**Expérimentation : Application du Topic Modeling sur la base de CV**

**Objectifs :**

* Faire une cartographie des métiers / compétences chez Devoteam sur base des dossiers de compétences des employés (CV)
* Être capable d’évaluer le degré de proximité de 2 CV

**Hypothèses :**

* CV pris dans leur globalité (pas de parsing)
* Le corpus pris pour l’étude ne doit comporter que des CV dans la même langue (français)
* Application d’algorithmes de Topic Modeling (et non clustering) pour cartographier les CV.

Le topic modeling est un modèle probabiliste permettant de déterminer des sujets ou thèmes abstraits d’un document. L’approche est donc différente du clustering qui range le document dans un seul cluster.

Dans le cas du topic modeling, on a donc plutôt une approche par « compétences » tandis que dans le clustering, c’est plutôt une approche par « postes » qui est utilisée.

1. Préparation des données
   1. Etape préalable manuelle

J’ai tout d’abord téléchargé en local sur mon poste l’ensemble des 778 CV pdf à notre disposition (dossier Drive « [Renamed-CV](https://drive.google.com/drive/u/0/folders/1Y1oUP4v3YozHyVRELIDkdd7bMl5Px2JB)»).

J’ai pu constater que le CV 574 été vide, ce qui me donnait des messages d’erreurs dans les traitements, je l’ai retiré.

* 1. Récupération des CV en français

Pour le traitement du langage naturel, il est important de ne pas mélanger des textes de différentes langues, ce qui nuira à la performance des algorithmes.

J’ai donc utilisé la librairie python langdetect qui permet de détecter la langue d’un texte.

J’utilise la méthode parser de Tika afin d’extraire le contenu des CV pdf.

Je stocke le contenu des CV en français dans une liste : **liste\_resumes**



On constate que 38 CV ont été éliminés car dans une autre langue que le français. Notre périmètre pour cette étude sera donc les 739 CV restants.

1. Préparation de la liste des stop\_words



Les stop\_words sont des mots très couramment utilisés et qui n’apportent aucune information utile. Ils sont donc nuisibles pour notre problème : ils « polluent » et augmentent considérablement la complexité des calculs. Il faut donc les supprimer.

Je suis parti de la librairie stop\_words qui contient des listes de stopwords dans plusieurs langues dont le français qui nous intéresse.

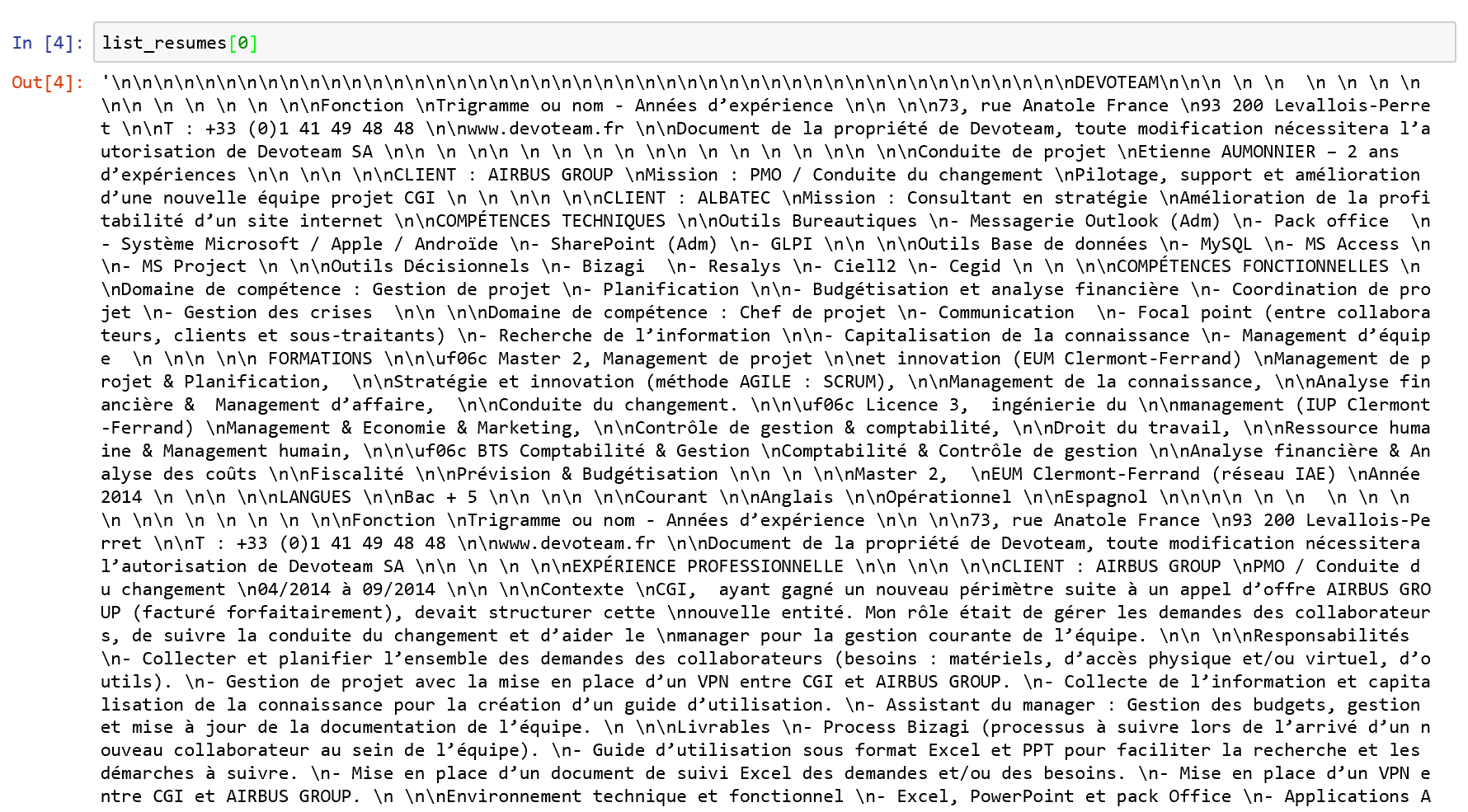
A cette liste, j’ajoute un certain nombre de mots spécifiques à notre corpus, qui réapparaissent souvent dans les CV : les nombres de 0 à 999, des mots tels que « devoteam », « France »…

On applique ensuite à la liste les transformations suivantes :

* Racinisation (FrenchStemmer)
* Mise en minuscule
* Normalisation : suppression des accents

Ces stopwords sont stockés dans la variable **stem\_stopwords**.

1. Extraction des mots, racinisation, normalisation et suppression des stop\_words



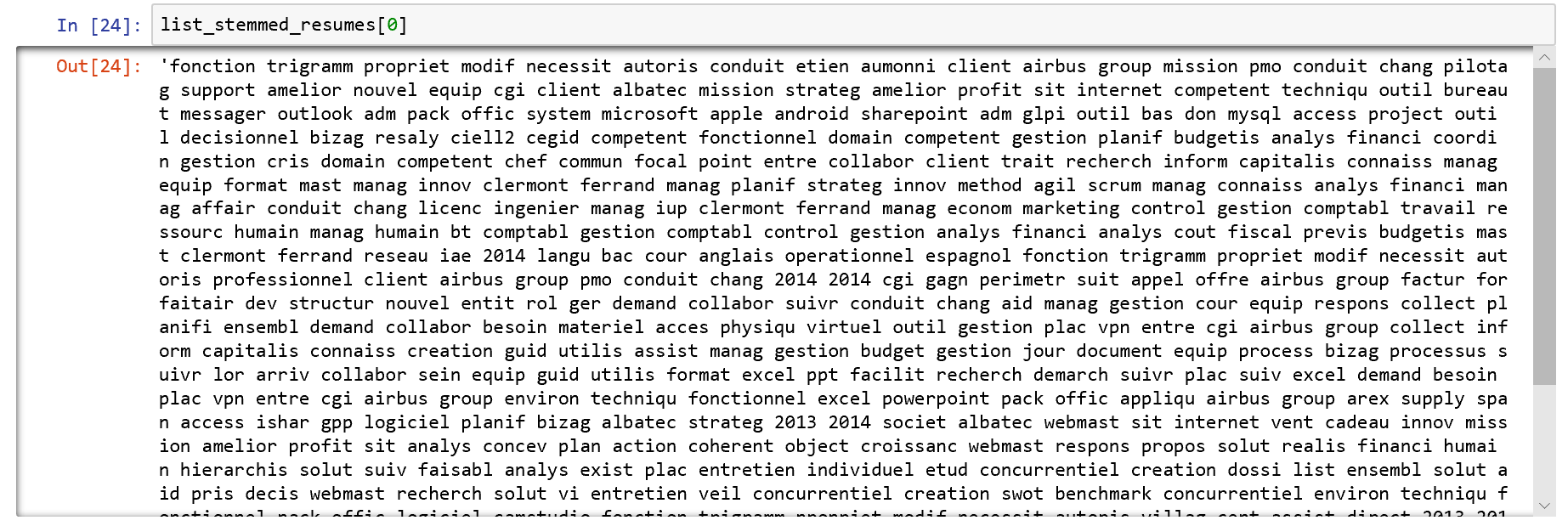
Sur un exemple de contenu de CV, on constate qu’un premier nettoyage est nécessaire : la suppression des « \n ».

On applique donc pour chacun des CV les étapes suivantes :

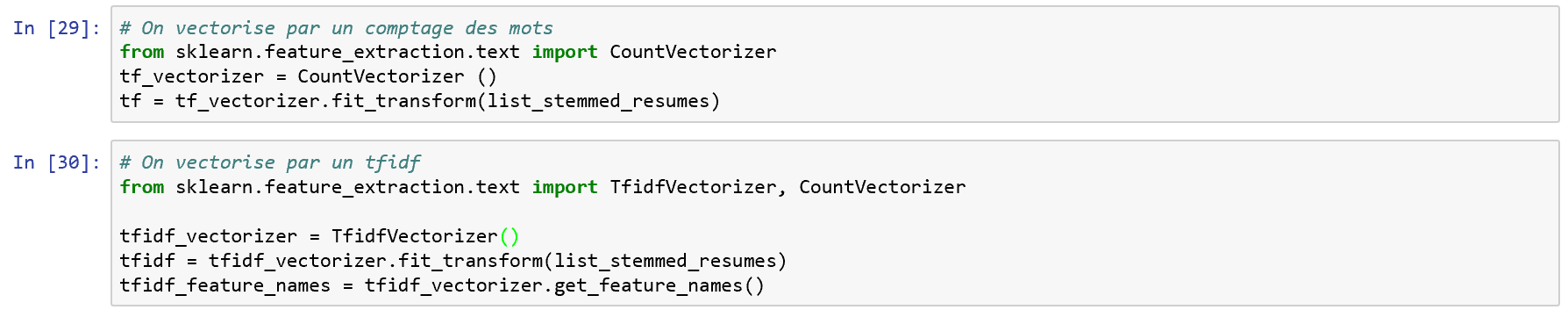
* Suppression des « \n »
* A l’aide de RegexpTokeniser de nltk, extraction des mots du CV
* Pour chaque mot, on applique les mêmes étapes que pour les stopwords :
  + Racinisation (FrenchStemmer)
  + Mise en minuscule
  + Normalisation : suppression des accents
* Reconstitution du CV nettoyé (avec suppression des stopwords) et ajout dans une liste **list\_stemmed\_resumes**.



Voici l’exemple d’un CV après ces étapes de nettoyage :



1. Application des méthodes BoW (Bag of Words) et TFIDF



1. Application du LDA (Latent Dirichlet Allocation)

La LDA suppose que chaque document du corpus étudié est un mélange d'un petit nombre de sujets ou thèmes, et que la création de chaque mot est attribuable (probabilités) à l'un des thèmes du document.

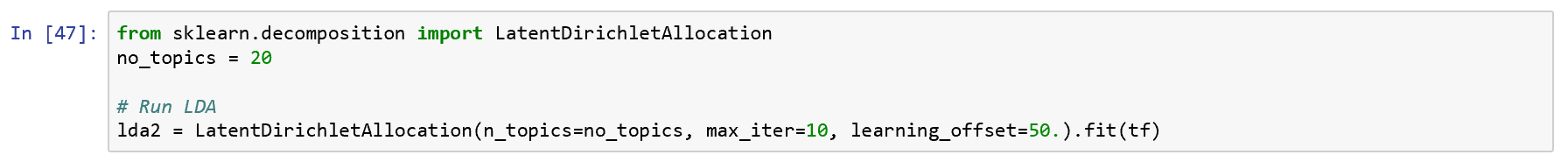
Comme elle utilise une méthode probabiliste, on ne peut appliquer qu’un BoW en entrée de l’algorithme et non un TFIDF.

* 1. Librairie Sklearn

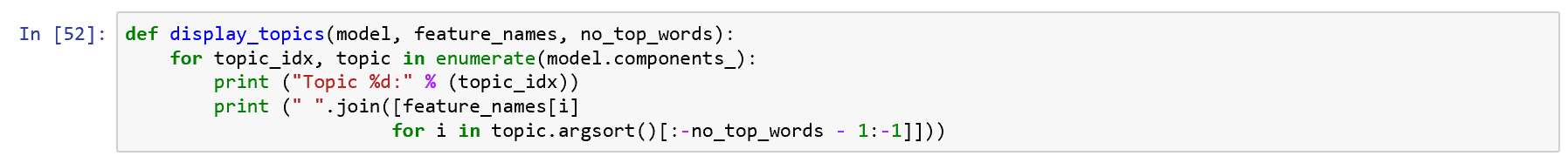
Dans un premier temps, j’ai utilisé la librairie [**sklearn**](https://medium.com/mlreview/topic-modeling-with-scikit-learn-e80d33668730) qui propose une implémentation de la LDA.

Il faut préciser en entrée de l’algorithme ne nombre de topic que l’on souhaite.

J’ai testé pour plusieurs nombres de topic. Pour l’exemple ici, je prends **no\_topics = 20.** Ci-dessous l’entrainement du modèle :



Afin d’afficher les résultats, je créé une fonction qui permet d’afficher pour chacun des topic, le Top des mots les plus fréquents. Voici le code correspondant :



Les résultats obtenus pour 20 topics sont les suivants :



La difficulté avec cette libraire utilisée est que l’optimisation du nombre de topics et l’évaluation des résultats ne semble pouvoir se faire que manuellement.

Après recherches, nous avons découvert que la **cohérence** pouvait servir de mesure de performance.

La cohérence mesure la performance d’un modèle selon le nombre de topic choisit. Afin de déterminer le nombre de topics optimal, il faut maximiser la coherence (méthode d’Elbow). Explication plus en détail :    
[https://stats.stackexchange.com/questions/375062/how-does-topic-coherence-score-in-lda-intuitively-makes-sense](https://www.google.com/url?q=https://stats.stackexchange.com/questions/375062/how-does-topic-coherence-score-in-lda-intuitively-makes-sense&sa=D&source=hangouts&ust=1553769946732000&usg=AFQjCNGliWmkJ7hK9QnUKfi-uZOmvD9Bnw)

L’implémentation du LDA de sklearn ne met pas à disposition dans sa mallette d’outils le facteur de cohérence.

* 1. Librairie Gensim

La librairie Gensim possède une implémentation du LDA qui semble plus complète et qui permet le calcul du facteur de cohérence, comme le montre l’article suivant :

[https://towardsdatascience.com/basic-nlp-on-the-texts-of-harry-potter-topic-modeling-with-latent-dirichlet-allocation-f3c00f77b0f5](https://www.google.com/url?q=https://towardsdatascience.com/basic-nlp-on-the-texts-of-harry-potter-topic-modeling-with-latent-dirichlet-allocation-f3c00f77b0f5&sa=D&source=hangouts&ust=1553769946732000&usg=AFQjCNH4PWwBrZmqM87PGe3sM7VT5hFfRg)

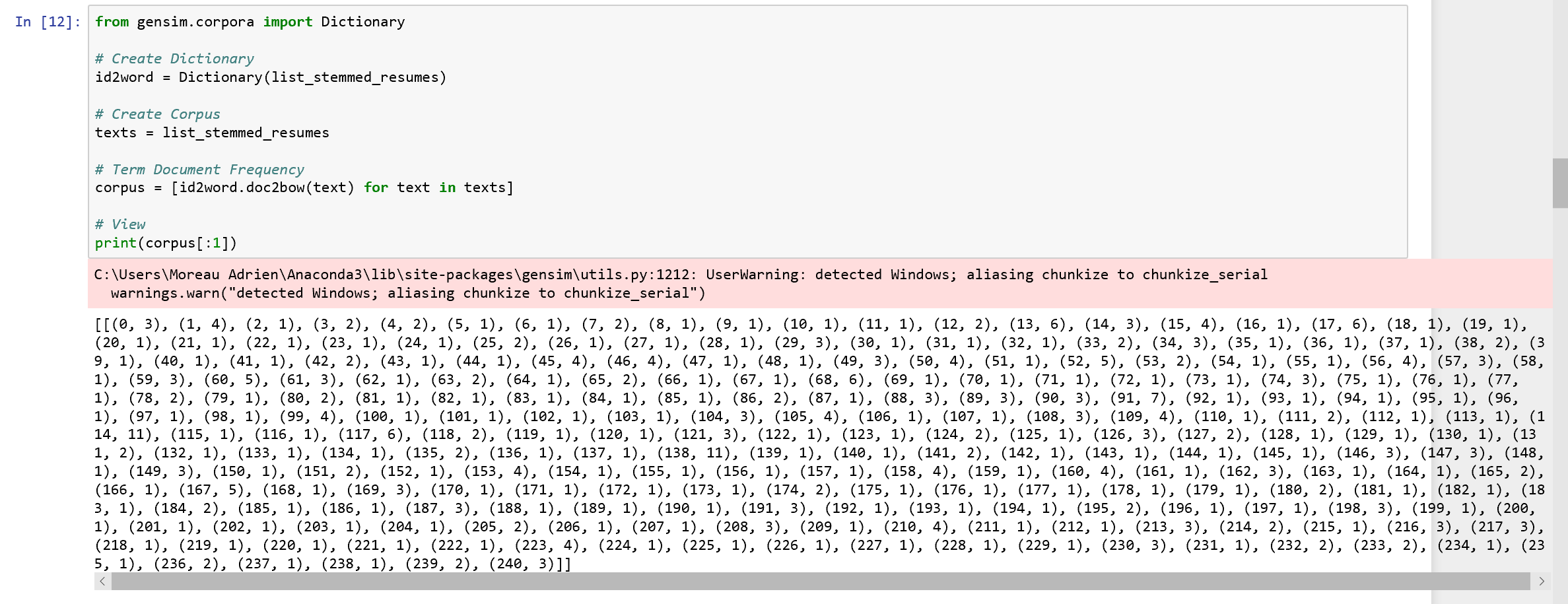
C’est pourquoi j’ai choisi de me rabattre vers Gensim plutôt que sklearn.

L’étape de préparation des données est quasiment la même que pour le LDA de sklearn.

On a juste besoin en plus, en input de l’algorithme, de donner :

* Le dictionnaire de mots du corpus **id2word**, que l’on récupère via la méthode Dictionnary
* La liste de textes **texts**, qui correspond à liste **list\_stemmed\_resumes** créée précédemment
* La liste des Bag of words **corpus,** que l’on avait précédemment réalisé avec la méthode **countVectorizer de sklearn** et que l’on fait cette fois-ci avec **doc2bow de gensim**

Voici le code pour créer ces 3 éléments ainsi que le bag of words d’1 CV affiché.



On créé ensuite une fonction **compute\_coherence\_value** qui évalue le facteur de cohérence du modèle de LDA entrainé sur le corpus pour un nombre de topics allant de *start* à *limit.*

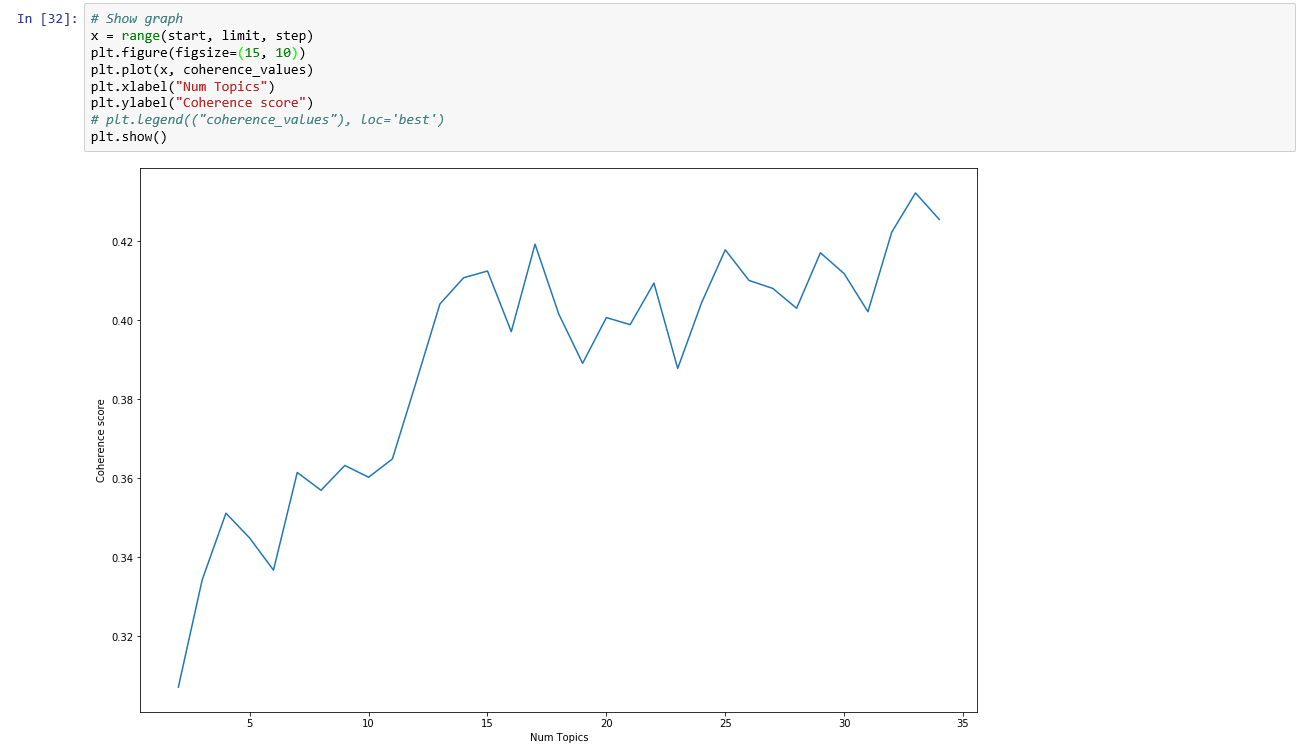


On applique ensuite cette fonction à notre corpus et notre dictionnaire de mots et pour un nombre de topics allant de **2 à 35**.



On affiche ensuite les résultats sous forme de graphe avec en abscisse le nombre de topics et en ordonnées le facteur de cohérence à optimiser.

Voici le code et les résultats obtenus :



On cherche généralement à prendre le nombre de topic minimum qui permettent de maximiser le facteur de cohérence, ce qui revient à faire une méthode d’Elbow.

Ici on choisit comme **nombre de topic 15**.

Maintenant qu’on a trouvé le nombre de topic optimal pour notre problème, représentons les topics pour donner un peu plus de sens aux résultats obtenus.

On peut tout d’abord, comme on l’avait fait pour le LDA de sklearn, représenter le top des mots les plus fréquents pour chaque topic.



On voit que les résultats sont assez intéressants et qu’on peut interpréter les topics.

Topic 0 : requètage de base de données

Topic 1 : virtualisation d’applications – VM – administration des serveurs

Topic 2 : Infrastructure d’exploitation – software de gestion des systèmes (BMC – HP)

Topic 3 : Poste de travail – Packaging d’applications

Topic 4 : Architecture – aéronautique

Topic 5 : Data engineering – data science

Topic 6 : Gestion de projet fonctionnelle

Topic 7 : Sécurité et risques informatiques

Topic 8 : Infrastructure – Réseau

Topic 9 : coordination – gestion de projet sur des sujets d’infra

Topic 10 : ??

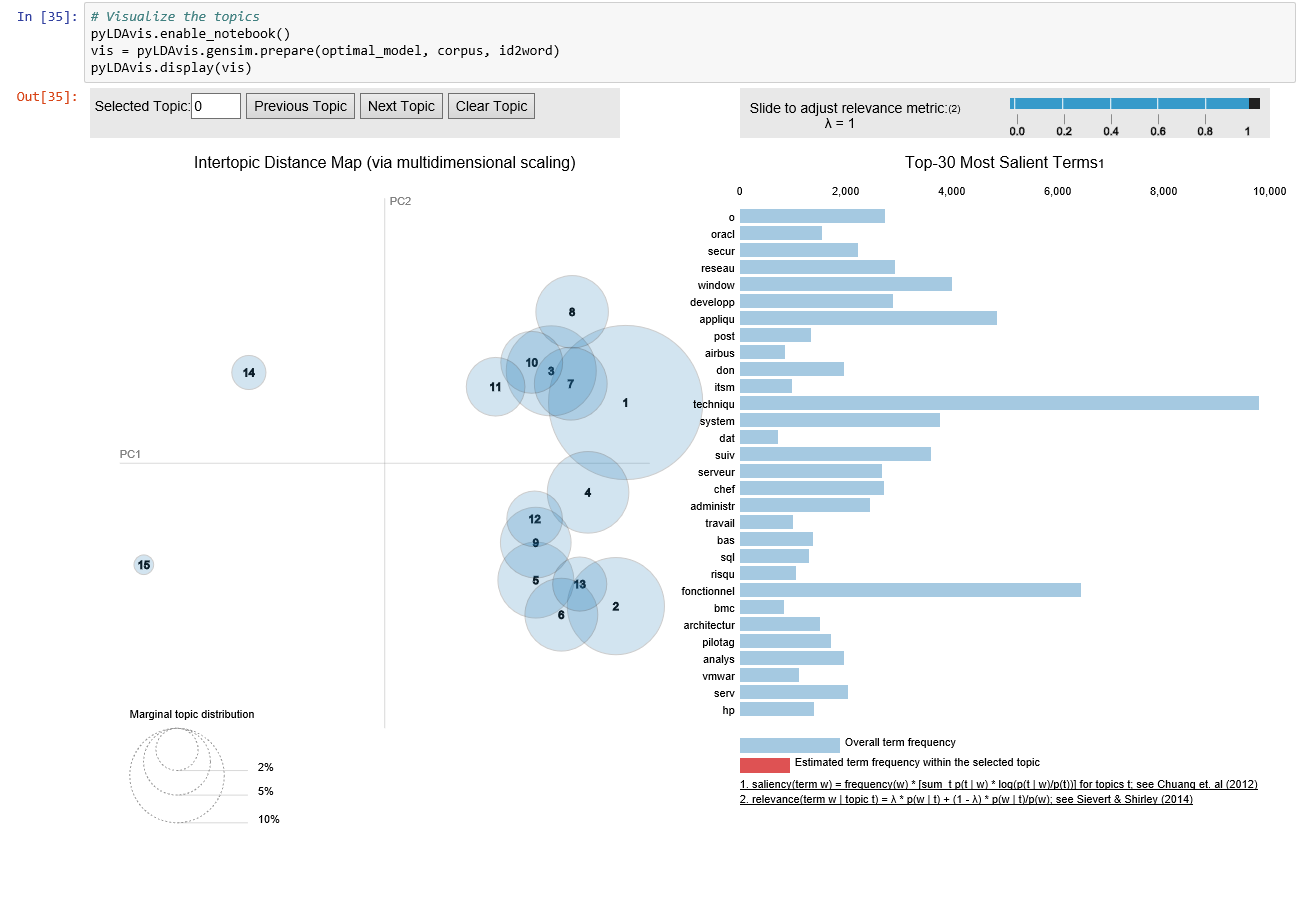
Topic 11 : Gestion du support (incidents)

Topic 12 : Gestion infra - Cloud

Topic 13 : Développeur java – web

Topic 14 : administration – supervision système

Afin de visualiser de manière interactive les topics, on utilise la librairie **pyLDAvis.** Cette représentation permet en plus d’afficher le top des mots fréquents par topic d’afficher la distribution des topics.



On voit ici que les topics sont assez rapprochés les uns des autres.

Améliorations possibles de l’approche :

* Améliorer la préparation des données (supprimer les mots qui apportent du bruit)
* Faire une normalisation (centrage et réduction des data)
* Utiliser des bigrammes ou trigrammes

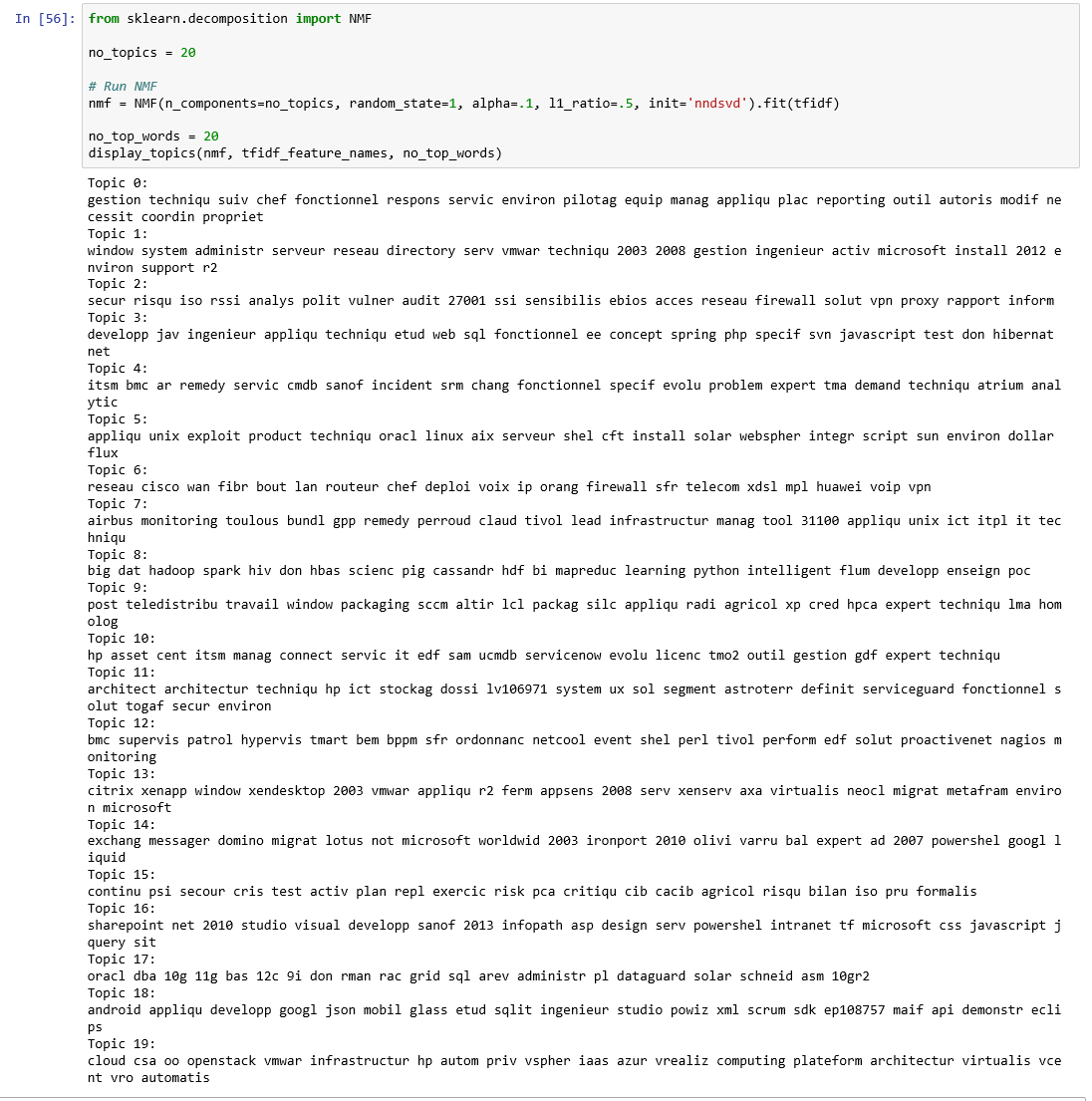
1. Application du NMF (Non-negative Matrix Factorization)

J’ai eu l’idée de tester un autre algorithme de topic modeling qui est le NMF pour comparer les résultats avec la LDA et de choisir le plus adapté.

On utilise la librairie Sklearn qui en possède une implémentation.

Pour le NMF, la littérature conseille fortement d’utiliser le TFIDF et non le Bag of Words, en entrée de l’algorithme. Nous utiliserons donc l’objet **tfidf** défini dans la partie 2).

On entraine le modèle pour 20 topics et on représente le top des mots par topic à l’aide de la fonction display\_topics définie en 3.a).



Les résultats semblent assez prometteurs et les topics bien discernables.

Seulement, comme nous avons fait pour LDA, il nous faudrait trouver une fonction d’évaluation qui permettent nous seulement d’évaluer les résultats de l’algorithme mais aussi trouver le nombre de topics optimal.

Améliorations possibles de l’approche :

* Améliorer la préparation des données (supprimer les mots qui apportent du bruit)
* Faire une normalisation (centrage et réduction des data)
* Utiliser des bigrammes ou trigrammes
* Trouver une fonction d’évaluation pour le NMF comme on a pu le faire avec la LDA