Carlos Andres CORTES MIRANDA
DATA732
Janvier 2024
Explications du code

Sémantique Clustering of articles

Table des matières

1. Introduction

Dans ce document je vais expliquer les résultats et conclusions trouves ainsi que l'explication plus détaille des choix de paramétrisation et approches utilises lors de cette étude.

Il faut faire la remarque que l'utilisation des jeux des données de grande taille, en plus des méthodes qui sont classifies comme complexes ont rendu les résultats un peu limites. Tous les notebooks ont été développes avec un jeu des données des 965 articles et dans les cas où il était possible ils ont été lancés et enregistrer avec un jeu des données de 36 592 articles. Le problème est que j'ai utilisé la même paramétrisation pour les deux cas. Il faudrait plusieurs études pour regarder l'impact de la paramétrisations dans les résultats trouves.

Cependant les résultats trouves semblent être adéquates et suffisamment cohérent pour extrapoler des conclusions.

2. Utilisation du pipeline

Toute le code est charge sur un fichier GitHub public dans el lien : cortesmc/DATA732_TopicClassification: Application of différents méthodes to make a classification of topics of a dataset of articles (github.com).

En essaient de maintenir une séparation de modelés et résultats j'ai réalisé plusieurs fichiers .ipynb, avec la distribution suivant :

- 0.TD_plotly : Introduction à Plotly et à l'utilisation de Dash en python (fichier réalise pendant le TP de présentation).
- 1.Dataframe_creation: Le jeu de données initial "preprocess-topaz-data732" a était fourni en format Json. Ce NoteBook me permet de générer des dataframes pandas à utiliser dans les parties suivantes de l'étude.
- 2.1.kmeans_clustering: Première approche du clustering utilisant une méthode kmeans avec une méthode TF-IDF de vectorisation.
- 2.2.LDA_Cluster: Deuxième approche pour réaliser un regroupement sémantique des articles en utilisant Latent Dirichlet Allocation (LDA) avec CountVectorizer. Ce code utilise comme guide cette publication kaggle : Topic Modeling: LSA and LDA | Kaggle
- 3.1.LDA-CountVect_content_cluster: LDA + CountVectorizer avec un ensemble différent de paramètres pour comparer l'utilisation de LDA avec différentes entrées ; CountVectorization dans ce cas.
- 3.2 LDA-TFIDF_content_cluster: LDA + TfidfVectorizer avec un ensemble différent de paramètres pour comparer l'utilisation de LDA avec différentes entrées ; TfidfVectorizer dans ce cas.
- 4.bert_ZeroShotLabel: Approche consistant à utiliser un modèle de classification « Zero-shot » pour attribuer des étiquettes aux éléments. Cette approche n'a pas été développée en raison de l'importance du processus nécessaire pour effectuer les calculs.

3. Notebooks

a. 2.1.kmeans_clustering

Dans ce notebook j'ai utilisé TfidfVectorizer avec les 36000 articles. La seul contraint impose sur TdIDFvectorizer c'est : min_df = 0.1, c'est qui me permet enlever tous les mots dont la fréquence dans les

documents est strictement inférieure au seuil fixé (0.1 dans ce cas). Ce qui me donne une matrice de 36600x289, c-a-d 36600 articles (après enlever toutes les données avec none values) ou chaque article possède 289 « features ». Comment TF-IDF fonctionne en mesurant la fréquence d'apparition d'un mot dans un document et compense sa fréquence dans l'ensemble des documents, donc on finis avec une représentation basée sur la fréquence qui capture l'importance des mots dans un document par rapport à un corpus.

Donc j'ai utilisé cette matrice pour alimenter un « MiniBatchKMeans » avec 7 clusters. J'ai décidé d'utiliser parce que le résultat est similaire à celui de K-means et qu'il est plus rapide à traiter, donc en utilisant un ordinateur qui n'est pas très puissante il me semble une option approprie. Cet comparaison est traite même par scikit learn dans Comparison of the K-Means and MiniBatchKMeans clustering algorithms — scikit-learn 1.3.2 documentation.

b. 2.2.LDA_Cluster

Cet notebook utilise comme guide le notebook kaggle suivante : Topic Modeling: LSA and LDA | Kaggle. En appliquant les mêmes fonctionnes et organisation mais pour notre information.

Le notebook a été lance est sauvegarde avec les data frame de 36000 articles. Il utilise CountVectorizer deux fois, la première avec la contraint de max_features = 1 000 et la deuxième avec 40 000. Dans les deux cas j'ai ajoute une liste de stop_words. Cette liste permet ajouter la contraint d'ignorer les mots qui sont dans la liste.

La liste des stop_words est forme par un liste que j'ai trouve en internet, plus les listes des stop_words fournier par la librairie python « nltk.corpus » en français et anglais. En plus j'ai ajoute les mots 'france', '2019', '2020' et '2021' ; parce que l'apparition fréquent de ces mots était redondant dans l'étude et n'apportent rien.

La function LDA utilise est la suivante :

```
lda_model = LatentDirichletAllocation(n_components=7, learning_method='online', random_state=0, verbose=0)
```

learning_method: le choix de 'online', est plus utilise pour des entrainement ou il a nouvelle data qui arrive avec le temps. Dans ce cas le mode 'batch' est plus approprié à notre cas, mais le choix du mode 'online' c'est parce que c'est plus rapide même si elle est un peu moins précise.

random_state: Permet de reproduire les résultats même en lançant le code plusieurs fois.

Verbose : Il contrôle le niveau de détails dans l'output de la fonction. Comme l'objectif n'est pas de comprendre le modelé, sinon de démontrer les résultats, je n'ai pas besoin des informations supplémentaires.

c. 3.1.LDA-CountVect_content_cluster

L'objectif des deux notebooks suivantes est de comparer l'utilisation de LDA avec deux input différentes : CountVectorizer et TfidfVectorizer.

Dans ce notebook, j'ai utilisé une approche similaire a celé du « 2.2.LDA_Cluster ». Mais avec un seul CountVectorizer avec le même stop_list, mais un contraint différent. Cet fois ci j'ai utilisé min_df= 0,1. Ce qui me génère une matrice de 36600 articles ou chaque article possède 205 « features ».

La fonction LDA utilise est la suivante :

```
lda_model = LatentDirichletAllocation(n_components=6, learning_method='bacth', random_state=0, verbose=0)
```

d. 3.2.LDA-TFIDF_content_cluster

Dans ce code j'ai utilisé TfidfVectorizer à la place de CountVectorizer pour effectuer toutes les vectorisations des données. En imposant les mêmes contraintes que en «3.1.LDA-CountVect_content_cluster » (cad min_df=0,1). L'utilisation de min_df est du a que cette contrainte réduise les « features » énormément. La matrice finale est de taille 36477 x 206. En utilisant moins des « features », j'ai réduit le temp de calcul énormément. Dans le cadre de l'étude principal, ces résultats ne sont pas utiles à cause de la perte d'information. Mais pour faire un comparaison entre les deux méthodes de vectorisation en égalité des conditions, les résultats sont tout à fait valables.

e. 4.bert_ZeroShotLabel

Dans ce notebook j'ai utilisé un modelé de « Zero-shot classification ». J'ai récupère le modèle de « Hugging Face » : MoritzLaurer/DeBERTa-v3-base-mnli-fever-anli · Hugging Face. Ce modelé permet de donner en input un texte et un groupe des labels. En return, il t'envoie les pourcentages correspondants aux classifications du texte selon les labels. En appliquant faisant une classification de toutes les articles publiés dans un pages web, on aura trouvé quels sont les tendances de chaque page web. Le seul problème c'est qu'effectuer la classification d'un seul texte prend au tour de 5- 10 seconds. Réaliser la classification avec 1000 articles aura pris un temp non négligeable. Apres avoir utilisé quelque méthodes de parallélisme et m'avoir aperçu que le temp reste non négligeable, j'ai abandonné l'idée.

4. Résultats

J'ai trouvé plusieurs résultats très intéressants et différentes conclusions qu'on peut extraire des différentes comparassions effectues lors de l'étude. Sur les objectifs principaux de l'étude, je voulais classifier tous les articles de façon sémantique pour pouvoir voir le comportement et tendances des différentes pages web au long du temps.

En utilisant les résultats trouves dans « 2.2.LDA_Cluster» et à l'aide de tableau j'ai trouvé les graphs suivantes :

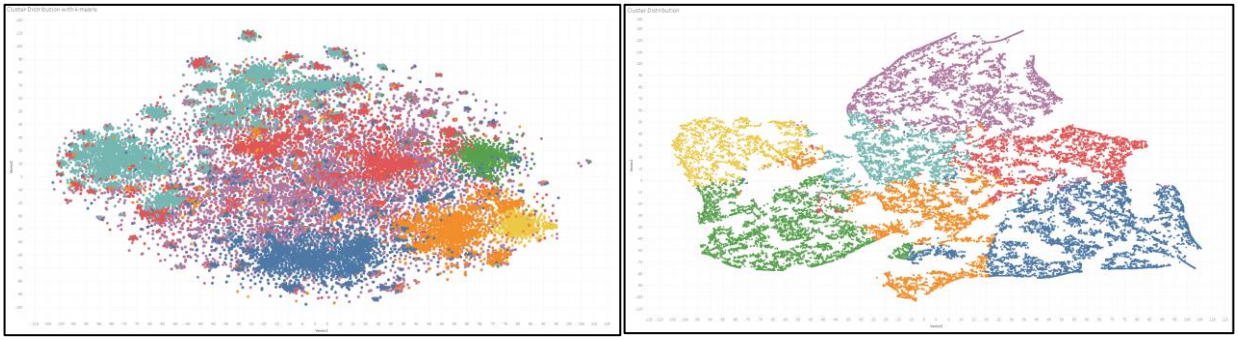


Figure 1: à gauche il y a la distribution du clusters generes avec k-means et à droite il y a la distribution des clusters generes avec LDA.

Nous pouvons regarder dans la **figure 1** directement que la distribution est plus efficace avec LDA qu'avec k-means. Donc les graphs finals sont générés avec les données de la classification de 36 000 articles à l'aide du Latent Dirichlet Allocation.

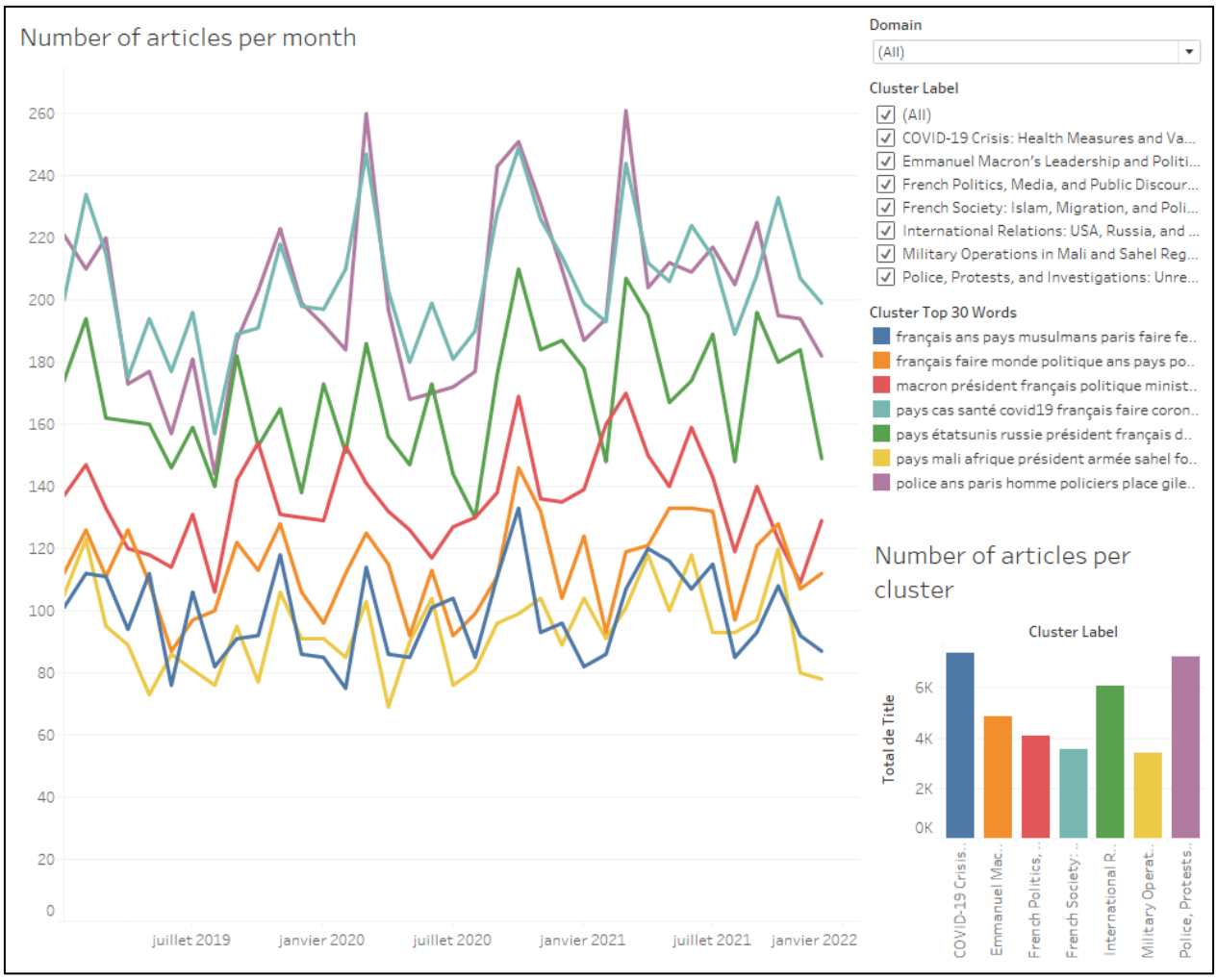
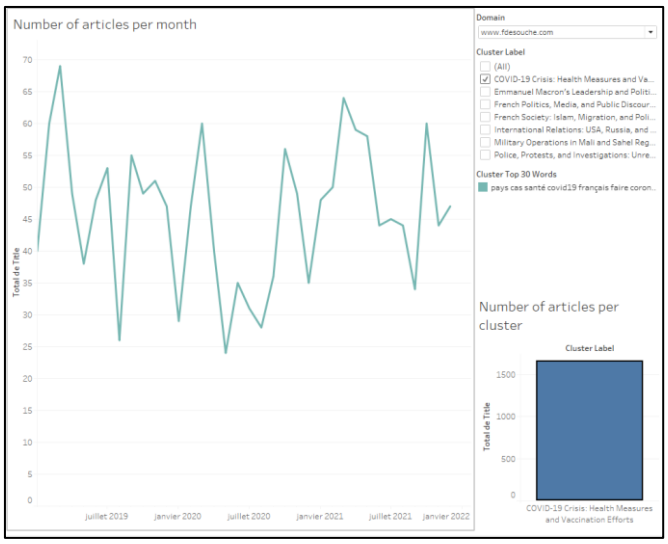
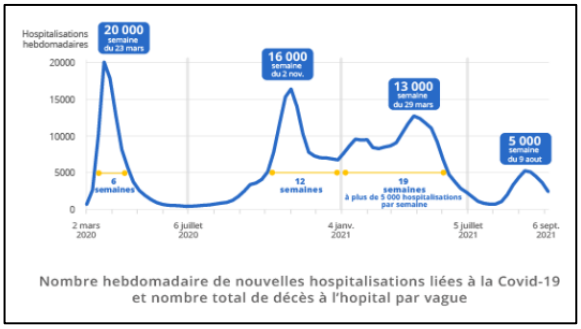


Figure 2 : Dashboard crée en Tableau. Dans lequel on a la distribution des quantités des articles publiés par mois selon le sujet entre début de 2019 et final de 2021.

Dans la **figure 2,** on peut voir le Dashboard génère avec les données du cluster de LDA. Dans le Dashboard on peut choisir le Domain entre : 'fr.sputniknews.africa', 'www.egaliteetreconciliation.fr', 'french.presstv.ir' , 'www.fdesouche.com' ou tous mélanges, ainsi que le cluster qu'on veut étudier.

Par exemple, si on choisit d'étudier la publication des articles qui parlent du COVID-19 en « fdesouche », on se trouve avec le cadre suivante :





Dans lequel on peut voir le diffèrent piques qu'on peut comparer avec les pics des hospitalisations liées à la COVID-19 en France. On peut voir une corrélation avec les piques plus hautes au début de 2019 et la fin de 2020. On peut faire plusieurs comparassent comme cela, aussi en trouvant des corrélation négatives. Ce un sujet très intéressent qu'on pourrait continuer à étudier.

Dans les résultats de la comparaison entre CountVectorizer et TfidfVectorizer, j'ai trouvé deux distributions des clustering différentes très similaires :

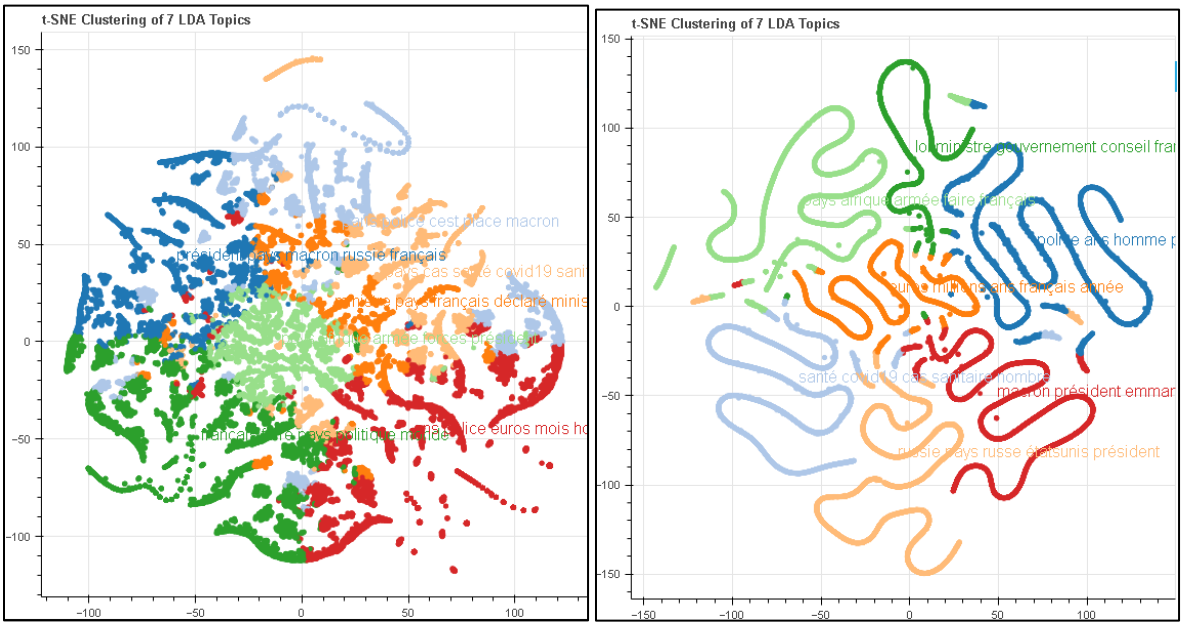


Figure 3 : Dans la figure à gauche il y a la distribution des clusters avec CountVectorizer et a droite il y a la distribution des clusters avec TfidfVectorizer.