

Extraire les thèmes (topics) de documents via LDA

Dans ce notebook nous nous situons dans le cadre de l'apprentissage non supervisé (nous n'avons pas de label de classe sur les données) même si, comme nous allons le voir dans la suite, les résultats obtenus peuvent également être utilisés en apprentissage supervisé.

L'objectif est d'extraire à partir des documents les thèmes ou sujets cachés (*topics*) et pour cela nous allons utiliser le *topic model* LDA (*Latent Dirichlet Allocation* ou *Allocation de Dirichlet Latente*). Il existe de nombreuses classes disponibles pour LDA. Par exemple, scikit learn propose ses propres classes (e.g. <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html>). Dans ce notebook, nous introduisons une nouvelle librairie *gensim* qui outre de nombreuses fonctions ou classes pour manipuler les données textuelles notamment lors des pré-traitements, propose toutes les fonctionnalités pour extraire efficacement les topics. De nombreuses informations sur *gensim* sont disponibles ici : <https://radimrehurek.com/gensim/>

▼ Installation

Avant de commencer, il est nécessaire de déjà posséder dans son environnement toutes les librairies utiles. Dans la seconde cellule nous importons toutes les librairies qui seront utiles à ce notebook. Il se peut que, lorsque vous lanciez l'exécution de cette cellule, une soit absente. Dans ce cas il est nécessaire de l'installer. Pour cela dans la cellule suivante utiliser la commande :

```
! pip install nom_librairie
```

Attention : il est fortement conseillé lorsque l'une des librairies doit être installer de relancer le kernel de votre notebook.

Remarque : même si toutes les librairies sont importées dès le début, les librairies utiles pour des fonctions présentées au cours de ce notebook sont ré-importées de manière à indiquer d'où elles viennent et ainsi faciliter la réutilisation de la fonction dans un autre projet.

Ici nous avons besoin de 2 librairies : *gensim* et *PyLDAvis* qui permet de visualiser les topics.

```
# utiliser cette cellule pour installer les librairies manquantes
# pour cela il suffit de taper dans cette cellule : !pip install nom_librairie_manquante
# d'exécuter la cellule et de relancer la cellule suivante pour voir si tout se passe bien
# recommencer tant que toutes les librairies ne sont pas installées ...

#!pip install ..
!pip install pyLDAvis==2.1.2
!pip install -U gensim
```

```
# ne pas oublier de relancer le kernel du notebook
```

```
# ne pas oublier de relancer le kernel du notebook
```

```
Collecting pyLDAvis==2.1.2
```

```
  Downloading pyLDAvis-2.1.2.tar.gz (1.6 MB)
```

```
1.6/1.6 MB 7.8 MB/s eta 0:00:00
```

```
  Preparing metadata (setup.py) ... done
```

```
Requirement already satisfied: wheel>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (0.41.2)
```

```
Requirement already satisfied: numpy>=1.9.2 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (1.23.5)
```

```
Requirement already satisfied: scipy>=0.18.0 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (1.10.1)
```

```
Requirement already satisfied: pandas>=0.17.0 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (1.5.3)
```

```
Requirement already satisfied: joblib>=0.8.4 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (1.3.2)
```

```
Requirement already satisfied: Jinja2>=2.7.2 in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (3.1.2)
```

```
Requirement already satisfied: numexpr in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (2.8.5)
```

```
Requirement already satisfied: pytest in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (7.4.1)
```

```
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from pyLDAvis==2.1.2) (0.18.3)
```

```
Collecting funcy (from pyLDAvis==2.1.2)
```

```
  Downloading funcy-2.0-py2.py3-none-any.whl (30 kB)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=2.7.2->pyLDAvis==2.1.2) (2.1.3)
```

```
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.17.0->pyLDAvis==2.1.2) (2.8.2)
```

```
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.17.0->pyLDAvis==2.1.2) (2023.3.post1)
```

```
Requirement already satisfied: iniconfig in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDAvis==2.1.2) (2.0.0)
```

```
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDAvis==2.1.2) (23.1)
```

```
Requirement already satisfied: pluggy<2.0,>=0.12 in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDAvis==2.1.2) (1.3.0)
```

```
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDAvis==2.1.2) (1.1.3)
```

```
Requirement already satisfied: tomli>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDAvis==2.1.2) (2.0.1)
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>=0.17.0->pyLDAvis==2.1.2) (1.16.0)
```

```
Building wheels for collected packages: pyLDAvis
```

```
  Building wheel for pyLDAvis (setup.py) ... done
```

```
  Created wheel for pyLDAvis: filename=pyLDAvis-2.1.2-py2.py3-none-any.whl size=97718 sha256=c28db81f9194a731d2356afd39ac0d50e80366ca341538
```

```
  Stored in directory: /root/.cache/pip/wheels/d9/93/d6/16c95da19c32f037fd75135ea152d0df37254c25cd1a8b4b6c
```

```
Successfully built pyLDAvis
```

```
Installing collected packages: funcy, pyLDAvis
```

```
Successfully installed funcy-2.0 pyLDAvis-2.1.2
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.10/dist-packages (4.3.2)
```

```
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.23.5)
```

```
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from gensim) (1.10.1)
```

```
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from gensim) (6.3.0)
```

```
# Importation des différentes librairies utiles pour le notebook
```

```
#Sickit learn met régulièrement à jour des versions et
```

```
#indique des futurs warnings.
```

```
#ces deux lignes permettent de ne pas les afficher.
```

```
import warnings
```

```
warnings.filterwarnings("ignore", category=FutureWarning)
```

```
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
warnings.filterwarnings('ignore', 'SelectableGroups dict interface')
```

```
with warnings.catch_warnings():
    warnings.simplefilter('ignore')
    # do something here and its warning is suppressed

# librairies générales
import pandas as pd
import re
import time
import numpy as np
import string
import base64

# librairie affichage
import matplotlib.pyplot as plt
import seaborn as sns

# librairies pour LDA
# exceptionnellement la librairie suivante n'est pas chargée car la librairie est
# un peu ancienne et pose des problèmes de deprecated warnings avec IPython
#import pyLDAvis

# librairies scikit learn
import sklearn

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report

# librairies NLTK
import nltk
from nltk.corpus import stopwords
from nltk import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.corpus import wordnet

# librairie spacy
```

```
import spacy

# librairies de gensim
import gensim
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
from gensim.models import Phrases
from gensim.models.phrases import Phraser
from gensim import corpora
from gensim import models

nltk.download('wordnet')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

# chargement spacy en (english)
!python3 -m spacy download en
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
```

```
2023-09-07 22:51:22.837301: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
```

```
△ As of spaCy v3.0, shortcuts like 'en' are deprecated. Please use the
full pipeline package name 'en_core_web_sm' instead.
```

```
Collecting en-core-web-sm==3.6.0
```

```
  Downloading https://github.com/explosion/spacy-models/releases/download/en\_core\_web\_sm-3.6.0/en\_core\_web\_sm-3.6.0-py3-none-any.whl (12.8 MB)
    12.8/12.8 MB 16.6 MB/s eta 0:00:00
```

```
Requirement already satisfied: spacy<3.7.0,>=3.6.0 in /usr/local/lib/python3.10/dist-packages (from en-core-web-sm==3.6.0) (3.6.1)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-we
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-we
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==
Requirement already satisfied: thinc<8.2.0,>=8.1.8 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm
Requirement already satisfied: typer<0.10.0,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3
```

```

Requirement already satisfied: pathy>=0.10.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.6.0) (
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-s
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.6.0) (
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm
Requirement already satisfied: pydantic!=1.8,!<1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->e
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.6.0) (3.1.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.6.0) (67.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm==3.6.0)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from spacy<3.7.0,>=3.6.0->en-core-web-sm
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!<1.8.1,<3.0.0,>=1.7.4
Requirement already satisfied: pydantic-core==2.6.3 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!<1.8.1,<3.0.0,>=1.7.4->
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydantic!=1.8,!<1.8.1,<3.0.0,>=1.7
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.7.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.7.0,>=3.6.0->
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.7.0,>=3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.7.0,>=3
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.10/dist-packages (from thinc<8.2.0,>=8.1.8->spacy<3.7.0,>=3.6.0
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.10/dist-packages (from thinc<8.2.0,>=8.1.8->spacy<3.7.0,>
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer<0.10.0,>=0.3.0->spacy<3.7.0,>=3.6
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->spacy<3.7.0,>=3.6.0->en-core-web-sm
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')

```

Pour pouvoir sauvegarder sur votre répertoire Google Drive, il est nécessaire de fournir une autorisation. Pour cela il suffit d'exécuter la ligne suivante et de saisir le code donné par Google.

```

# pour monter son drive Google Drive local
from google.colab import drive
drive.mount('/content/gdrive')

```

Mounted at /content/gdrive

Corriger éventuellement la ligne ci-dessous pour mettre le chemin vers un répertoire spécifique dans votre répertoire Google Drive :

```
import sys
my_local_drive='/content/gdrive/My Drive/Colab Notebooks/ML2_2023_2024'
# Ajout du path pour les librairies, fonctions et données
sys.path.append(my_local_drive)
# Se positionner sur le répertoire associé
%cd $my_local_drive

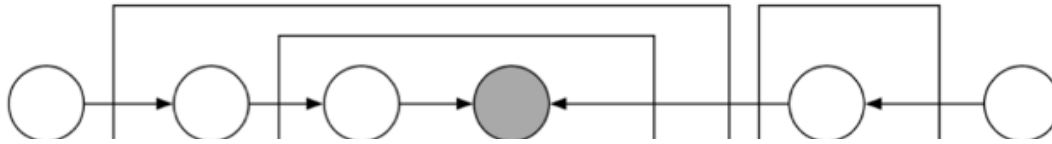
%pwd

/content/gdrive/My Drive/Colab Notebooks/ML2_2023_2024
'/content/gdrive/My Drive/Colab Notebooks/ML2_2023_2024'
```

```
# fonctions utilities (affichage, confusion, etc.)
from MyNLPUtilities import *
```

▼ Extraction de topics : utilisation de LDA

L'Allocation de Dirichlet Latente (*Latent Dirichlet Allocation*) ou LDA est un modèle probabiliste génératif qui permet de décrire des collections de textes et qui a été introduit par Blei et al. en 2003 (Blei, D. M., A. Y. Ng, et M. I. Jordan (2003). Latent Dirichlet Allocation. Journal of machine learning research : JMLR 3, 993–1022.). Il fait partie des modèles de type *topic models* comme L'Analyse Sémantique Latente (LSA - *Latent semantic analysis*) ou l'Analyse Sémantique Latente Probabiliste (PLSA - *Probabilistic latent semantic analysis*), qui cherchent à découvrir des thématiques cachées (*les topics*) dans les documents. Ainsi le principe est que chaque document peut être décrit par une distribution de topics et chaque topic peut aussi être décrit par une distribution de mots. Plus précisément, chaque document est modélisé par un mélange de thèmes générateur des mots du document. LDA est un modèle Bayésien à trois couches (c.f. figure) qui utilise l'approche Bag of Words pour chaque document d du corpus D défini par (w_1, \dots, w_D) comme un N-uplet de mots, $w_d = (w_1, \dots, w_N)$. A chaque mot $w(d, n)$ est alors associé un topic représenté par la variable $z_{(d,n)}$. Θ_d représente la distribution de topics du document d . Des hyperparamètres, α et η , définissent l'*a priori* sur Θ et β où β_k décrit la distribution du topic k .



▼ Des prétraitements de données

| | | | | **K** |

Comme pour la classification, il est indispensable de pré-traiter nos données avant de pouvoir appliquer LDA. Dans le notebook "2 - Traitement de données textuelles", nous avons vu que NLTK ou spacy offraient de nombreuses fonctionnalités. Dans cette partie nous introduisons quelques fonctions pratiques issues de la librairie gensim qui permettent aussi de faire des pré-traitements.

Il existe bien entendu de nombreuses locutions qui sont composées de deux mots (e.g. 'apprentissage automatique', 'machine learning') ou de trois mots (e.g. 'chemin de fer', 'pros and cons'). Nous avons vu précédemment qu'il était possible d'appliquer des n-grammes pour déterminer les mots qui se suivent. Gensim propose différentes fonctions pour extraire simplement les bigrams, trigrams etc :

1. La fonction *Phrase* détecte automatiquement les termes en commun dans un ensemble de phrases (notamment les n-grammes). Les paramètres `min_count`, `threshold` correspondent respectivement au nombre minimal d'occurrences d'un mot ou d'un ngramme pour être retenu et au seuil de score à atteindre pour former une phrase (le seuil de score par défaut est 10.0, la méthode pour calculer le score par défaut est celle décrite dans <https://arxiv.org/abs/1310.4546>). Plus le seuil est haut moins il y a de phrases.
2. La fonction *Phraser* permet d'optimiser la recherche dans la fonction précédente. Généralement elle est utilisée en même temps pour des raisons de performances.

Pour connaître les différentes classes, fonctions disponibles et les paramètres associés, voir ici

<https://radimrehurek.com/gensim/models/phrases.html>

```
import gensim
from gensim.models import Phrases
from gensim.models.phrases import Phraser

texts = ["in software engineering and computer science abstract data type are often used",
         "machine learning is sometimes using abstract data type",
         "an abstract data type is defined by its behavior",
         "machine learning has now become a trending research area",
         "abstract data type may be defined as a class of objects whose logical behavior is defined by a set of values and a set of operations ",
         "an abstract data type is a theoretical concept in computer science"]
# il faut au préalable découper les textes en tokens
```

```

tokens = [doc.split(" ") for doc in texts]

# extraction des bigrams (min_count précise qu'ils doivent apparaître au moins 1 fois)
bigram = Phrases(tokens, min_count=1, delimiter=' ')#threshold=1, delimiter=b' ')
bigram_phraser = Phraser(bigram)

# extraction des trigrams à partir des bigrams précédents
trigram = Phrases(bigram_phraser[tokens], min_count=1)#, delimiter=b' ')
trigram_phraser = Phraser(trigram)

# Recupération des différents composants des phrases
bigram_tokens = []
for sent in tokens:
    bigram_tokens.append(bigram_phraser[sent])
    bigram_tokens.append(trigram_phraser[bigram[sent]])

print ("Les phrases et les bi/trigrammes associés :\n")
for i in range (len(bigram_tokens)):
    print (bigram_tokens[i])

```

Les phrases et les bi/trigrammes associés :

```

['in', 'software', 'engineering', 'and', 'computer science', 'abstract data', 'type', 'are', 'often', 'used']
['in', 'software', 'engineering', 'and', 'computer science', 'abstract data_type', 'are', 'often', 'used']
['machine learning', 'is', 'sometimes', 'using', 'abstract data', 'type']
['machine learning', 'is', 'sometimes', 'using', 'abstract data_type']
['an', 'abstract data', 'type', 'is', 'defined by', 'its', 'behavior']
['an', 'abstract data_type', 'is_defined by', 'its', 'behavior']
['machine learning', 'has', 'now', 'become', 'a', 'trending', 'research', 'area']
['machine learning', 'has', 'now', 'become', 'a', 'trending', 'research', 'area']
['abstract data', 'type', 'may', 'be', 'defined', 'as', 'a', 'class', 'of', 'objects', 'whose', 'logical', 'behavior', 'is', 'defined by',
['abstract data_type', 'may', 'be', 'defined', 'as', 'a', 'class', 'of', 'objects', 'whose', 'logical', 'behavior', 'is_defined by', 'a', '
['an', 'abstract data', 'type', 'is', 'a', 'theoretical', 'concept', 'in', 'computer science']
['an', 'abstract data_type', 'is', 'a', 'theoretical', 'concept', 'in', 'computer science']

```

A partir du moment où les textes ont été nettoyés, les bi, tri-grammes trouvés, il est nécessaire de définir le vocabulaire (i.e. l'ensemble des mots, composition de mots) sur lequel LDA va apprendre.

Pour cela il existe la classe *Dictionary* qui va, à partir de l'ensemble des tokens appris précédemment, créer un dictionnaire contenant l'ensemble des mots ainsi qu'un index associé.

Cet index est utilisé pour transformer le corpus initial.


```
import gensim
from gensim import corpora

# bigram_tokens a été obtenu à l'étape précédente
dictionnaire = gensim.corpora.Dictionary(bigram_tokens)

# pour connaitre le nombre de chaque token
print("Les différents mots du dictionnaire : \n",dictionnaire.token2id)

# conversion des mots extraits du texte en vecteur de type bag of words
corpus = [dictionnaire.doc2bow(text) for text in bigram_tokens]

print ("\nLa matrice obtenue après tranformation où chaque ligne correspond à un document \n")
for doc in corpus:
    print([[dictionnaire[id], freq] for id, freq in doc])

# conversion via TF-IDF
tfidf_model = models.TfidfModel(corpus)
corpus_tfidf = tfidf_model[corpus]
print ("\nLa matrice obtenue après tranformation TF-IDF")
for doc in tfidf_model[corpus]:
    print([[dictionnaire[id], np.around(freq,2)] for id, freq in doc])
```

```
{'abstract data': 0, 'and': 1, 'are': 2, 'computer science': 3, 'engineering': 4, 'in': 5, 'often': 6, 'software': 7, 'type': 8, 'used': 9}
```

```
[['abstract data', 1], ['and', 1], ['are', 1], ['computer science', 1], ['engineering', 1], ['in', 1], ['often', 1], ['software', 1], ['type', 1], ['used', 1], ['abstract data', 1], ['type', 1], ['is', 1], ['machine learning', 1], ['sometimes', 1], ['using', 1]]
[['abstract data_type', 1], ['is', 1], ['machine learning', 1], ['sometimes', 1], ['using', 1]]
[['abstract data', 1], ['type', 1], ['is', 1], ['an', 1], ['behavior', 1], ['defined by', 1], ['its', 1]]
[['abstract data_type', 1], ['an', 1], ['behavior', 1], ['its', 1], ['is_defined by', 1]]
[['machine learning', 1], ['a', 1], ['area', 1], ['become', 1], ['has', 1], ['now', 1], ['research', 1], ['trending', 1]]
[['machine learning', 1], ['a', 1], ['area', 1], ['become', 1], ['has', 1], ['now', 1], ['research', 1], ['trending', 1]]
```

```
[['abstract data', 1], ['and', 1], ['type', 1], ['is', 1], ['behavior', 1], ['defined by', 1], ['a', 3], ['as', 1], ['be', 1], ['class', 1]
[['and', 1], ['abstract data_type', 1], ['behavior', 1], ['is_defined by', 1], ['a', 3], ['as', 1], ['be', 1], ['class', 1], ['defined', 1]
[['abstract data', 1], ['computer science', 1], ['in', 1], ['type', 1], ['is', 1], ['an', 1], ['a', 1], ['concept', 1], ['theoretical', 1]]
[['computer science', 1], ['in', 1], ['abstract data_type', 1], ['is', 1], ['an', 1], ['a', 1], ['concept', 1], ['theoretical', 1]]
```

La matrice obtenue après tranformation TF-IDF

```
[['abstract data', 0.19], ['and', 0.24], ['are', 0.39], ['computer science', 0.24], ['engineering', 0.39], ['in', 0.24], ['often', 0.39], [
[['and', 0.24], ['are', 0.4], ['computer science', 0.24], ['engineering', 0.4], ['in', 0.24], ['often', 0.4], ['software', 0.4], ['used', 0
[['abstract data', 0.28], ['type', 0.28], ['is', 0.22], ['machine learning', 0.35], ['sometimes', 0.58], ['using', 0.58]]
[['abstract data_type', 0.29], ['is', 0.23], ['machine learning', 0.37], ['sometimes', 0.6], ['using', 0.6]]
[['abstract data', 0.27], ['type', 0.27], ['is', 0.21], ['an', 0.33], ['behavior', 0.33], ['defined by', 0.54], ['its', 0.54]]
[['abstract data_type', 0.28], ['an', 0.35], ['behavior', 0.35], ['its', 0.58], ['is_defined by', 0.58]]
[['machine learning', 0.24], ['a', 0.15], ['area', 0.39], ['become', 0.39], ['has', 0.39], ['now', 0.39], ['research', 0.39], ['trending',
[['machine learning', 0.24], ['a', 0.15], ['area', 0.39], ['become', 0.39], ['has', 0.39], ['now', 0.39], ['research', 0.39], ['trending',
[['abstract data', 0.11], ['and', 0.14], ['type', 0.11], ['is', 0.09], ['behavior', 0.14], ['defined by', 0.23], ['a', 0.27], ['as', 0.23],
[['and', 0.14], ['abstract data_type', 0.11], ['behavior', 0.14], ['is_defined by', 0.23], ['a', 0.27], ['as', 0.23], ['be', 0.23], ['class
[['abstract data', 0.25], ['computer science', 0.31], ['in', 0.31], ['type', 0.25], ['is', 0.2], ['an', 0.31], ['a', 0.2], ['concept', 0.51
[['computer science', 0.32], ['in', 0.32], ['abstract data_type', 0.26], ['is', 0.2], ['an', 0.32], ['a', 0.2], ['concept', 0.52], ['theore
```

Maintenant nous proposons en nous inspirant de la fonction *MyCleanText* introduite dans le notebook "3 - Classification de données textuelles", une nouvelle fonction *MyCleanTextsforLDA* qui va prendre un ensemble de textes en entrée, les nettoyer, calculer les bigrams, déterminer le vocabulaire (dictionnaire) et retourner deux corpus (un avec uniquement un bag of words, l'autre en utilisant TF-IDF).

```
import re
import spacy
import gensim
import string
import nltk
from nltk.corpus import stopwords
from nltk.corpus import wordnet
import gensim
from gensim.utils import simple_preprocess
from gensim.models import Phrases
from gensim.models.phrases import Phraser
from gensim import corpora
from gensim import models
nltk.download('wordnet')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))
#nlp = spacy.load('en', disable=['parser', 'ner'])
nlp = spacy.load("en_core_web_sm",disable=['parser', 'ner'])

def MyCleanTextsforLDA(texts,
```

```
min_count=1, # nombre d'apparitions minimale pour un bigram
threshold=2,
no_below=1, # nombre minimum d'apparitions pour être dans le dictionnaire
no_above=0.5, # pourcentage maximal (sur la taille totale du corpus) pour filtrer
stop_words=stop_words
):

allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']
sentences=texts.copy()

# suppression des caractères spéciaux
sentences = [re.sub(r'[\w\s]', ' ', str(sentence)) for sentence in sentences]
# suppression de tous les caractères uniques
sentences = [re.sub(r'\s+[a-zA-Z]\s+', ' ', str(sentence)) for sentence in sentences]
# substitution des espaces multiples par un seul espace
sentences = [re.sub(r'\s+', ' ', str(sentence), flags=re.I) for sentence in sentences]

# conversion en minuscule et split des mots dans les textes
sentences = [sentence.lower().split() for sentence in sentences]

# utilisation de spacy pour ne retenir que les allowed_postags
texts_out = []
for sent in sentences:
    if len(sent) < (nlp.max_length): # si le texte est trop grand
        doc = nlp(" ".join(sent))
        texts_out.append(" ".join([token.lemma_ for token in doc if token.pos_ in allowed_postags]))
    else:
        texts_out.append(sent)
sentences=texts_out

# suppression des stopwords
words=[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in sentences]

# recherche des bigrammes
#bigram = Phrases(words, min_count, threshold,delimiter=b' ')
bigram = Phrases(words, min_count, threshold,delimiter=' ')
bigram_phraser = Phraser(bigram)

# sauvegarde des tokens et des bigrammes
bigram_token = []
for sent in words:
    bigram_token.append(bigram_phraser[sent])
```

```
# creation du vocabulaire
dictionary = gensim.corpora.Dictionary(bigram_token)

# il est possible de filtrer des mots en fonction de leur occurrence d'apparitions
#dictionary.filter_extremes(no_below, no_above)
# et de compacter le dictionnaire
# dictionary.compactify()
corpus = [dictionary.doc2bow(text) for text in bigram_token]

# recuperation du tfidf plutôt que uniquement le bag of words
tfidf = models.TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]

return corpus, corpus_tfidf, dictionary, bigram_token
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

▼ Construction du modèle LDA

A part le nombre de topics à spécifier, nous disposons, à présent, avec le corpus et le dictionnaire de tout ce qu'il faut pour construire un modèle LDA.

Gensim propose différentes classes pour obtenir le modèle : *LdaModel* ou *LdaMulticore*. Le second offre la possibilité de répartir des tâches sur différents CPU et est donc plus rapide.

Les principaux paramètres pour *LdaMulticore** sont les suivants :

1. *corpus* qui correspond au corpus représenté sous la forme d'une matrice (il s'agit du corpus que nous avons transformé précédemment).
2. *num_topics* est un entier qui correspond au nombre de topics à extraire du corpus d'apprentissage.
3. *id2word* correspond au dictionnaire ou vocabulaire de mots ou combinaisons de mots utilisés (il s'agit du dictionnaire que nous avons initialisé précédemment).
4. *chunksize* correspond au nombre de documents à utiliser à chaque étape d'entraînement du modèle.

5. *alpha* et *eta* qui sont des hyperparamètres qui considèrent la rareté des topics. Par défaut leur valeur est de 1.0.
6. *update_every* détermine la fréquence à laquelle les paramètres du modèle doivent être mis à jour.
7. *pas* correspond au nombre total de passes d'entraînement.

De plus amples informations sur les paramètres de `LdaModel` ou de `LdaMulticore` sont disponibles ici :

<https://radimrehurek.com/gensim/models/ldamodel.html> <https://radimrehurek.com/gensim/models/ldamulticore.html?highlight=multicore#module-gensim.models.ldamulticore>

* les paramètres indiqués sont similaires pour `LdaModel`.

Considérons le jeu de données des opinions imdb, amazon et yelp que nous avons vu dans les notebooks précédents :

```
!wget https://www.lirmm.fr/~poncelet/Ressources/ReviewsLabelled.csv
```

```
df_donnees = pd.read_csv("ReviewsLabelled.csv", names=['sentence', 'sentiment', 'source'],
                        header=0, sep='\t', encoding='utf8')

print (df_donnees.shape)
print (df_donnees.head())
```

```
(3000, 3)
```

	sentence	sentiment	source
0	So there is no way for me to plug it in here i...	0	amazon
1	Good case, Excellent value.	1	amazon
2	Great for the jawbone.	1	amazon
3	Tied to charger for conversations lasting more...	0	amazon
4	The mic is great.	1	amazon

Par la suite, nous ne nous intéressons qu'aux topics des avis de imdb.

```
# selection des données de imdb
df_source=df_donnees[(df_donnees['source']=='imdb')]
df_source.reset_index(drop = True, inplace = True)
print (df_source.shape)
```

```
(1000, 3)
```

Nous pré-traitions les données à l'aide de la fonction `MyCleanTextsforLDA`. Elle retourne le corpus transformé en BoW (*corpus*), transformé en TF-IDF (*corpus_tfidf*), le dictionnaire et l'ensemble des tokens et bigrams.

```
stop_words = stopwords.words('english')

# enrichissement des stopwords
stop_words.extend(['always', 'try', 'go', 'get', 'make', 'would', 'really',
                  'like', 'came', 'got'])
corpus, corpus_tfidf, dictionary, bigram_token=MyCleanTextsforLDA(df_source.sentence, stop_words=stop_words)
```

Dans un premier temps nous considérons le corpus de type bag of words :

```
import gensim
from gensim import models

num_topics=7 # nombre de topics
num_words=10 # nombre de mots par topics

lda_model_bow = gensim.models.ldamulticore.LdaMulticore(
    corpus=corpus,
    num_topics=num_topics,
    id2word=dictionary,
    chunksize=100,
    workers=7,
    passes=10,
    random_state=100,
    eval_every = 1,
    per_word_topics=True)

print ("Affichage des ", num_topics, " différents topics pour le corpus BOW :\n")
for idx, topic in lda_model_bow.print_topics(-1, num_words):
    print('Topic: {} Word: {}'.format(idx, topic))
```

Affichage des 7 différents topics pour le corpus BOW :

```
Topic: 0 Word: 0.016*"time" + 0.010*"movie" + 0.010*"character" + 0.008*"even" + 0.008*"see" + 0.006*"enjoy" + 0.005*"want" + 0.005*"thing"
Topic: 1 Word: 0.035*"movie" + 0.017*"bad" + 0.015*"watch" + 0.009*"film" + 0.007*"well" + 0.007*"awful" + 0.007*"even" + 0.006*"boring" +
Topic: 2 Word: 0.030*"movie" + 0.026*"film" + 0.010*"good" + 0.010*"see" + 0.007*"recommend" + 0.007*"drama" + 0.006*"look" + 0.006*"dialog
Topic: 3 Word: 0.016*"suck" + 0.014*"great" + 0.013*"movie" + 0.011*"way" + 0.010*"time" + 0.010*"film" + 0.009*"end" + 0.009*"bad" + 0.009
Topic: 4 Word: 0.014*"love" + 0.011*"good" + 0.009*"also" + 0.008*"piece" + 0.008*"film" + 0.006*"place" + 0.006*"movie" + 0.006*"performan
Topic: 5 Word: 0.025*"film" + 0.017*"movie" + 0.010*"look" + 0.010*"story" + 0.009*"script" + 0.007*"never" + 0.007*"use" + 0.007*"give" +
Topic: 6 Word: 0.029*"film" + 0.025*"bad" + 0.011*"well" + 0.009*"character" + 0.007*"story" + 0.007*"absolutely" + 0.006*"year" + 0.006*"t
```

▼ Evaluation de la qualité des topics extraits

La meilleure évaluation de la qualité des topics extraits reste ... l'évaluation humaine. Toutefois il existe deux mesures qui peuvent être utilisées :

1. La *perplexité* qui mesure la capacité d'un modèle probabiliste à généraliser, i.e. comment un modèle se comporte avec de nouvelles données qu'il n'a pas vu auparavant. Il est bien entendu préférable que la perplexité soit petite.
2. La *cohérence* qui évalue la cohérence entre les topics déduits du modèle en combinant un certain nombre de mesures. Il existe en effet de nombreuses mesures de cohérences. Par exemple, la mesure C_v est basée sur une fenêtre glissante, une segmentation en un seul ensemble des premiers mots et une mesure de confirmation indirecte qui utilise des informations mutuelles ponctuelles normalisées (NPMI) et la similitude cosinus. Il est bien entendu préférable d'avoir une forte valeur.

```
print ("Calcul de la cohérence et de la perplexité du modèle pour le corpus BOW : ")
# cohérence
coherence_model_lda = CoherenceModel(model=lda_model_bow, texts=bigram_token, dictionary=dictionary,
                                     coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCohérence : ', coherence_lda)

# perplexité
print('\nPerplexité : ', lda_model_bow.log_perplexity(corpus))
```

Calcul de la cohérence et de la perplexité du modèle pour le corpus BOW :

Cohérence : 0.5025704603798004

Perplexité : -8.186905040175743

Exercice : précédemment nous avons extrait la matrice du corpus obtenue via un TF-IDF, i.e. *corpus_tfidf*. Rechercher le modèle LDA associé à ce corpus, afficher les différents topics et évaluer la perplexité et la cohérence. Vous pouvez faire varier le nombre de topics et le nombre de mots affichés (*num_topics*, Comparer avec les résultats obtenus sans appliquer de TF-IDF. Est-ce que les topics vous semblent mieux décrire les documents ?

Solution :

```
#il suffit d'appliquer LdaMulticore sur corpus_tfidf.
lda_model_tfidf = gensim.models.ldamulticore.LdaMulticore(
    corpus=corpus_tfidf,
    num_topics=num_topics,
    id2word=dictionary,
    chunksize=100,
    workers=7,
    passes=10,
    random_state=100,
    eval_every = 1,
    per_word_topics=True)

print ("Affichage des ",num_topics," différents topics pour le corpus TF-IDF :")

for idx, topic in lda_model_tfidf.print_topics(-1,num_words):
    print('Topic: {} Word: {}'.format(idx, topic))

coherence_model_lda_tfidf = CoherenceModel(model=lda_model_tfidf, texts=bigram_token, dictionary=dictionary,
    coherence='c_v')
coherence_lda = coherence_model_lda_tfidf.get_coherence()
print('Cohérence : ', coherence_lda)

print('Perplexité : ', lda_model_tfidf.log_perplexity(corpus_tfidf))

Affichage des 7 différents topics pour le corpus TF-IDF :
Topic: 0 Word: 0.007*"time" + 0.007*"enjoy" + 0.006*"watch" + 0.005*"thing" + 0.005*"easy" + 0.005*"bore" + 0.005*"see" + 0.004*"whole" + 0
Topic: 1 Word: 0.013*"movie" + 0.010*"awful" + 0.006*"right" + 0.005*"dialogue" + 0.005*"excellent" + 0.004*"bad" + 0.004*"film" + 0.004*"s
Topic: 2 Word: 0.010*"recommend" + 0.008*"see" + 0.007*"movie" + 0.006*"good" + 0.006*"see film" + 0.006*"film" + 0.005*"drama" + 0.005*"se
Topic: 3 Word: 0.010*"great" + 0.008*"movie" + 0.008*"suck" + 0.006*"way" + 0.005*"actress" + 0.005*"bad" + 0.004*"show" + 0.004*"scene" +
Topic: 4 Word: 0.008*"avoid cost" + 0.006*"love" + 0.006*"rent" + 0.006*"film" + 0.006*"beautiful" + 0.005*"slow" + 0.005*"truly" + 0.004*"
Topic: 5 Word: 0.008*"funny" + 0.007*"look" + 0.007*"give" + 0.006*"story" + 0.006*"film" + 0.006*"movie" + 0.005*"waste time" + 0.005*"act
```


Topic: 6 Word: 0.018*"bad" + 0.009*"film" + 0.005*"hate" + 0.005*"absolutely" + 0.004*"plot" + 0.004*"rate movie" + 0.004*"character" + 0.0
 Cohérence : 0.6324022311239281
 Perplexité : -10.01927277965965

▼ Quel est le topic dominant ?

Maintenant que nous savons calculer la cohérence et la perplexité du modèle, il est intéressant de pouvoir déterminer pour un document quel est le topic qui est le plus dominant. En fait l'application du modèle appris sur un document retourne un vecteur contenant pour chacun des topics le pourcentage d'appartenance de ce topic dans le document :

```
def dominant_topic(model,corpus,num_topics):
    # recuperation du vecteur associé
    # creation d'un dictionnaire pour stocker les résultats
    topic_dictionary = {i: [] for i in range(num_topics)}
    topic_probability_scores = model.get_document_topics(corpus,minimum_probability=0.000)
    if len(topic_probability_scores) ==1 : # il y a plusieurs predictions on recupere la premiere
        row=topic_probability_scores[0]
    else: # on concatene les predictions
        tab=[]
        for j in range (len(topic_probability_scores)):
            tab.append(topic_probability_scores[j])
        row=tab
    # parcours des différents topics
    for (topic_num, prop_topic) in row:
        topic_dictionary[topic_num].append(prop_topic)
    # tri pour avoir le plus grand en premier
    list_proba=topic_dictionary
    topic_dictionary=sorted(topic_dictionary, key=topic_dictionary.get,reverse = True)
    return topic_dictionary, list_proba

print ("Le premier document pré-traité : ",bigram_token[0])

# recuperation du topic dominant
topic_dictionary,list_proba=dominant_topic(lda_model_bow,corpus[0],num_topics)

print ("le topic dominant pour le model BOW est : ",topic_dictionary[0])
print ("la liste des topics par ordre d'importance : ",topic_dictionary)
print ("les probabilités associées : ",list_proba)
```

```
topic_dictionary,list_proba=dominant_topic(lda_model_tfidf,corpus[0],num_topics)
print ("le topic dominant pour le model TF-IDF est : ",topic_dictionary[0])
print ("la liste des topics par ordre d'importance : ",topic_dictionary)
print ("les probabilités associées : ",list_proba)
```

```
Le premier document pré-traité : ['slow move', 'aimless', 'movie', 'distressed', 'drift', 'young', 'man']
le topic dominant pour le model BOW est : 2
la liste des topics par ordre d'importance : [2, 0, 5, 1, 6, 3, 4]
les probabilités associées : {0: [0.017953046], 1: [0.017878884], 2: [0.8926046], 3: [0.017870558], 4: [0.017865907], 5: [0.017949773], 6: [0.017865907]}
le topic dominant pour le model TF-IDF est : 3
la liste des topics par ordre d'importance : [3, 5, 2, 1, 6, 0, 4]
les probabilités associées : {0: [0.01787138], 1: [0.017885141], 2: [0.01789161], 3: [0.8926333], 4: [0.017868709], 5: [0.017974827], 6: [0.017865907]}
```

La fonction *format_topics_sentences* permet de créer à partir du corpus de document un dataframe qui contient à la fois le topic dominant, i.e. celui qui représente le mieux le document, la liste des mots associés au topic ainsi que sa contribution.

```
def format_topics_sentences(ldamodel, corpus, texts):
    # Initialisation du dataframe de sortie
    sent_topics_df = pd.DataFrame()

    # Recherche le topic dominant pour chaque document
    for i, row_list in enumerate(ldamodel[corpus]):
        row = row_list[0] if ldamodel.per_word_topics else row_list

        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Donne le topic dominant, le pourcentage de contribution
        # et les mots clés pour chaque document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0: # => topic dominant
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num), round(prop_topic,4), topic_keywords]), ignore_index=True)
            else:
                break
        sent_topics_df.columns = ['topic_dominant', 'pourcentage_contrib', 'topic_keywords']

    # Ajout du texte original à la fin de la sortie
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)
```

```
print ("\nLes différents topics pour le modèle BOW : ")
for i,topic in lda_model_bow.show_topics(formatted=True, num_topics=num_topics, num_words=num_words):
    print(str(i)+": " + topic)

df_topic_sents_keywords = format_topics_sentences(ldamodel=lda_model_bow, corpus=corpus, texts=df_source.sentence)

display(df_topic_sents_keywords)
```

Les différents topics pour le modèle BOW :

0: 0.016*"time" + 0.010*"movie" + 0.010*"character" + 0.008*"even" + 0.008*"see" + 0.006*"enjoy" + 0.005*"want" + 0.005*"thing" + 0.005*"wa
 1: 0.035*"movie" + 0.017*"bad" + 0.015*"watch" + 0.009*"film" + 0.007*"well" + 0.007*"awful" + 0.007*"even" + 0.006*"boring" + 0.006*"writi
 2: 0.030*"movie" + 0.026*"film" + 0.010*"good" + 0.010*"see" + 0.007*"recommend" + 0.007*"drama" + 0.006*"look" + 0.006*"dialogue" + 0.006*
 3: 0.016*"suck" + 0.014*"great" + 0.013*"movie" + 0.011*"way" + 0.010*"time" + 0.010*"film" + 0.009*"end" + 0.009*"bad" + 0.009*"character"
 4: 0.014*"love" + 0.011*"good" + 0.009*"also" + 0.008*"piece" + 0.008*"film" + 0.006*"place" + 0.006*"movie" + 0.006*"performance" + 0.005*
 5: 0.025*"film" + 0.017*"movie" + 0.010*"look" + 0.010*"story" + 0.009*"script" + 0.007*"never" + 0.007*"use" + 0.007*"give" + 0.007*"also"
 6: 0.029*"film" + 0.025*"bad" + 0.011*"well" + 0.009*"character" + 0.007*"story" + 0.007*"absolutely" + 0.006*"year" + 0.006*"think" + 0.00

	topic_dominant	pourcentage_contrib	topic_keywords	sentence
0	2	0.8926	movie, film, good, see, recommend, drama, look...	A very, very, very slow-moving, aimless movie ...
1	2	0.9044	movie, film, good, see, recommend, drama, look...	Not sure who was more lost - the flat characte...
2	1	0.9463	movie, bad, watch, film, well, awful, even, bo...	Attempting artiness with black & white and cle...
3	4	0.7850	love, good, also, piece, film, place, movie, p...	Very little music or anything to speak of.
4	5	0.4883	film, movie, look, story, script, never, use, ...	The best scene in the movie was when Gerardo i...
...
995	3	0.8773	suck, great, movie, way, time, film, end, bad,...	I just got bored watching Jessica Lange take h...
996	4	0.9045	love, good, also, piece, film, place, movie, p...	Unfortunately, any virtue in this film's produ...
997	1	0.7141	movie, bad, watch, film, well, awful, even, bo...	In a word, it is embarrassing.
998	1	0.7140	movie, bad, watch, film, well, awful, even, bo...	Exceptionally bad!
999	3	0.7856	suck, great, movie, way, time, film, end, bad,...	All in all its an insult to one's intelligence...

1000 rows x 4 columns

A partir de ce dataframe il est possible de connaître le document le plus représentatif du topic :

```
# groupement par topic dominant
sent_topics_gbt = df_topic_sents_keywords.groupby('topic_dominant')

sentence_for_topics=pd.DataFrame()
# concaténation en fonction des pourcentage de contribution
for i, grp in sent_topics_gbt:
    sentence_for_topics = pd.concat([sentence_for_topics,
                                     grp.sort_values(['pourcentage_contrib'], ascending=[0]).head(1)],
                                     axis=0)

sentence_for_topics.reset_index(drop=True, inplace=True)
sentence_for_topics.columns = ['topic_num', "pourcentage_contrib", "topic_keywords", "text"]

display(sentence_for_topics)
```

	topic_num	pourcentage_contrib	topic_keywords	text
0	0	0.9590	time, movie, character, even, see, enjoy, want...	Elias Koteas,Jack Palance play good roles Ange...
1	1	0.9609	movie, bad, watch, film, well, awful, even, bo...	About 30 minutes of footage is wasted to show ...
2	2	0.9627	movie, film, good, see, recommend, drama, look...	A cheap and cheerless heist movie with poor ch...
3	3	0.9591	suck, great, movie, way, time, film, end, bad,...	The acting from all involved and that includes...
4	4	0.9785	love, good, also, piece, film, place, movie, p...	This is a masterful piece of film-making, with...
5	5	0.9627	film, movie, look, story, script, never, use, ...	The film deserves strong kudos for taking this...
6	6	0.9657	film, bad, well, character, story, absolutely,...	The attempts at humor were pitiful and story i...

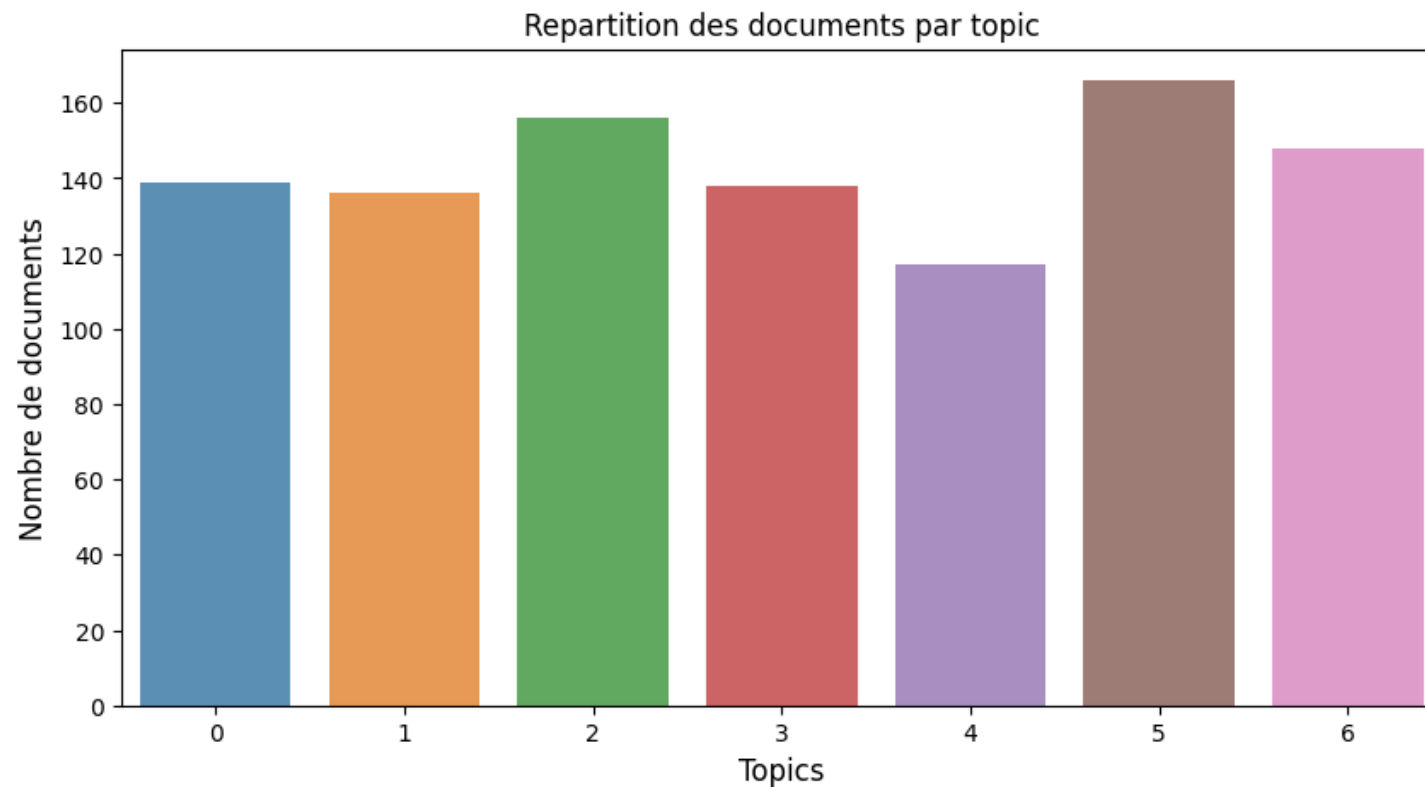


ou bien la répartition des documents dans les topics :

```
# nombre de documents par topics
topic_counts = df_topic_sents_keywords['topic_dominant'].value_counts()

# affichage
plt.figure(figsize=(10,5))
ax=sns.barplot(x=topic_counts.index,y=topic_counts.values, alpha=0.8)
plt.title('Repartition des documents par topic')
plt.ylabel('Nombre de documents', fontsize=12)
```

```
plt.xlabel('Topics', fontsize=12)  
plt.show()
```



▼ Prediction des topics pour des nouveaux documents

Il est tout à fait possible de faire de la prédiction de topics appris pour un nouveau document. Pour cela il est indispensable de d'abord le transformer en vecteur qui puisse être utilisé par le modèle. En outre, si des pré-traitements ont été effectués, ces derniers doivent également être réalisés.

Le principe général est tout simplement d'appliquer le modèle sur le document qui a été transformé. Cette action retourne un vecteur qui contient pour chaque topic sa probabilité de participation.

```

mytext = ["This is a new text that I have never seen before "]

print ("transformation du texte :", mytext)

# application des pré-traitements pour appliquer les pré-traitements et transformer le texte en vecteur
my_corpus, my_corpus_tfidf, my_dictionary, my_bigram_token=MyCleanTextsforLDA(mytext)

print ("la matrice obtenue après transformation : ", my_corpus)
# il suffit alors d'appliquer le modèle appris sur le texte transformé
# ici nous considérons la matrice (my_corpus) et le modèle précédent : lda_model_bow
vector=lda_model_bow[my_corpus]

print ("les probabilités pour chaque topic : ", vector[0])

# il est possible de déterminer le topic dominant à l'aide la fonction dominant_topic

topic_dictionary,list_proba=dominant_topic(lda_model_bow,my_corpus,num_topics)

print ("le topic dominant pour le model BOW sur un nouveau texte : ",topic_dictionary[0])
print ("la liste des topics par ordre d'importance : ",topic_dictionary)
print ("les probabilités associées : ",list_proba)

```

```

transformation du texte : ['This is a new text that I have never seen before ']
la matrice obtenue après transformation : [[(0, 1), (1, 1), (2, 1), (3, 1)]]
les probabilités pour chaque topic : ((0, 0.02858948), (1, 0.028584832), (2, 0.8283572), (3, 0.028587334), (4, 0.02858257), (5, 0.0287160
le topic dominant pour le model BOW sur un nouveau texte : 2
la liste des topics par ordre d'importance : [2, 5, 0, 3, 1, 4, 6]
les probabilités associées : {0: [0.028589478], 1: [0.028584829], 2: [0.8283626], 3: [0.028587328], 4: [0.028582567], 5: [0.028710693], 6

```

▼ Sauvegarde d'un modèle

Il est bien entendu possible de sauvegarder un modèle pour pouvoir le réutiliser. Par contre il est peut être aussi utile de sauvegarder le corpus transformé ainsi que les dictionnaires.

```

# pour sauvegarder le modèle
lda_model_bow.save('firstlda_model_bow.model')

# pour charger le modèle
lda_new = gensim.models.LdaMulticore.load('firstlda_model_bow.model')

# test sur un texte non lu voir section précédente

```

```

# c'est sur un texte non lu, voir section précédente
mytext = ["This is another new text that I have never seen before even if I have already seen new texts"]
my_corpus, my_corpus_tfidf, my_dictionary, my_bigram_token=MyCleanTextsforLDA(mytext)

topic_dictionary, list_proba=dominant_topic(lda_new, my_corpus, num_topics)

print ("le topic dominant à partir du modèle sauvegardé sur un nouveau texte : ", topic_dictionary[0])
print ("la liste des topics par ordre d'importance : ", topic_dictionary)
print ("les probabilités associées : ", list_proba)

le topic dominant à partir du modèle sauvegardé sur un nouveau texte : 2
la liste des topics par ordre d'importance : [2, 5, 1, 3, 0, 4, 6]
les probabilités associées : {0: [0.017874178], 1: [0.01789168], 2: [0.89271086], 3: [0.017874913], 4: [0.017865555], 5: [0.0179211], 6:

```

▼ Recherche automatique des meilleurs paramètres

Il est bien évident possible de trouver les meilleurs paramètres, notamment alpha, beta et le nombre de topics afin d'obtenir le meilleur modèle. Dans ce notebook nous avons utilisé la librairie gensim et notamment la classe LdaMulticore qui est plus performante que celles proposées par scikit learn (LatentDirichletAllocation). Il est donc assez compliqué d'appliquer GridSearchCV. Néanmoins nous pouvons faire un traitement assez similaire.

La cellule ci-dessous propose la fonction MyGridSearchLda. Cette dernière est largement inspirée de

<https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0>

Elle peut être bien entendue améliorée en intégrant par exemple d'autres paramètres, différents découpages du jeu de données et évaluation d'un jeu de test/apprentissage différent dans une approche "à la cross validation". Elle a ici plus un but pédagogique pour rapidement tester les paramètres.

```

# ce code est inspiré de
# https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0
def compute_coherence_values(corpus, dictionary, listtokens, k, alpha, eta):

    lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                             id2word=dictionary,
                                             num_topics=k,
                                             random_state=100,
                                             chunksize=100,
                                             passes=10,
                                             alpha=alpha,
                                             eta=eta,
                                             per_word_topics=True)

```

```

coherence_model_lda = CoherenceModel(model=lda_model, texts=listtokens, dictionary=dictionary, coherence='c_v')

return coherence_model_lda.get_coherence()

def MyGridSearchLda (corpus,listtokens,dictionnary,nb_topics,alpha,beta,verbose=1):

    grid = {}
    model_results = {'topics': [],
                     'alpha': [],
                     'eta': [],
                     'coherence': []
                    }
    # iteration sur le nombre de topics
    for k in nb_topics:
        # iteration sur les valeurs d'alpha
        for a in alpha:
            # iteration sur les valeurs de eta
            for e in eta:
                # calcul du score de coherence
                cv = compute_coherence_values(corpus=corpus,
                                              dictionary=dictionary,
                                              listtokens=listtokens,
                                              k=k, alpha=a, eta=e)

                if verbose==1:
                    print ('topics:', k, ' alpha: %0.3f  eta: %0.3f  coherence: %0.3f'%(a,e,cv))

                # sauvegarde des résultats
                model_results['topics'].append(k)
                model_results['alpha'].append(a)
                model_results['eta'].append(e)
                model_results['coherence'].append(cv)

    df_result=pd.DataFrame(model_results)
    df_result = df_result.sort_values('coherence',ascending=False)
    df_result.reset_index(drop=True, inplace=True)
    return df_result

# variation du nombre de topics
min_topics = 7
max_topics = 10

```





```
step_size = 1
nb_topics = range(min_topics, max_topics, step_size)
# variation d'alpha
alpha = list(np.arange(0.01, 1, 0.3))
# variation de eta
eta = list(np.arange(0.01, 1, 0.3))

print ("Extraction des meilleurs paramètres pour le corpus via BOW : ")
df_result=MyGridSearchLda (corpus,bigram_token,dictionary,nb_topics,alpha,eta,verbose=1)
display (df_result)
```

Extraction des meilleurs paramètres pour le corpus via BOW :

topics: 7	alpha: 0.010	eta: 0.010	coherence: 0.547
topics: 7	alpha: 0.010	eta: 0.310	coherence: 0.511
topics: 7	alpha: 0.010	eta: 0.610	coherence: 0.515
topics: 7	alpha: 0.010	eta: 0.910	coherence: 0.495
topics: 7	alpha: 0.310	eta: 0.010	coherence: 0.613
topics: 7	alpha: 0.310	eta: 0.310	coherence: 0.574
topics: 7	alpha: 0.310	eta: 0.610	coherence: 0.560
topics: 7	alpha: 0.310	eta: 0.910	coherence: 0.543
topics: 7	alpha: 0.610	eta: 0.010	coherence: 0.608
topics: 7	alpha: 0.610	eta: 0.310	coherence: 0.582
topics: 7	alpha: 0.610	eta: 0.610	coherence: 0.545
topics: 7	alpha: 0.610	eta: 0.910	coherence: 0.531
topics: 7	alpha: 0.910	eta: 0.010	coherence: 0.596
topics: 7	alpha: 0.910	eta: 0.310	coherence: 0.571
topics: 7	alpha: 0.910	eta: 0.610	coherence: 0.532
topics: 7	alpha: 0.910	eta: 0.910	coherence: 0.506
topics: 8	alpha: 0.010	eta: 0.010	coherence: 0.566
topics: 8	alpha: 0.010	eta: 0.310	coherence: 0.517
topics: 8	alpha: 0.010	eta: 0.610	coherence: 0.527
topics: 8	alpha: 0.010	eta: 0.910	coherence: 0.550
topics: 8	alpha: 0.310	eta: 0.010	coherence: 0.606
topics: 8	alpha: 0.310	eta: 0.310	coherence: 0.571
topics: 8	alpha: 0.310	eta: 0.610	coherence: 0.558
topics: 8	alpha: 0.310	eta: 0.910	coherence: 0.557
topics: 8	alpha: 0.610	eta: 0.010	coherence: 0.597
topics: 8	alpha: 0.610	eta: 0.310	coherence: 0.571
topics: 8	alpha: 0.610	eta: 0.610	coherence: 0.557
topics: 8	alpha: 0.610	eta: 0.910	coherence: 0.535
topics: 8	alpha: 0.910	eta: 0.010	coherence: 0.607
topics: 8	alpha: 0.910	eta: 0.310	coherence: 0.566
topics: 8	alpha: 0.910	eta: 0.610	coherence: 0.529
topics: 8	alpha: 0.910	eta: 0.910	coherence: 0.516
topics: 9	alpha: 0.010	eta: 0.010	coherence: 0.582
topics: 9	alpha: 0.010	eta: 0.310	coherence: 0.519
topics: 9	alpha: 0.010	eta: 0.610	coherence: 0.507
topics: 9	alpha: 0.010	eta: 0.910	coherence: 0.535
topics: 9	alpha: 0.310	eta: 0.010	coherence: 0.634
topics: 9	alpha: 0.310	eta: 0.310	coherence: 0.585
topics: 9	alpha: 0.310	eta: 0.610	coherence: 0.561
topics: 9	alpha: 0.310	eta: 0.910	coherence: 0.548
topics: 9	alpha: 0.610	eta: 0.010	coherence: 0.624
topics: 9	alpha: 0.610	eta: 0.310	coherence: 0.576
topics: 9	alpha: 0.610	eta: 0.610	coherence: 0.566
topics: 9	alpha: 0.610	eta: 0.910	coherence: 0.497
topics: 9	alpha: 0.910	eta: 0.010	coherence: 0.642
topics: 9	alpha: 0.910	eta: 0.310	coherence: 0.562
topics: 9	alpha: 0.910	eta: 0.610	coherence: 0.496

topics: 9 alpha: 0.910 eta: 0.910 coherence: 0.547

	topics	alpha	eta	coherence	
0	9	0.91	0.01	0.642312	
1	9	0.31	0.01	0.633976	
2	9	0.61	0.01	0.624112	
3	7	0.31	0.01	0.613357	
4	7	0.61	0.01	0.608018	
5	8	0.91	0.01	0.607396	
6	8	0.31	0.01	0.606186	
7	8	0.61	0.01	0.597428	
8	7	0.91	0.01	0.595863	
9	9	0.31	0.31	0.585334	
10	7	0.61	0.31	0.582256	
11	9	0.01	0.01	0.581793	
12	9	0.61	0.31	0.576327	
13	7	0.31	0.31	0.574379	
14	7	0.91	0.31	0.571414	
15	8	0.31	0.31	0.571378	
16	8	0.61	0.31	0.570866	
17	8	0.01	0.01	0.566345	
18	8	0.91	0.31	0.566219	
19	9	0.61	0.61	0.566179	
20	9	0.91	0.31	0.562497	
21	9	0.31	0.61	0.560681	
22	7	0.31	0.61	0.559780	
23	8	0.31	0.61	0.558379	
24	8	0.31	0.91	0.556944	
25	8	0.61	0.61	0.556595	

26	8	0.01	0.91	0.549724
27	9	0.31	0.91	0.548462
28	7	0.01	0.01	0.546937

Exercice : à partir des résultats précédents, appliquer les meilleurs paramètres pour le modèle LDA sur le corpus obtenu via BOW afin d'obtenir les topics et les mots associés. Comparer les résultats avec l'exécution précédente.

31	7	0.31	0.91	0.543310

Solution :

```
# les résultats sont actuellement stockés dans le dataframe df_result.
# Il suffit de récupérer le nombre de topics, alpha et eta à partir de ce dernier et de relancer LdaMulticore.
num_topic_best=df_result['topics'][0]
alpha_best=df_result['alpha'][0]
eta_best=df_result['eta'][0]

lda_model_bow_best = gensim.models.ldamulticore.LdaMulticore(
    corpus=corpus,
    num_topics=num_topic_best,
    alpha=alpha_best,
    eta=eta_best,
    id2word=dictionary,
    chunksize=100,
    workers=7, # Nombre de coeurs - 1
    passes=10,
    random_state=100,
    eval_every = 1,
    per_word_topics=True)

print ("Affichage des ",num_topic_best," différents topics pour le corpus BOW :")
for idx, topic in lda_model_bow_best.print_topics(-1,num_words):
    print('Topic: {} Word: {}'.format(idx, topic))

# cohérence
coherence_model_lda = CoherenceModel(model=lda_model_bow_best, texts=bigram_token, dictionary=dictionary,
                                     coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('Cohérence : ', coherence_lda)
```

```
# perplexité
print('Perplexité : ', lda_model_bow_best.log_perplexity(corpus))

Affichage des 9 différents topics pour le corpus BOW :
Topic: 0 Word: 0.034*"time" + 0.023*"even" + 0.017*"recommend" + 0.017*"much" + 0.016*"character" + 0.014*"thing" + 0.014*"one" + 0.014*"en
Topic: 1 Word: 0.078*"movie" + 0.023*"watch" + 0.021*"bad" + 0.017*"right" + 0.016*"good" + 0.015*"find" + 0.013*"people" + 0.012*"game" +
Topic: 2 Word: 0.033*"movie" + 0.029*"see" + 0.026*"look" + 0.024*"film" + 0.013*"many" + 0.012*"boring" + 0.012*"play" + 0.011*"think" + 0
Topic: 3 Word: 0.029*"story" + 0.019*"great" + 0.019*"know" + 0.018*"end" + 0.016*"ever" + 0.016*"still" + 0.015*"character" + 0.015*"film"
Topic: 4 Word: 0.021*"good" + 0.019*"show" + 0.017*"also" + 0.017*"acting" + 0.014*"movie" + 0.011*"role" + 0.011*"script" + 0.010*"put" +
Topic: 5 Word: 0.053*"film" + 0.026*"way" + 0.025*"movie" + 0.017*"use" + 0.014*"never" + 0.012*"script" + 0.011*"mostly" + 0.011*"man" + 0
Topic: 6 Word: 0.061*"film" + 0.020*"bad" + 0.018*"well" + 0.018*"funny" + 0.016*"awful" + 0.011*"see movie" + 0.010*"portrayal" + 0.010*"t
Topic: 7 Word: 0.032*"movie" + 0.018*"well" + 0.015*"plot" + 0.014*"drama" + 0.012*"bad" + 0.012*"fact" + 0.011*"child" + 0.011*"year" + 0
Topic: 8 Word: 0.027*"film" + 0.025*"character" + 0.025*"suck" + 0.019*"actor" + 0.016*"bad" + 0.013*"well" + 0.013*"dialogue" + 0.012*"lov
Cohérence : 0.5437271986618121
Perplexité : -14.193060935913266
```

La fonction suivante illustre comment obtenir également le meilleur nombre de topics. Elle est assez simple car applique sur un intervalle de valeur le modèle et calcule la cohérence associée. Elle peut bien entendu être étendue en intégrant une variation de paramètres différents comme nous l'avons vu précédemment. Par contre ce traitement est forcément long (Cf. l'application de GridSearchCV).

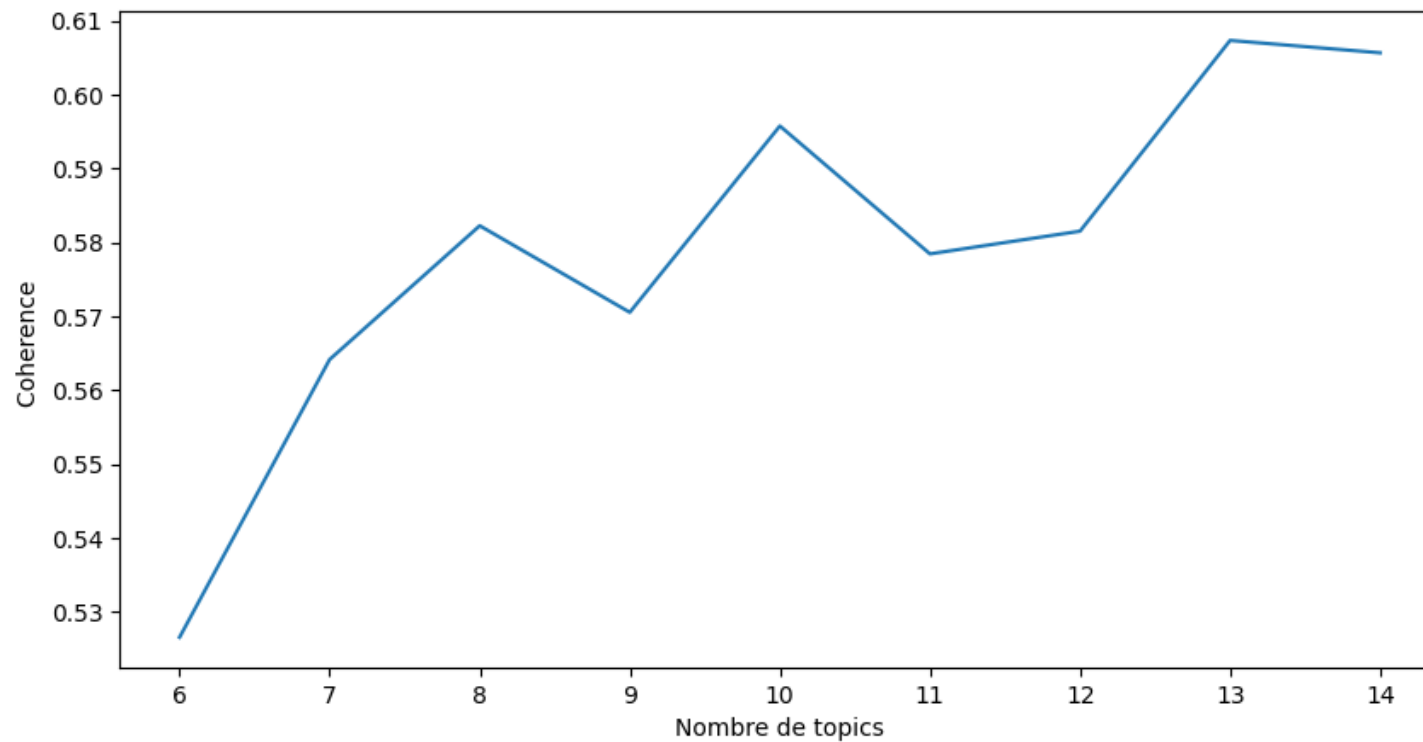
```
def get_best_coherence_values(corpus, dictionary, listtokens, start=5, stop=15, step=2):
    coherence_values = []
    model_list = []
    for num_topics in range(start, stop, step):
        lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                                id2word=dictionary,
                                                num_topics=num_topics,
                                                random_state=100,
                                                chunksize=100,
                                                passes=10,
                                                per_word_topics=True)

        coherence_model_lda = CoherenceModel(model=lda_model, texts=listtokens, dictionary=dictionary, coherence='c_v')
        model_list.append(lda_model)
        coherence_values.append(coherence_model_lda.get_coherence())
    return model_list, coherence_values
```

```
# test sur un intervalle de 6 à 15 en utilisant le corpus Bow
start=6
```

```
stop=15
step=1
model_list, coherence_values = get_best_coherence_values(dictionary=dictionary,
                                                         corpus=corpus,
                                                         listtokens=bigram_token,
                                                         start=start, stop=stop, step=step)
```

```
# affichage du graphe associé à la recherche du nombre de topics
plt.figure(figsize=(10,5))
x = range(start, stop, step)
plt.plot(x, coherence_values)
plt.xlabel("Nombre de topics")
plt.ylabel("Coherence ")
#plt.legend(("Valeurs de cohérencescoherence_values"), loc='best')
plt.show()
```



▾ Visualisation interactive des topics

La librairie pyLDavis propose de pouvoir visualiser de manière interactive les différents topics obtenus. Il est possible de les afficher dans le notebook ou bien de sauvegarder une page html.

Plus d'informations sur la librairie pyLDavis ici : <https://pyldavis.readthedocs.io/en/latest/readme.html>

Il suffit dans un premier temps de préparer les données à l'aide de gensimvis.

```
!pip install pyLDavis==2.1.2
```

```
Requirement already satisfied: pyLDavis==2.1.2 in /usr/local/lib/python3.10/dist-packages (2.1.2)
Requirement already satisfied: wheel>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (0.41.2)
Requirement already satisfied: numpy>=1.9.2 in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (1.23.5)
Requirement already satisfied: scipy>=0.18.0 in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (1.10.1)
Requirement already satisfied: pandas>=0.17.0 in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (1.5.3)
Requirement already satisfied: joblib>=0.8.4 in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (1.3.2)
Requirement already satisfied: jinja2>=2.7.2 in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (3.1.2)
Requirement already satisfied: numexpr in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (2.8.5)
Requirement already satisfied: pytest in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (7.4.1)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (0.18.3)
Requirement already satisfied: funcy in /usr/local/lib/python3.10/dist-packages (from pyLDavis==2.1.2) (2.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2>=2.7.2->pyLDavis==2.1.2) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.17.0->pyLDavis==2.1.2) (2.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.17.0->pyLDavis==2.1.2) (2023.3.post1)
Requirement already satisfied: iniconfig in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDavis==2.1.2) (2.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDavis==2.1.2) (23.1)
Requirement already satisfied: pluggy<2.0,>=0.12 in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDavis==2.1.2) (1.3.0)
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDavis==2.1.2) (1.1.3)
Requirement already satisfied: tomli>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from pytest->pyLDavis==2.1.2) (2.0.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>=0.17.0->pyLDavis==
```

```
import pyLDavis
#import pyLDavis.gensim_models

import pyLDavis.gensim as gensimvis
pyLDavis.enable_notebook()
#import pyLDavis
```

```
visu_data = gensimvis.prepare(lda_model_bow, corpus, dictionary)
#visu_data = pyLDAvis.gensim_models.prepare(lda_model_bow, corpus, dictionary)
```

Pour afficher les données et les sauvegarder sur une page html :

```
# save the results to html file
filename='my_first_lda.html'
my_lda = open(filename, 'w')
pyLDAvis.save_html(visu_data,my_lda)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au
and should_run_async(code)
```

Exercice : aller sur le répertoire courant et ouvrir dans un navigateur le fichier my_first_lda.html.

Pour afficher le résultat directement dans le notebook :

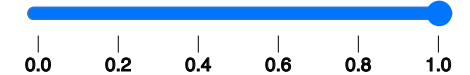
```
pyLDAvis.display(visu_data)
```


/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au and should_run_async(code)

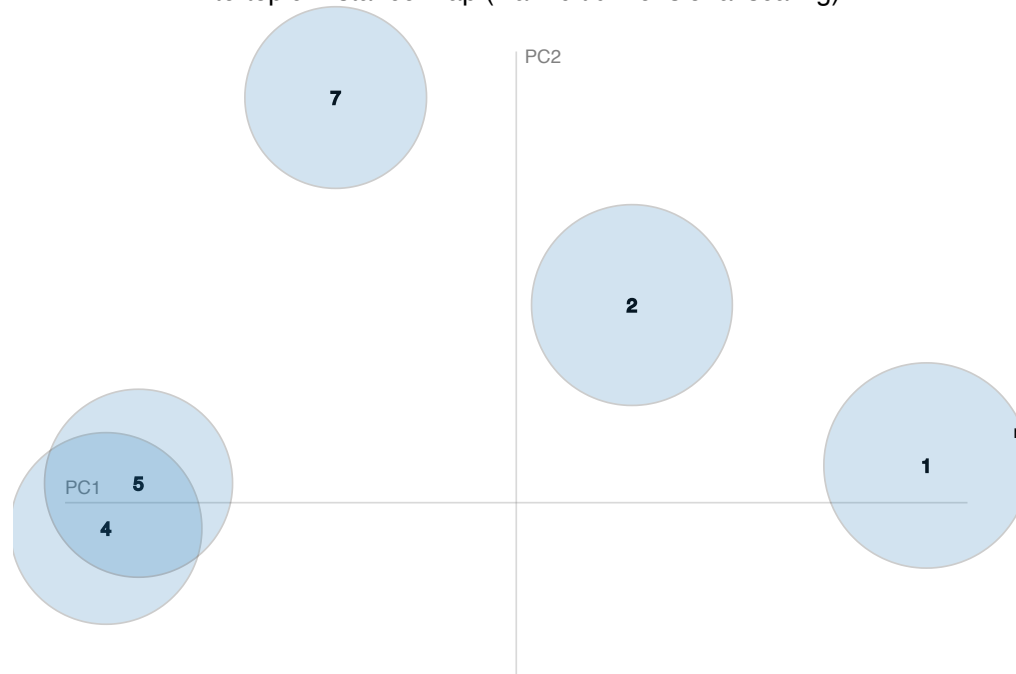
Selected Topic:

Slide to adjust relevance metric:(2)

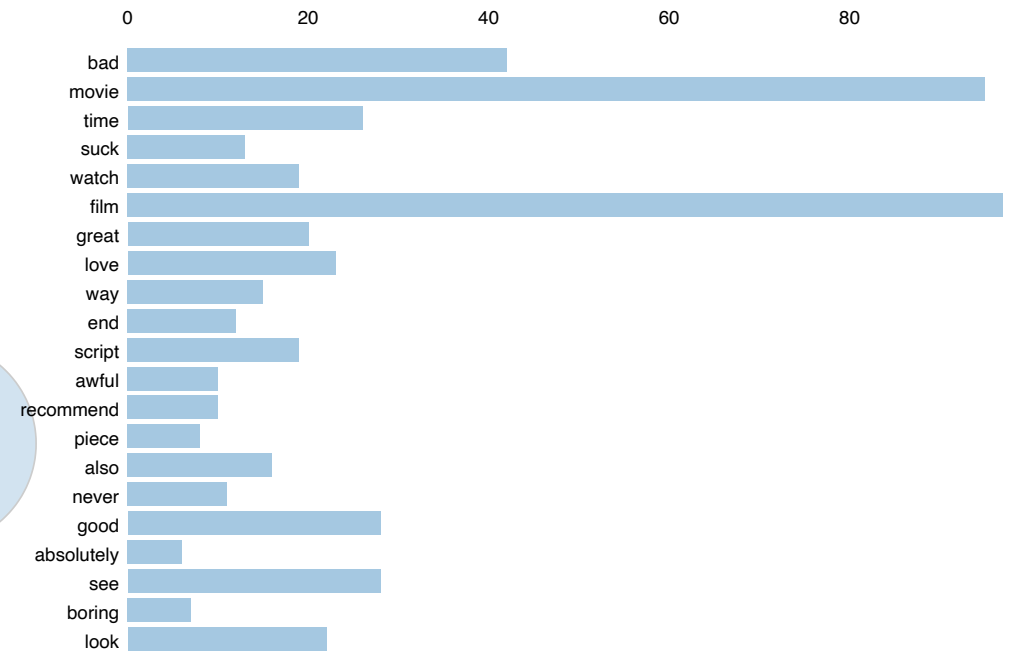
$\lambda = 1$



Intertopic Distance Map (via multidimensional scaling)



Top-30 Most Salient Terms⁽¹⁾



Exercice : nous avons réalisé plusieurs expérimentations notamment en utilisant bag of words ou tf-idf. Afficher les deux corpus pour pouvoir les comparer à l'aide de la visualisation de pyLDAvis.

Solution :

```
# il suffit de préparer les données sur les deux modèles et de les visualiser ou les sauvegarder dans des fichiers html.
# Attention si l'affichage se fait dans le notebook, il faut les mettre dans deux cellules différentes.
visu_data_bow = gensimvis.prepare(lda_model_bow, corpus, dictionary)
visu_data_tfidf=gensimvis.prepare(lda_model_tfidf,corpus_tfidf,dictionary)
pyLDAvis.display(visu_data_bow)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au
and should_run_async(code)
```

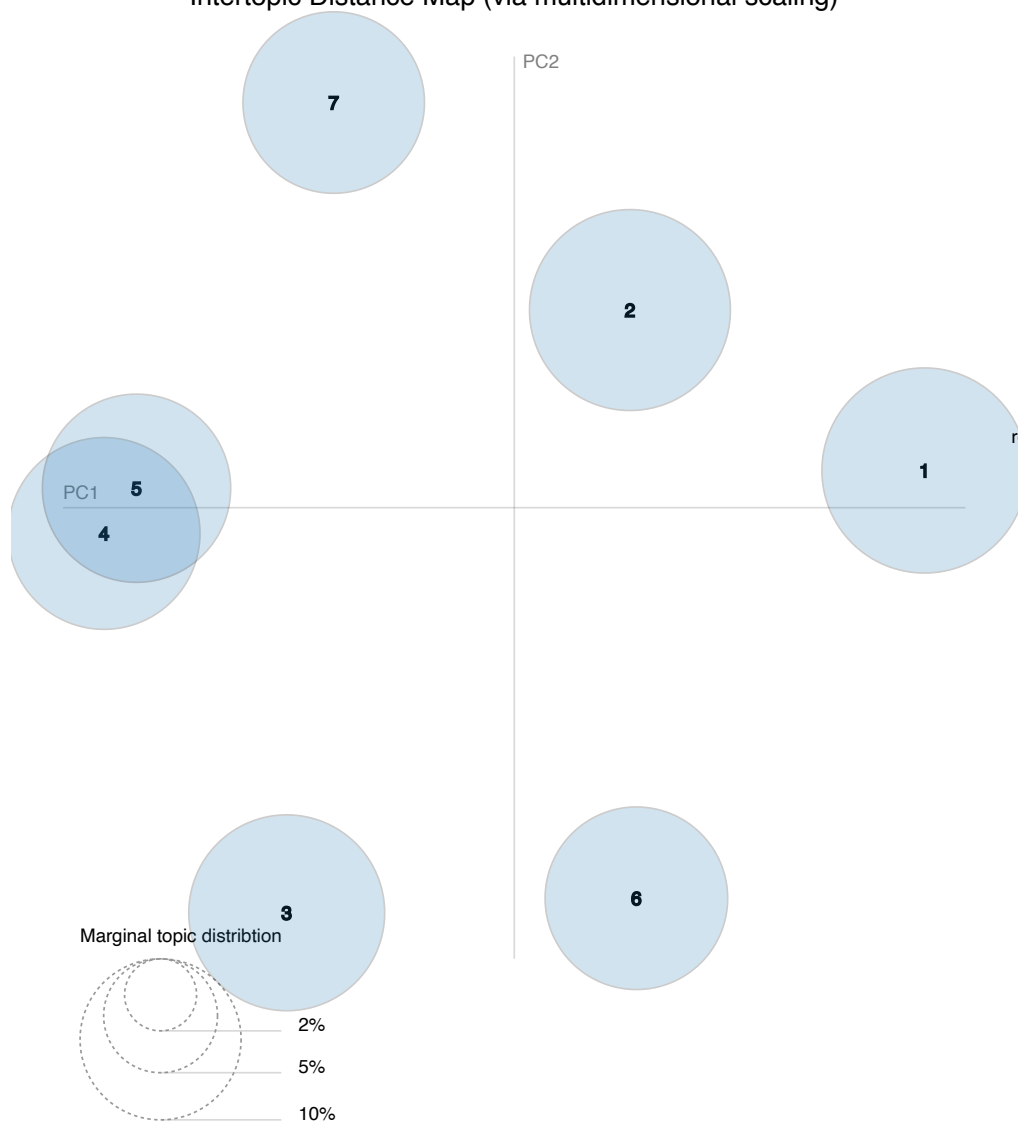
Selected Topic:

Slide to adjust relevance metric:(2)

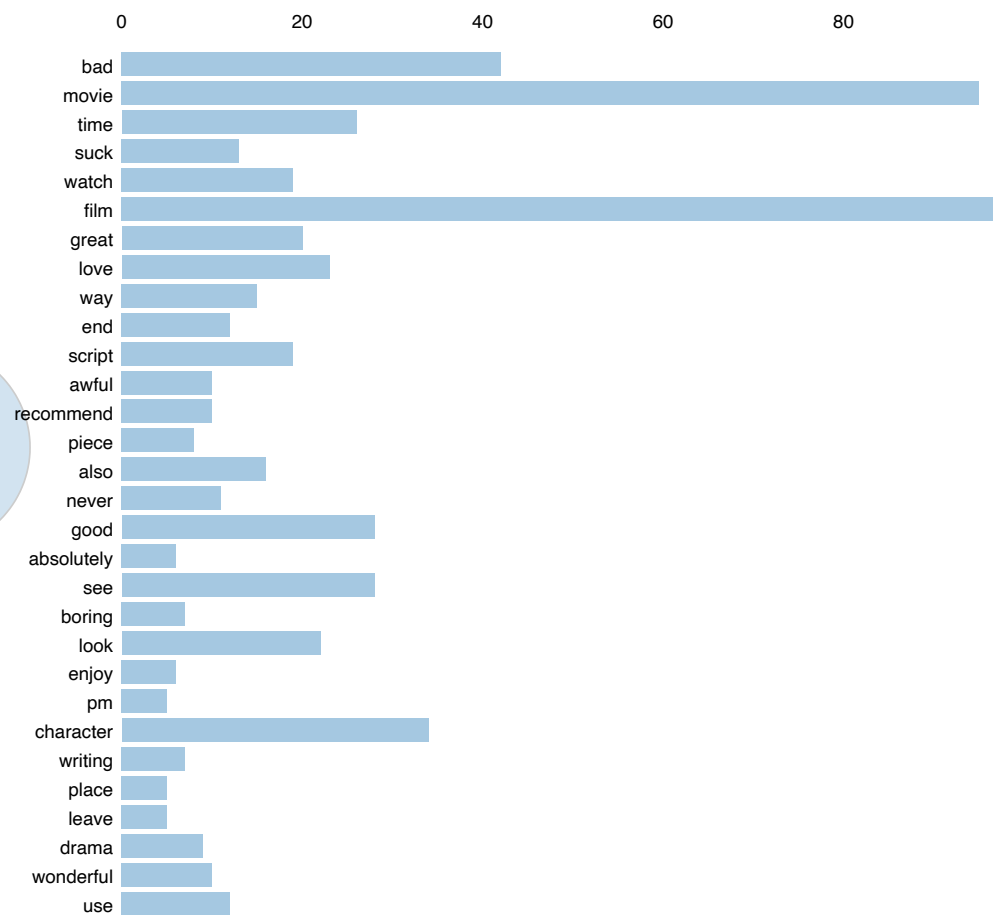
$\lambda = 1$

0.0 0.2 0.4 0.6 0.8 1.0

Intertopic Distance Map (via multidimensional scaling)



Top-30 Most Salient Terms⁽¹⁾



Overall term frequency

Estimated term frequency within the selected topic

1. saliency(term w) = frequency(w) * $[\sum_t p(t | w) * \log(p(t | w)/p(t))]$ for topics t ; see Chuang et. al (2012)
 2. relevance(term w | topic t) = $\lambda * p(w | t) + (1 - \lambda) * p(w | t)/p(w)$; see Sievert & Shirley (2014)

```
pyLDavis.display(visu_data_tfidf)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au
and should_run_async(code)
```

Selected Topic:

Slide to adjust relevance metric:(2)

$\lambda = 1$

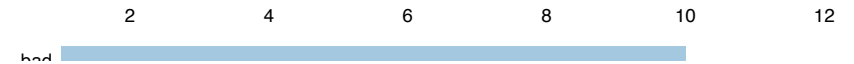
0.0 0.2 0.4 0.6 0.8 1.0

Intertopic Distance Map (via multidimensional scaling)



PC2

Top-30 Most Salient Terms⁽¹⁾



▼ Utilisation de LDA pour faire de la classification supervisée

En introduction, nous avons précisé que LDA était une approche de classification non supervisée. En effet, il n'existe pas de label de classes et nous avons vu que nous pouvions pour les différents documents leur attribuer des topics et même un topic dominant. En fait il est tout à fait possible de considérer LDA comme un moyen de réduire les dimensions (un peu comme une ACP) et donc de pouvoir utiliser les différents mots des topics comme représentant un document plutôt que le texte associé. Par la suite, il suffira d'utiliser ces mots clés pour pouvoir appliquer un algorithme de classification supervisée.


Dans cette section, nous montrons comment il est possible d'utiliser les topics pour classer les différentes opinions sur imdb, amazon et yelp.

```
# chargement des données
df_all = pd.read_csv("ReviewsLabelled.csv", names=['sentence', 'sentiment', 'source'],
                    header=0, sep='\t', encoding='utf8')

print (df_all.shape)
display (df_all)
```

```
(3000, 3)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au  
and should_run_async(code)
```

	sentence	sentiment	source	
0	So there is no way for me to plug it in here i...	0	amazon	
1	Good case, Excellent value.	1	amazon	
2	Great for the jawbone.	1	amazon	
3	Tied to charger for conversations lasting more...	0	amazon	
4	The mic is great.	1	amazon	

Le principe consiste à partir des différents documents du corpus d'apprendre les topics associés. Pour cela il faut appliquer les étapes que nous avons vu précédemment : pre-traitement, obtention de la matrice représentant les documents et du vocabulaire associé et apprentissage du modèle.

```
299/ Overall I was not impressed and would not go b... U yelp
```

```
stop = stopwords.words('english')

# enrichissement des stopwords
stop.extend(['always', 'try', 'go', 'get', 'make', 'would', 'really',
            'like', 'came', 'got'])
corpus_all, corpus_tfidf_all, dictionary_all, bigram_token_all=MyCleanTextsforLDA(df_all.sentence)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au  
and should_run_async(code)
```

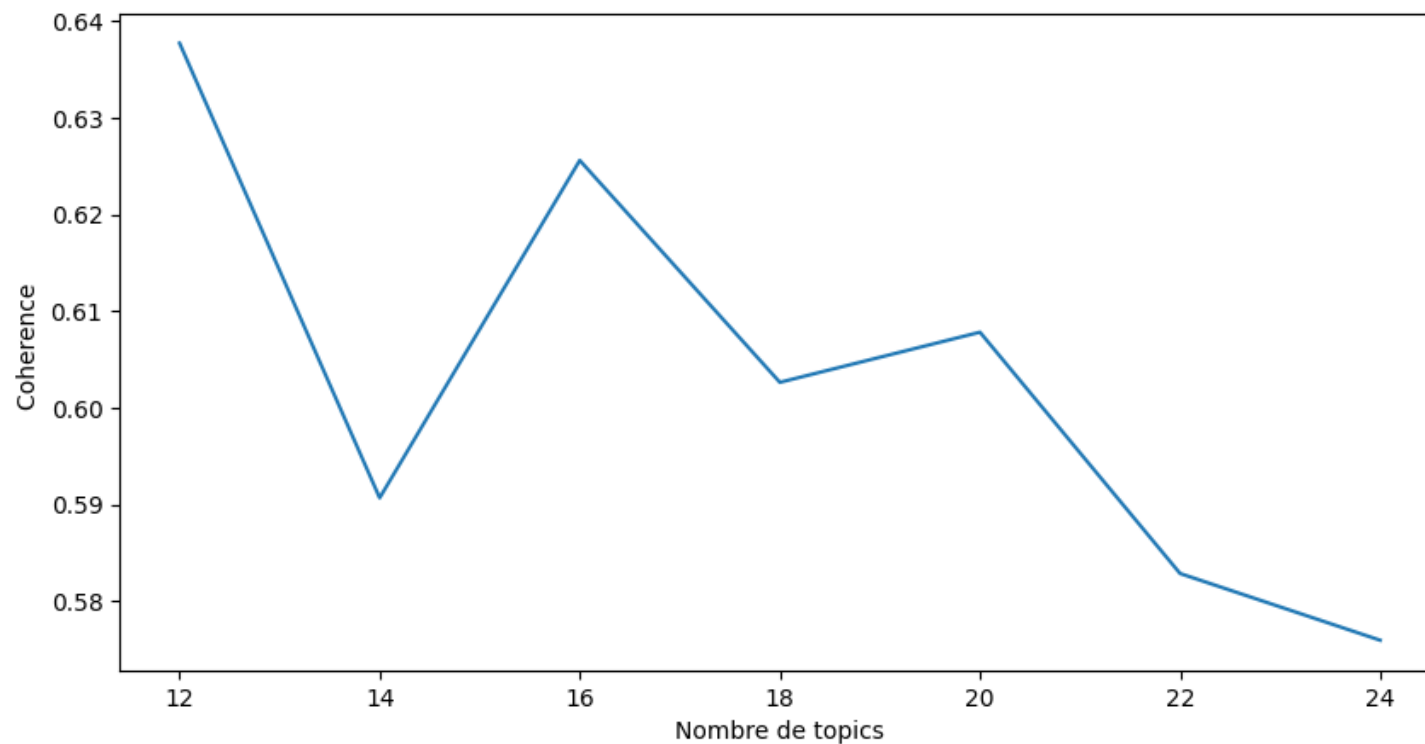
Nous pouvons effectuer une recherche rapide pour avoir une idée du meilleur nombre de topics à retenir.

```
start=12
stop=25
step=2
model_list, coherence_values = get_best_coherence_values(dictionary=dictionary_all,
                                                         corpus=corpus_all,
                                                         listtokens=bigram_token_all,
                                                         start=start, stop=stop, step=step)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au  
and should_run_async(code)
```

```
# affichage du graphe associé à la recherche du nombre de topics  
plt.figure(figsize=(10,5))  
x = range(start, stop, step)  
plt.plot(x, coherence_values)  
plt.xlabel("Nombre de topics")  
plt.ylabel("Coherence ")  
#plt.legend(("Valeurs de cohérences",coherence_values), loc='best')  
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au  
and should_run_async(code)
```



Un nombre entre 20 et 21 topics semble être un bon compromis

```

num_topics=21 # nombre de topics
num_words=50 # nombre de mots par topics

lda_model_all = gensim.models.ldamulticore.LdaMulticore(
    corpus=corpus_all,
    num_topics=num_topics,
    id2word=dictionary_all,
    chunksize=100,
    workers=7,
    passes=10,
    random_state=100,
    eval_every = 1,
    per_word_topics=True)

print ("Affichage des ",num_topics," différents topics pour le corpus BOW :\n")
for idx, topic in lda_model_all.print_topics(-1,num_words):
    print('Topic: {} Word: {}'.format(idx, topic))

```

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au and should_run_async(code)

Affichage des 21 différents topics pour le corpus BOW :

```

Topic: 0 Word: 0.053*"know" + 0.035*"never" + 0.034*"special" + 0.030*"real" + 0.019*"life" + 0.016*"want" + 0.016*"green" + 0.012*"get" +
Topic: 1 Word: 0.031*"take" + 0.026*"fantastic" + 0.025*"horrible" + 0.024*"terrible" + 0.022*"awful" + 0.018*"time" + 0.018*"awesome" + 0.
Topic: 2 Word: 0.023*"movie" + 0.023*"selection" + 0.016*"actually" + 0.015*"live" + 0.015*"pretty good" + 0.015*"close" + 0.014*"else" + 0
Topic: 3 Word: 0.035*"like" + 0.032*"people" + 0.022*"really" + 0.019*"check" + 0.017*"enough" + 0.016*"day" + 0.015*"area" + 0.015*"bit" +
Topic: 4 Word: 0.070*"bad" + 0.052*"order" + 0.029*"disappointed" + 0.023*"nice" + 0.019*"movie" + 0.019*"wait" + 0.019*"expect" + 0.016*"k
Topic: 5 Word: 0.049*"many" + 0.032*"definitely" + 0.019*"bad" + 0.019*"black" + 0.019*"family" + 0.016*"cast" + 0.015*"waste time" + 0.014
Topic: 6 Word: 0.044*"film" + 0.023*"menu" + 0.018*"incredible" + 0.015*"scene" + 0.014*"sure" + 0.014*"couple" + 0.013*"kid" + 0.012*"howe
Topic: 7 Word: 0.056*"see" + 0.037*"probably" + 0.030*"fresh" + 0.026*"wonderful" + 0.025*"excellent" + 0.016*"full" + 0.014*"consider" + 0
Topic: 8 Word: 0.116*"well" + 0.023*"play" + 0.021*"comfortable" + 0.018*"good" + 0.013*"spot" + 0.013*"music" + 0.012*"dialogue" + 0.012*"
Topic: 9 Word: 0.044*"way" + 0.036*"experience" + 0.032*"make" + 0.030*"much" + 0.023*"one" + 0.022*"bland" + 0.018*"watch" + 0.017*"show"
Topic: 10 Word: 0.055*"think" + 0.050*"say" + 0.042*"go" + 0.035*"feel" + 0.034*"come" + 0.023*"even" + 0.016*"film" + 0.014*"waste" + 0.01
Topic: 11 Word: 0.117*"love" + 0.022*"fact" + 0.018*"part" + 0.014*"yet" + 0.014*"huge" + 0.011*"pretty cool" + 0.010*"thin" + 0.008*"flick
Topic: 12 Word: 0.156*"good" + 0.021*"perfect" + 0.021*"great" + 0.014*"disappointing" + 0.013*"lady" + 0.012*"great service" + 0.012*"ever
Topic: 13 Word: 0.034*"give" + 0.033*"also" + 0.032*"recommend" + 0.027*"always" + 0.023*"star" + 0.021*"friend" + 0.020*"thing" + 0.016*"c
Topic: 14 Word: 0.054*"first" + 0.041*"suck" + 0.034*"little" + 0.022*"bring" + 0.021*"seem" + 0.017*"slow" + 0.015*"try" + 0.015*"get" + 0
Topic: 15 Word: 0.048*"amazing" + 0.022*"hand" + 0.020*"old" + 0.019*"small" + 0.014*"easy" + 0.013*"immediately" + 0.012*"phone" + 0.012*"
Topic: 16 Word: 0.134*"place" + 0.060*"time" + 0.040*"pretty" + 0.032*"work" + 0.026*"far" + 0.019*"lot" + 0.019*"night" + 0.018*"tell" + 0
Topic: 17 Word: 0.051*"get" + 0.027*"happy" + 0.019*"return" + 0.018*"interesting" + 0.017*"pleasant" + 0.017*"worth" + 0.016*"point" + 0.0
Topic: 18 Word: 0.088*"great" + 0.025*"movie" + 0.018*"ask" + 0.017*"absolutely" + 0.014*"fast" + 0.014*"end" + 0.014*"location" + 0.013*"c

```

Topic: 19 Word: 0.113*"service" + 0.074*"back" + 0.027*"fry" + 0.016*"authentic" + 0.015*"impressed" + 0.015*"also" + 0.014*"job" + 0.014*"Topic: 20 Word: 0.054*"phone" + 0.021*"case" + 0.015*"reasonable" + 0.013*"light" + 0.012*"cool" + 0.012*"buy" + 0.012*"stuff" + 0.012*"toa

Pour chaque document, nous pouvons donc rechercher le topic dominant :

```
df_topic_sents_keywords = format_topics_sentences(ldamodel=lda_model_all, corpus=corpus_all, texts=df_all.sentence)
```

```
display(df_topic_sents_keywords)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au
and should_run_async(code)
```



	topic_dominant	pourcentage_contrib	topic_keywords	sentence	
0	13	0.8095	give, also, recommend, always, star, friend, t...	So there is no way for me to plug it in here i...	
1	20	0.4185	phone, case, reasonable, light, cool, buy, stu...	Good case, Excellent value.	
2	18	0.6825	great, movie, ask, absolutely, fast, end, loca...	Great for the jawbone.	
3	20	0.8639	phone, case, reasonable, light, cool, buy, stu...	Tied to charger for conversations lasting more...	
4	18	0.6825	great, movie, ask, absolutely, fast, end, loca...	The mic is great.	
...	
2995	10	0.3492	think, say, go, feel, come, even, film, waste,...	I think food should have flavor and texture an...	
2996	10	0.5238	think, say, go, feel, come, even, film, waste,...	Appetite instantly gone.	
2997	0	0.0476	know, never, special, real, life, want, green,...	Overall I was not impressed and would not go b...	
2998	16	0.5238	place, time, pretty, work, far, lot, night, te...	The whole experience was underwhelming, and I ...	
2999	1	0.8413	take, fantastic, horrible, terrible, awful, ti...	Then, as if I hadn't wasted enough of my life ...	

3000 rows x 4 columns

```
# modification du dataframe pour intégrer les mots associés au topic dominant à chaque document
df_all['keywords']=df_topic_sents_keywords['topic_keywords']
display(df_all)
```



```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au
and should_run_async(code)
```

	sentence	sentiment	source	keywords	
0	So there is no way for me to plug it in here i...	0	amazon	give, also, recommend, always, star, friend, t...	
1	Good case, Excellent value.	1	amazon	phone, case, reasonable, light, cool, buy, stu...	
2	Great for the jawbone.	1	amazon	great, movie, ask, absolutely, fast, end, loca...	
3	Tied to charger for conversations lasting more...	0	amazon	phone, case, reasonable, light, cool, buy, stu...	
4	The mic is great.	1	amazon	great, movie, ask, absolutely, fast, end, loca...	
...	
2995	I think food should have flavor and texture an...	0	yelp	think, say, go, feel, come, even, film, waste,...	
2996	Appetite instantly gone.	0	yelp	think, say, go, feel, come, even, film, waste,...	
2997	Overall I was not impressed and would not go b...	0	yelp	know, never, special, real, life, want, green,...	
2998	The whole experience was underwhelming, and I ...	0	yelp	place, time, pretty, work, far, lot, night, te...	
2999	Then, as if I hadn't wasted enough of my life ...	0	yelp	take, fantastic, horrible, terrible, awful, ti...	

Il reste juste à appliquer une classification supervisée mais en considérant à présent les mots clés associés aux topics. Le code ci-dessous illustre le principe. Bien entendu il faudrait faire une cross validation, une recherche du meilleur classifieur ... comme nous l'avons vu dans le notebook "3 - Classification de données textuelles"

```
# selection des données
X=df_all.keywords
y=df_all.sentiment

# Création d'un jeu d'apprentissage et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# création du pipeline en ajoutant le classifieur
pipe = Pipeline([("tfidf_vectorizer", TfidfVectorizer()),
                  ("SVM", SVC())])
pipe.fit(X_train,y_train)

y_true, y_pred = y_test, pipe.predict(X_test)
```

```
print(classification_report(y_true, y_pred))
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au
and should_run_async(code)
```

	precision	recall	f1-score	support
0	0.57	0.45	0.50	475
1	0.50	0.62	0.55	425
accuracy			0.53	900
macro avg	0.54	0.53	0.53	900
weighted avg	0.54	0.53	0.53	900

Exercice : dans les cellules ci-dessus, faites varier le nombre de topics ainsi que le nombre de mots pour voir si vous pouvez améliorer la classification.

▼ Un dernier exercice pour terminer

Le 16 mars 2020, l'Institut Allen pour l'IA (AI2), en collaboration avec la Maison Blanche (OSTP), la Bibliothèque nationale de médecine (NLM), l'Initiative Chan Zuckerberg (CZI), Microsoft Research et Kaggle, coordonnés par le Center for Security and Emerging Technology (CSET) de l'Université de Georgetown, ont publié la première version de d'un jeu de données appelé : CORD-19.

CORD-19 est une collection de publications et de pré-impressions sur la Covid-19 et les coronavirus historiques associés (e.g. le SRAS et le MERS). L'objectif est de connecter la communauté d'apprentissage automatique avec des experts du domaine biomédical et des décideurs politiques pour aider à mieux comprendre le virus et à identifier des traitements efficaces pour Covid-19.

Les données sont mises à jour régulièrement et de plus amples informations sont disponibles ici :

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7251955/>

Kaggle propose ainsi un challenge : <https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge/tasks?taskId=570>

Exercice : l'objectif ici est d'utiliser des données du challenge (des articles de recherche) et de rechercher les topics qui sont présents d'abord dans le titre puis après dans le texte afin de les visualiser. Pour cela vous pourrez récupérer le corpus ici :

```
!wget https://www.lirmm.fr/~poncelet/Ressources/clean_noncomm_use.csv
```

```
--2023-09-07 23:02:40-- https://www.lirmm.fr/~poncelet/Ressources/clean_noncomm_use.csv
Resolving www.lirmm.fr (www.lirmm.fr)... /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run
and should_run_async(code)
193.49.104.251
Connecting to www.lirmm.fr (www.lirmm.fr)|193.49.104.251|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 147732912 (141M) [text/csv]
Saving to: 'clean_noncomm_use.csv'

clean_noncomm_use.c 100%[=====>] 140.89M  27.5MB/s   in 5.9s

2023-09-07 23:02:50 (23.7 MB/s) - 'clean_noncomm_use.csv' saved [147732912/147732912]
```

Ce corpus est un petit sous-ensemble extrait du challenge Kaggle. La transformation en csv a été réalisée par :

https://www.kaggle.com/xhlulu/cord-19-eda-parse-json-and-generate-clean-csv?select=clean_pmc.csv

Pour le lire il suffit d'exécuter la cellule suivante.

Remarque : les articles biomédicaux ont souvent une structure particulière avec des mots clés définis (e.g. 'Introduction', 'materials','methods','results',...) qui peuvent sans doute être ajoutés dans les stopwords. De plus, vous verrez qu'il y a beaucoup de bruits (e.g. 'creativecommons','license','http', etc.) et une première analyse des données en appliquant ce qui a été vu dans le notebook "2 - Ingénierie des données textuelles" pourrait vous aider à supprimer des mots inutiles.

Remarque : attention les articles peuvent être très longs aussi si vous ne souhaitez pas attendre trop longtemps les pré-traitements et l'obtention du modèle pour extraire les topics à partir du texte, vous pouvez prendre un sous-ensemble de 500 articles. Pour prendre un sous ensemble de 500 :

```
df=df.sample(n=500, replace=True, random_state=1)
df.reset_index(drop=True, inplace=True)
```

```
# lecture du fichier
df_covid = pd.read_csv("clean_noncomm_use.csv", encoding='utf8')

# taille du fichier
print (df_covid.shape)

# les premières lignes
display (df_covid)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au
and should_run_async(code)
(2489, 9)
```

	paper_id	title	authors	affiliations	abstract	text	bibliog
0	cd92f91038067e7a10aa27d676ce696e1e4d67ce	EXPERIMENTAL AND THERAPEUTIC MEDICINE Dimethyl...	Zhen-Hong Zhu, Wen-Qi Song, Chang-Qing Zhang, ...	Zhen-Hong Zhu (Shanghai Jiao Tong University, ...	Abstract\n\nMesenchymal stem cells have been w...	Introduction\n\nOsteonecrosis of the femoral h...	Avē necrosis femora Vē
1	bab279da548d8bd363acd5033e9dc54e7dbb7107	Effects of school breaks on influenza- like il...	Yanhui Chu, Zhenyu Wu, Jiayi Ji, Jingyi Sun, X...	Yanhui Chu, Zhenyu Wu (Fudan University, Shang...	NaN	INTRODUCTION\n\nSchoolchildren play a major ro...	Esti househc com transm
2	71edbd57cdd9af956a12054932e0cbdb87ce1fea	Social Network Characteristics and Body Mass I...	Won Joon Lee, Yoosik Youm, Yumie Rhee, Yeong-R...	Won Joon Lee (Yonsei University College of Med...	Abstract\n\nResearch has shown that obesity ap...	INTRODUCTION\n\nThe study of the effects of so...	The contr of the environr
3	2dfdbf2d6b77426866feaf93486327d372fd27c7	CLINICAL EXPERIMENTAL VACCINE RESEARCH	NaN	NaN	NaN	\n\nThere may be many reasons for the signific...	A short of vacci S L Plotk
4	0afa3ea846396533c7ca515968abcfea3f895082	Bone Marrow Dendritic Cells from Mice with an ...	Stacey L Burgess, Erica Buonomo, Maureen Carey...	Stacey L Burgess (Johns Hopkins Bloomberg Scho...	Abstract\n\nThere is an emerging paradigm that...	\n\nport neutrophil infiltration in inflammato...	WHO. ir consulta intesti
...
2484	3b8b2a835cfa770c6cef711d52e1f1e869d8aca8	Tumor-Treating Fields Induce RAW264.7 Macroph...	Jeong-In Park, Kyung-Hee Song, Seung-Youn Jung...	Jeong-In Park, Kyung-Hee Song, Seung-Youn Jung...	Abstract\n\nObjective: Tumor-treating fields a...	Introduction\n\nTumor-treating fields (TTFs) t...	NovoTTF phys chei

2485	500e585afac01e4b3847aa138db86b04352484c5	Efficacy and safety of Chou-Ling-Dan granules ...	Jiayang He, Zhengtu Li, Wanyi Huang, Wenda Gua...	Jiayang He, Zhengtu Li, Wanyi Huang, Wenda Gua...	NaN	\n\nIntroduction Chou-Ling-Dan (CLD) (Laggerap...	Effect epidemic heterog
2486	87f178ed1bcc5a747200ccd4adf42d883afd9fb7	Complete Genome Sequences of the SARS-CoV: the...	Shengli Bi, E'de Qin, Zuyuan Xu, Wei Li,...	Shengli Bi, E'de Qin (Chinese Academy of...	Abstract\n\nBeijing has been one of the epicen...	\n\nAccumulated number of probable cases and d...	Ile(16)/214 (7.5 27.1 43.
2487	5d4f1f02d0e731966ddddd635e8fa7bfdc169d3b9	Case Report A Rare Case of Autoimmune Polyglan...	Ryan Kenneth Smith, Beaumont Health, Peter M G...	Ryan Kenneth Smith, Beaumont Health, Peter M G...	Abstract\n\nAdrenal insufficiency is a rare, p...	Introduction\n\nAdrenal insufficiency is decre...	A insufficie Charman C
2488	c64c4f6efb9878bcc31980393c199d997805132a	Factors influencing Dipylidium sp. infection i...	Marion L East, Christoph Kurze, Kerstin Wilhel...	Marion L East (Leibniz Institute for Zoo and W...	Abstract\n\nWe provide the first genetic seque...	Introduction\n\nThe adult form of Dipylidium c...	Gapped I ar BLAST: generat

2489 rows x 9 columns

Solution :

```
# il suffit d'appliquer les principes précédents sur le titre et le texte
##### cas des titres

# selection du titre
df_title=df_covid['title'].copy()

print (df_title.shape)
display(df_title)

stop = stopwords.words('english')

# enrichissement des stopwords
```

```
stop.extend(['always','try','go','get','make','would','really',
            'like','came','got','article','creativecommons','license','http'])
corpus, corpus_tfidf, dictionary, bigram_token=MyCleanTextsforLDA(df_title)

num_topics=10 # nombre de topics
num_words=10 # nombre de mots par topics

lda_model_covid_tfidf = gensim.models.ldamulticore.LdaMulticore(
    corpus=corpus_tfidf,
    num_topics=num_topics,
    id2word=dictionary,
    chunksize=100,
    workers=7,
    passes=10,
    random_state=100,
    eval_every = 1,
    per_word_topics=True)

print ("Affichage des ",num_topics," différents topics pour le corpus TF-IDF :")

for idx, topic in lda_model_covid_tfidf.print_topics(-1,num_words):
    print('Topic : {} Words : {}'.format(idx, topic))

coherence_model_lda_tfidf = CoherenceModel(model=lda_model_covid_tfidf, texts=bigram_token, dictionary=dictionary,
    coherence='c_v')
coherence_lda = coherence_model_lda_tfidf.get_coherence()
print('Cohérence : ', coherence_lda)

print('Perplexité : ', lda_model_covid_tfidf.log_perplexity(corpus_tfidf))

visu_data_title = gensimvis.prepare(lda_model_covid_tfidf, corpus, dictionary)
pyLDavis.display(visu_data_title)
```

```
(2489,)
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` au
and should_run_async(code)
0     EXPERIMENTAL AND THERAPEUTIC MEDICINE Dimethyl...
1     Effects of school breaks on influenza- like il...
2     Social Network Characteristics and Body Mass I...
3         CLINICAL EXPERIMENTAL VACCINE RESEARCH
4     Bone Marrow Dendritic Cells from Mice with an ...
...
2484    Tumor-Treating Fields Induce RAW264.7 Macroph...
2485    Efficacy and safety of Chou-Ling-Dan granules ...
2486    Complete Genome Sequences of the SARS-CoV: the...
2487    Case Report A Rare Case of Autoimmune Polygla...
2488    Factors influencing Dipylidium sp. infection i...
Name: title, Length: 2489, dtype: object
Affichage des 10 différents topics pour le corpus TF-IDF :
Topic : 0 Words : 0.005*"patient" + 0.004*"pneumonia" + 0.004*"infection" + 0.003*"community acquire" + 0.003*"child" + 0.003*"coronavirus"
Topic : 1 Words : 0.004*"sar cov" + 0.003*"use" + 0.003*"genome organization" + 0.003*"type" + 0.003*"elderly" + 0.003*"rate" + 0.003*"new"
Topic : 2 Words : 0.010*"clinical experimental" + 0.010*"vaccine research" + 0.004*"virus" + 0.004*"virus infection" + 0.004*"cell" + 0.004
Topic : 3 Words : 0.013*"license" + 0.012*"creativecommons org" + 0.008*"case report" + 0.006*"health" + 0.005*"support information" + 0.004
Topic : 4 Words : 0.006*"emerge microbe" + 0.004*"infection" + 0.004*"ar ticle" + 0.004*"supplementary information" + 0.003*"supplemental r
Topic : 5 Words : 0.009*"infection chemotherapy" + 0.005*"health care" + 0.003*"editorial commentary" + 0.003*"publication" + 0.003*"global
Topic : 6 Words : 0.006*"virus" + 0.005*"review" + 0.005*"target" + 0.005*"infection" + 0.004*"coronavirus infection" + 0.004*"mer cov" + 0
Topic : 7 Words : 0.003*"disease" + 0.003*"porcine epidemic" + 0.003*"infection" + 0.003*"pathway" + 0.003*"hepatitis" + 0.003*"peer review
Topic : 8 Words : 0.007*"original article" + 0.005*"brief communication" + 0.004*"method" + 0.004*"virus" + 0.003*"avian influenza" + 0.003
Topic : 9 Words : 0.004*"veterinary science" + 0.003*"disease" + 0.003*"efficacy" + 0.003*"vaccine" + 0.003*"overview" + 0.003*"improve" +
Cohérence : 0.6375661721203262
Perplexité : -11.197188189857117
```

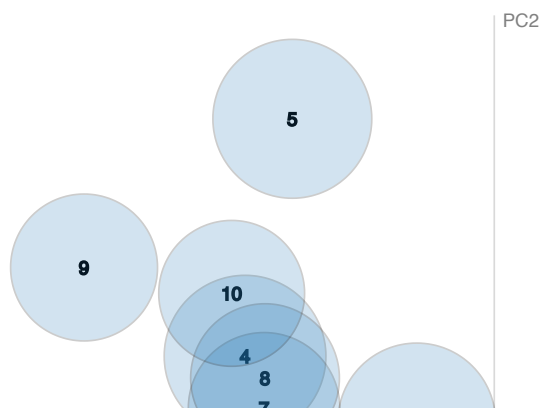
Selected Topic:

Slide to adjust relevance metric:(2)

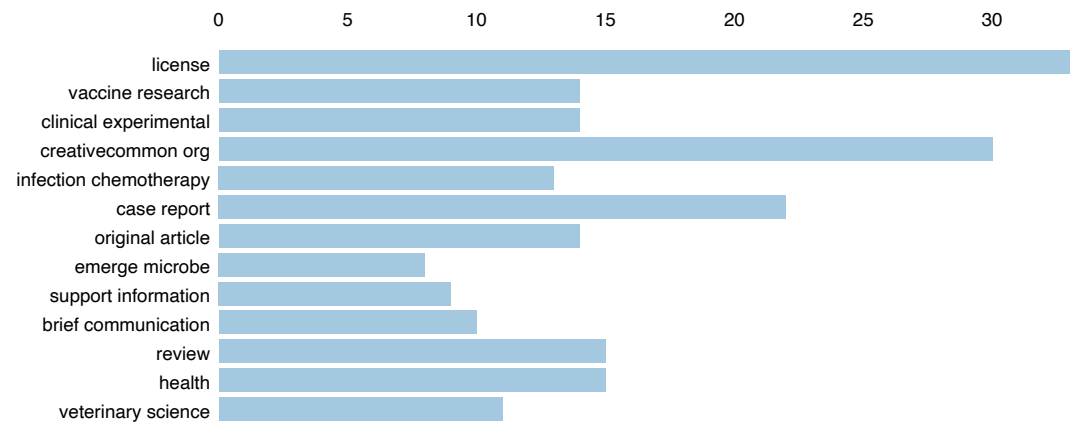
$\lambda = 1$

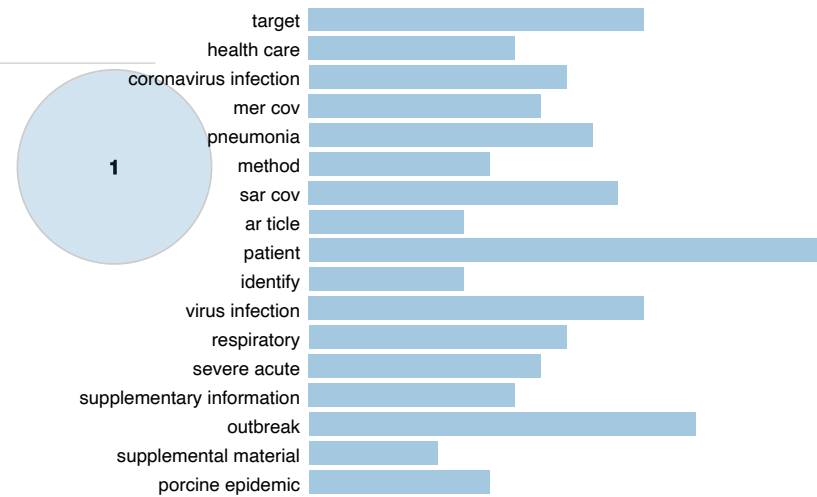
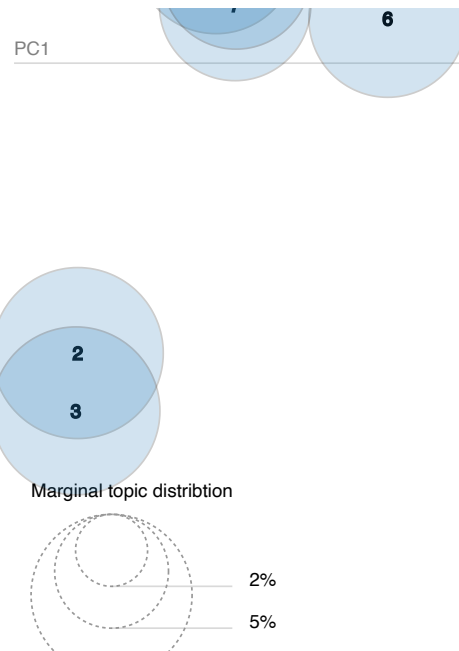


Intertopic Distance Map (via multidimensional scaling)



Top-30 Most Salient Terms⁽¹⁾





Overall term frequency

Estimated term frequency within the selected topic

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = $\lambda * p(w | t) + (1 - \lambda) * p(w | t)/p(w)$; see Sievert & Shirley (2014)

```
# Traitement du texte
# selection du texte
df_text=df_covid['text'].copy()

print (df_text.shape)
display(df_text)
# selection d'un echantillon de 500 articles
df_text=df_text.sample(n=500, replace=True, random_state=1)
print ("après tirage de 500 articles au hasard ")
df_text.reset_index(drop=True, inplace=True)
print (df_text.shape)
display(df_text)

stop = stopwords.words('english')

# enrichissement des stopwords
stop.extend(['always','try','go','get','make','would','really',
            'like','came','got','article','introduction','materials','methods',
            'results','creativecommons','license','http','fig','figure'])

corpus, corpus_tfidf, dictionary, bigram_token=MyCleanTextsforLDA(df_text)
```