ai essayé arbitrairement plusieurs nombres comprise entre 2 et 10, avant de choisir le nombre 3. Nous verrons plus loin que le résultat fait sens.
Après avoir appliqué le clustering à l'aide de la fonction *`fit_predict`* j'obtiens 1000 documents qui correspondent aux articles de la décade choisie. (Voir *output_s2_clustering.ipynb* dans TP3).

```
[67]  pprint(dict(clustering))                                                    Python

...  Output exceeds the size limit. Open the full output data in a text editor
     {0: ['KB_JB838_1960-01-21_01-00014.txt',
          'KB_JB838_1960-01-22_01-00015.txt',
          'KB_JB838_1960-02-24_01-00020.txt',
          'KB_JB838_1960-03-11_01-00022.txt',
```

Figure 1 - clustering

1.2. Visualisation des clusters

Pour visualiser les documents dans un espace 2D , les vecteurs sont réduits en 2 dimensions à l'aide de l'algorithme PCA.

```
pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(tfidf_vectors.toarray())
✓ 1.4s                                                                    Python
```

Figure 2-Algorithme PCA

Des points (marker = "x") encore appelés centroïdes, sont définis pour former chaque cluster.

```
# Ajouter les centroïdes
centroids = pca.transform(km_model.cluster_centers_)
plt.scatter(centroids[:, 0], centroids[:, 1],  marker = "x", s=100, linewidths = 2, color='black')
```

Figure 3- Ajout des centroïdes

1.3. Interprétation des résultats obtenus

En utilisant K-Means avec k=3 nous avons obtenu le résultat de clustering représenté sur l'image ci-dessous. L'image de sortie montre clairement un nuage de points de notre ensemble de données

avec les 3 différents clusters identifiés par des couleurs. Chaque couleur représente en effet un cluster d'où la palette de 3 couleurs en référence au nombre de clusters prédéterminée.

Ce regroupement s'explique par le fait que le clustering K-means est une méthode de clustering itérative qui divise les données en entrée en k clusters dans lesquels chaque observation appartient au cluster avec la moyenne la plus proche (centroïde). Chaque centroïde (centre de gravité du cluster) est associé à des points de données. On constate 3 centroïdes éloignés les uns des autres. Le regroupement ainsi crée semble raisonnable selon moi, bien que la variance égale inégale dans chaque dimension rende la méthode moins adaptée à cet ensemble de données.

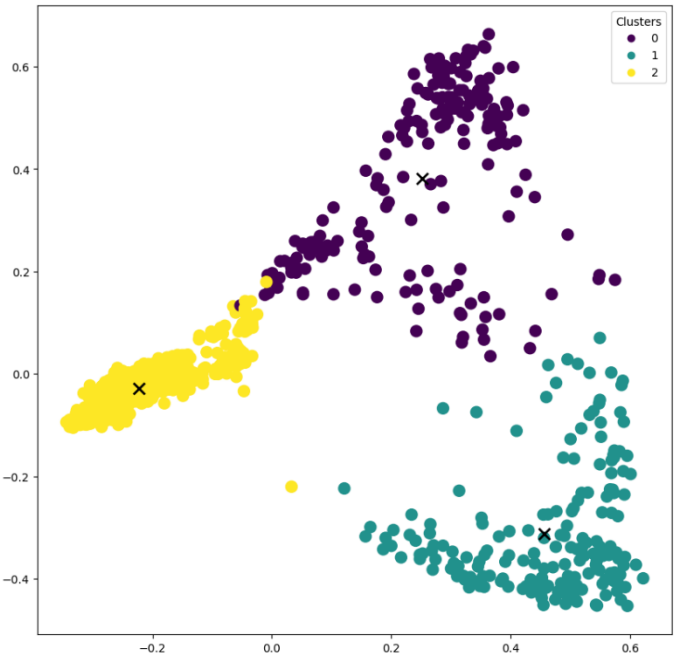

Figure 4 - Nuage de points

2. Entrainement d'un modèle word2vec et exploration des relations entre vecteurs

2.1. Entraînez un modèle word2vec (word embeddings)

Au vu de la quantité des données (pas très importante), nous avons changé quelques paramètres ci-après pour entrainer notre modèle.

Window = 2 : Pour éventuellement avoir des termes plus liés.
min_count = 10: Le modèle ignorera les mots qui ne satisfont pas le *min_count*. Les mots très peu fréquents sont généralement sans importance, il est donc préférable de s'en débarrasser.
workers=1 : Au vu du CPU cores de mon ordinateur, j'ai jugé sage de ne pas faire de parallélisation.
epochs=1 : J'ai constaté que l'augmentation du nombre de epochs augmente considérablement le temps de l'entrainement. Cela pourrait être envisager si nous avions de données plus importantes.

Finalement le modèle *word2vec* sauvegardé dans un fichier *"../data/newspapers.model"* est le suivant

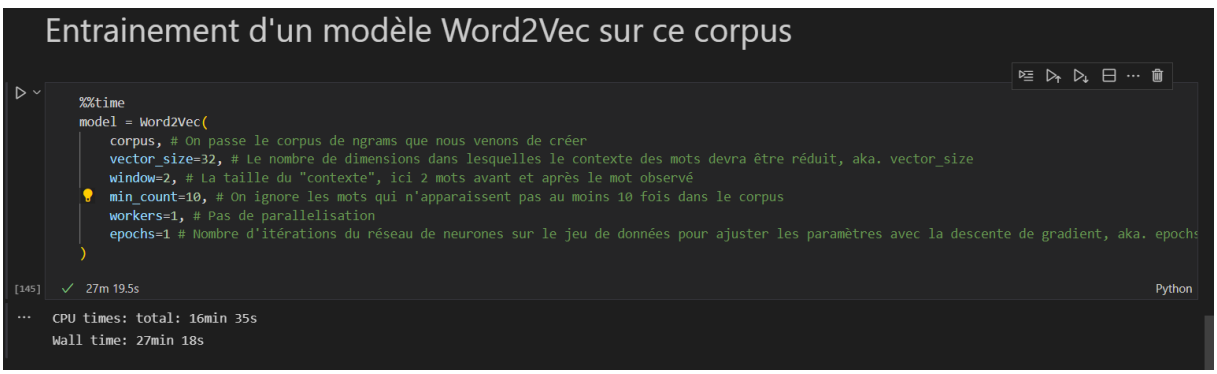

Figure 5-Modèle word2vec

2.2. Explorer les relations entre vecteurs.

2.2.1. Calcul de la similarité entre deux termes avec la fonction *similarity* (3 exemples)

Les trois exemples ci-dessus calculent la similarité cosinusoïdale entre les deux mots spécifiés en utilisant les vecteurs de mots de chacun. D'après les scores ci-dessus, il est logique que "*directeur*" soit similaire à "*doyen*" mais différent de "*élève*". À titre illustratif nous avons opter pour le 3éme exemple un terme identique pour montrer que lorsqu'on fait une similarité entre deux mots identiques, le score sera de 1,0 car la plage de similarité cosinus peut aller de [-1 à 1] et parfois délimitée entre [0,1] selon la façon dont elle est calculée et si les vecteurs ont à la fois des valeurs positives et négatives.

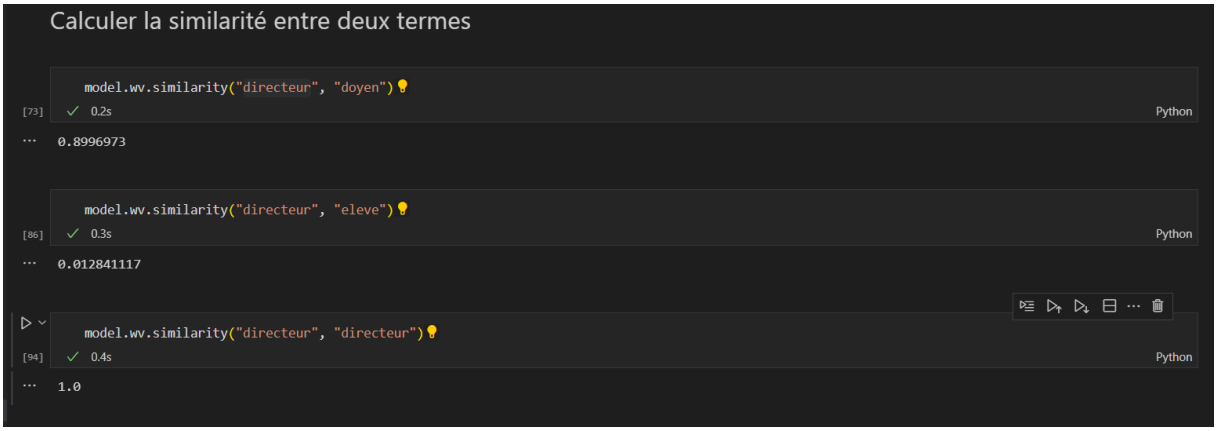

Figure 6-Calcul de la similarité avec la fonction similarity

2.2.2. Recherche de mots les plus proches avec la fonction *most_similar* (3 exemples)

Dans l'ensemble, les résultats ont du sens. Tous les mots associés ont tendance à être utilisés dans le même contexte pour le mot de requête donné.

Figure 7-Recherche de mots les plus proches avec la fonction most_similar

2.2.3. Recherches complexes à travers l'espace vectoriel

- ✓ Exemple 1 : Analogie simple. L'homme est à la femme, ce que le roi est à…

L'exemple consiste à trouver des mots similaires à « femme » et « roi », et non similaires à « homme ». Le premier mot renvoyé par la requête est "queen" étant donné que le mot « roi » a déjà « homme » comme composant de sa représentation vectorielle.

- ✓ Exemple 2 : Analogies géographiques. Trouver dans quel pays se trouve une ville
- ✓ Exemple 3 : Analogies géographiques. La requête capture les relations entre un pays déterminé et les villes associées.

Conclusion

À l'issu de ce TP, j'appréhende mieux trois enseignements vus en cours de traitement automatique de corpus :

- ✓ Prétraiter les données à utiliser avec un modèle Word2Vec
- ✓ Former un modèle Word2Vec
- ✓ Utilisez des métriques qualitatives, telles que *similarity* et *most_similar* pour évaluer la qualité des clusters formés

Par ailleurs je retiens aussi que le clustering avec K-Means est un type d'apprentissage automatique non-supervisé approprié pour regrouper selon un lien de similarité, une grande quantité de données non étiquetées en plusieurs sous-ensembles. Toutefois, je constate d'une part que c'est une méthode sensible aux valeurs aberrantes dans le paramétrage du clustering et l'entrainement du model et d'autre part le choix arbitraire de la valeur optimale de k n'est pas toujours facile.
Enfin j'ajouterai en dernier lieu une petite remarque personnelle et non moins importante : il faut une machine relativement puissante pour la création d'un corpus et l'entrainement un modèle Word2Vec.