Projet Advanced Machine Learning: Text synthesis

Simon Hervé
Jean-Louis Delebecque
Lien du github: https://github.com/jdelebec/Advanced_machine_learning_project1

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

Extraction des textes

On utilise la librairie Beautifulsoup pour extraire les données des XML.

```
# Packages for extracting data

import bs4
from bs4 import BeautifulSoup as bs
import lxml
import pandas as pd
import numpy as np
```

Afin de pouvoir travailler à deux sur le projet nous avons fait un google collabs partagé, d'ou la présence de deux path

```
simon_path = "/content/drive/MyDrive/ESILV S9/Advanced machine learning/corpus_taln_v1.tei
jl_path = "/content/drive/MyDrive/Advanced_ml/corpus_taln_v1.tei.xml"
```

```
#Pour JL
content = []
with open(jl_path, "r") as file:
  content = file.readlines()
  content = "".join(content)
  soup = bs(content, "lxml")
```

```
#Pour Simon
content = []
with open(simon_path, "r") as file:
  content = file.readlines()
  content = "".join(content)
  soup = bs(content, "lxml")
```

La stratégie d'extraction des données est d'appliquer la fonction get_text() à chaque balise dans le corpus_taln_v1.tei.xml. La function va extraire le titre de la publication, le nom du ou des auteurs quand il y en a, l'email du ou des auteurs si il y en a. De plus, elle récupère les abstract en français et en anglais ainsi que les mots clés en français et en anglais. Pour la partie text, la fonction compile tous les textes en une partie


```
dic = {}

#============================================================
language = soup.fin
title = soup.find("title").get_text().replace("\n\t", "")
author = soup.find("author")
author_names = author.find_all("name")
tab_name = []
for name in author_names:
  tab_name.append(name.get_text())
publication_place = soup.find("pubplace").get_text()
publication_date = soup.find("publicationstmt").find("date").get_text()
editor_name = soup.find("editor").find("name").get_text()
monogr_title = soup.find("monogr").find("title").get_text()
monogr_date = soup.find("monogr").find("date").get_text()
abstract = soup.find("text").find_all("div", {"type": "abstract"})
abstract_fr = abstract[0].find("p").get_text().replace("\n\t", "")
abstract_eng = abstract[1].find("p").get_text().replace("\n\t", "")
keywords = soup.find("text").find_all("div", {"type": "keywords"})
keywords_fr = keywords[0].find("p").get_text().replace("\n\t", "")
keywords_eng = keywords[1].find("p").get_text().replace("\n\t", "")
sections = soup.find_all("div", {"type": "section"})
tab = []

#============================================================

# Introduction - Sections - Conclusion
sections = soup.find_all("div", {"type": "section"})
for section in sections:
  # title section
  try:
    title_sec = section.get_text().split("\n")[2]
    tab.append(title_sec)
  except:
    pass
  # paragraph section
  try:
    parag_sec = section.find("p").get_text().replace("\n\t", "")
    tab.append(parag_sec)
  except:
    pass
  # Subsection
  try:
    subsections = section.find_all("div", {"type": "subsection"})
    for sub in subsections:
      # title subsection
      title_sub = sub.get_text().split("\n")[2]
```

```
        title_sub = sub.get_text().split("\n")[2]
        tab.append(title_sub)

        # paragraphs subsection
        paragraphs = sub.find_all("p")
        for parag in paragraphs:
          tab.append(parag.get_text().replace("\n\t", ""))
```

Updating preview...

```
        try:
          notes = sub.find_all("note")
          for note in notes:
            tab.append(note.get_text())
        except:
          pass

    except:
      pass

    # note
    try:
      notes = section.find_all("note")
      for note in notes:
        tab.append(note.get_text())
    except:
      pass


  # Bibl

  try:
    bibl = soup.find("bibl").get_text()
    tab.append(bibl)
  except:
    pass

  #================================================================

  dic["title"] = title
  dic["author_name"] =  ",".join(str(elem) for elem in tab_name)
  try:
    tab_email = []
    author_emails = author.find_all("email")
    for email in author_emails:
      tab_email.append(email.get_text().replace("\n", ""))
    dic["author_email"] =  ",".join(str(elem) for elem in tab_email)
  except:
    dic["author_email"] = None
  dic["publication_place"] = publication_place
  dic["publication_date"] = publication_date
  dic["editor_name"] = editor_name
  dic["monogr_title"] = monogr_title
  dic["monogr_date"] = monogr_date
  dic["abstract_fr"] = abstract_fr
  dic["abstract_eng"] = abstract_eng
  dic["keywords_fr"] = keywords_fr
```

```
  dic["keywords_fr"] = keywords_fr
  dic["keywords_eng"] = keywords_eng
  dic["text"] = " ".join(str(elem) for elem in tab)

  #>
  return dic
```

Updating preview...

```
corpus = []
texts = soup.find_all("tei")
for text in texts:
  corpus.append(get_text(text))
```

On transforme ensuite le tableau de dictionnaires contenant les données en un dataframe grâce à la fonction DataFrame de la librairie pandas.

```
df = pd.DataFrame(corpus)
```

```
df.head()
```

	title	author_name	author_email	public
0	\nÉléments de conception d'un système d'interp...	Delphine Battistelli,Cyril Valliez	battiste@msh-paris.fr,valliez@msh-paris.fr	
1	\nInformatisation du dictionnaire explicatif e...	Gilles Sérasset	Gilles.Serasset@imag.fr	
2	\nConstruction d'une	Pierre Zweigenbaum,Jacques	pz@biomath.jussieu.fr,jb@biomath.jussieu.fr	

On enregistre le dataframe pour l'exporter.

```
#Pour Simon
df.to_csv("/content/drive/MyDrive/ESILV S9/Advanced machine learning/corpus.csv")
```

```
#Pour JL
df.to_csv("/content/drive/MyDrive/Advanced_ml/corpus.cs")
```

Pour la suite du projet on va se concentrer sur les abstract_fr car c'est la colonne où il y a le moins de données manquantes (67).

On installe le lemmatizer fr_core_news_sm, mais on pourrait aussi utiliser le fr_core_news_md.

```
!python -m spacy download fr_core_news_sm
```

```
Requirement already satisfied: fr_core_news_sm==2.2.5 from https://github.com/explosi
Requirement already satisfied: spacy>=2.2.2 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.6/di
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.6/
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: blis<0.5.0,>=0.4.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /usr/local/lib/python3.6/di
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: importlib-metadata>=0.20; python_version < "3.8" in /u
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-packages (f
✓ Download and installation successful
You can now load the model via spacy.load('fr_core_news_sm')
```

Updating preview...

On importe les packages nécéssaires au processing des données textuelles.

```
# Packages for processing

import nltk
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
nltk.download('punkt')
nltk.download("stopwords")
from nltk.corpus import stopwords
french_stopwords = set(stopwords.words("french"))
import unicodedata
import re
import spacy
import fr_core_news_sm
nlp = fr_core_news_sm.load()
```

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

Processing

Crée des fonctions qui font le processing classique de texte. Dans notre démarche on va essayer de voir l'incidence du processing sur les résultats. Pour cela on créer plusieurs wrappers

où l'ordre d'appel des fonctions est différent. On fait cette démarche car on se rend compte qu'il y a de mauvaises transcriptions après processing avec par exemple issus qui devient i us.

```
# Processing
```



```
    norm = unicodedata.normalize('NFKD', str(text))
    ascii = norm.encode('ascii', 'ignore')
    text = ascii.decode('utf-8', 'ignore')
    return text

def lower_letters(text):

    return text.lower()

def drop_slash_n(text):

    return re.sub(r"\n", "", text)

def drop_special_carac(text):

    return re.sub(r"[^a-zA-Z\d\s]", " ", text)

def drop_brackets(text):

    return re.sub(r"\[(.*?)\]", "", text)

def drop_extra_space(text):

    return re.sub(r"s{2,}", " ", text)

def remove_stop_words(text):

    text = word_tokenize(text)
    text = " ".join([word for word in text if not word in french_stopwords])
    return text

def lemmatize(text):

    text = nlp(text)
    text = " ".join([word.lemma_ for word in text])
    return text

def drop_double_space(text):
    return re.sub(' +', ' ', text)

def drop_single_char(text):
    return re.sub('(\\b[A-Za-z] \\b|\\b [A-Za-z]\\b)', '', text)

# ================================================================================

# Wrapper
```

```
def processing1(text):

    text = str(text)
    text = remove_accents(text)
    text = lower_letters(text)
    text = drop_extra_space(text)
    text = remove_stop_words(text)
    text = lemmatize(text)

    return text

def processing2(text):

    text = str(text)
    text = remove_accents(text)
    text = lower_letters(text)
    text = drop_slash_n(text)
    text = drop_special_carac(text)
    text = remove_stop_words(text)
    text = lemmatize(text)

    return text

def processing3(text):

    text = str(text)
    text = lower_letters(text)
    text = remove_accents(text)
    text = remove_stop_words(text)
    text = drop_slash_n(text)
    text = drop_special_carac(text)
    text = lemmatize(text)

    return text

def processing4(text):

    text = str(text)
    text = remove_accents(text)
    text = lower_letters(text)
    text = drop_slash_n(text)

    return text

def processing5(text):

    text = str(text)
    text = lower_letters(text)
    text = remove_stop_words(text)
    text = remove_accents(text)
    text = drop_slash_n(text)
    text = drop_special_carac(text)
```

```
    text = lemmatize(text)

    return text

def processing6(text):
```

Updating preview...

```
    text = remove_stop_words(text)
    text = remove_accents(text)
    text = drop_slash_n(text)
    text = drop_special_carac(text)

    return text

def processing7(text):

    text = str(text)
    text = drop_brackets(text)
    text = lower_letters(text)
    text = remove_stop_words(text)
    text = remove_accents(text)
    text = drop_slash_n(text)
    text = drop_special_carac(text)
    text = drop_double_space(text)
    text = drop_single_char(text)

    return text

def processing8(text):

    text = str(text)
    text = drop_brackets(text)
    text = lower_letters(text)
    text = lemmatize(text)
    text = remove_stop_words(text)
    text = remove_accents(text)
    text = drop_slash_n(text)
    text = drop_special_carac(text)
    text = drop_double_space(text)
    text = drop_single_char(text)

    return text
```

On applique les différents processing qui génèrent une nouvelle colonne à chaque fois.

```
df["abstract_fr_proc_1"] = df.apply(lambda x: processing1(x["abstract_fr"]), axis=1)
df["abstract_fr_proc_2"] = df.apply(lambda x: processing2(x["abstract_fr"]), axis=1)
df["abstract_fr_proc_3"] = df.apply(lambda x: processing3(x["abstract_fr"]), axis=1)
df["abstract_fr_proc_4"] = df.apply(lambda x: processing4(x["abstract_fr"]), axis=1)
df["abstract_fr_proc_5"] = df.apply(lambda x: processing5(x["abstract_fr"]), axis=1)
df["abstract_fr_proc_6"] = df.apply(lambda x: processing6(x["abstract_fr"]), axis=1)
```

```
df["abstract_fr_proc_7"] = df.apply(lambda x: processing7(x["abstract_fr"]), axis=1)
df["abstract_fr_proc_8"] = df.apply(lambda x: processing8(x["abstract_fr"]), axis=1)
```

```
df.shape
```

```
(1602, 21)
```

Updating preview...

```
t()[0]
```

```
1535
```

```
df[df["abstract_eng"] != 'None'].count()[0]
```

```
1492
```

Nous avons décidé de nous focaliser sur les abstract français car les textes sont en français et il y a plus d'abstract français que d'anglais (de pas beaucoup: environ 60)

Listes de fréquences de mots

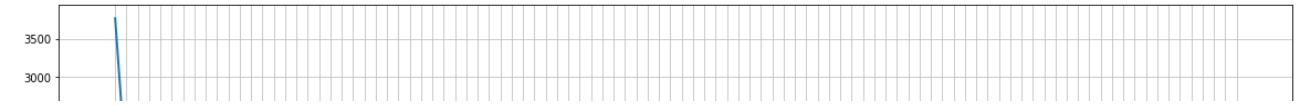
Dans cette partie nous allons voir quels sont les mots les plus fréquents dans chaque colonne. L'idée est de voir si le processing a une incidence. On remarquera que selon le processing on a des mots communs qui sont fréquents.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt


fdist1 = FreqDist()
for text in df["abstract_fr_proc_1"]:
    for word in word_tokenize(text):
        fdist1[word] += 1

#=========================================

fdist2 = FreqDist()
for text in df["abstract_fr_proc_2"]:
    for word in word_tokenize(text):
        fdist2[word] += 1

#=========================================

fdist3 = FreqDist()
for text in df["abstract_fr_proc_3"]:
    for word in word_tokenize(text):
        fdist3[word] += 1

#=========================================
```

```
fdist4 = FreqDist()
for text in df["abstract_fr_proc_4"]:
    for word in word_tokenize(text):
        fdist4[word] += 1

#=========================================
```

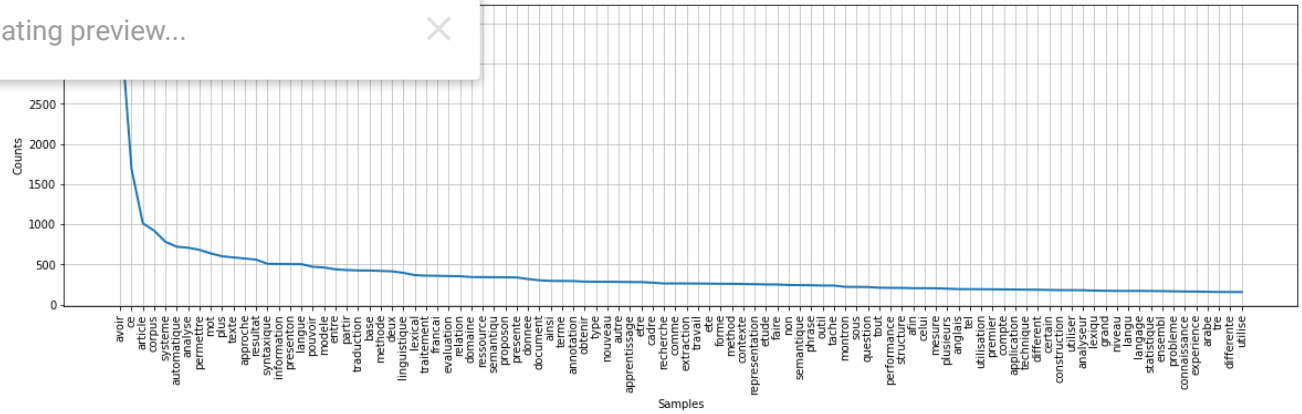
Updating preview...

```
for text in df["abstract_fr_proc_5"]:
    for word in word_tokenize(text):
        fdist5[word] += 1

#=========================================

fdist6 = FreqDist()
for text in df["abstract_fr_proc_6"]:
    for word in word_tokenize(text):
        fdist6[word] += 1

#=========================================

fdist7 = FreqDist()
for text in df["abstract_fr_proc_7"]:
    for word in word_tokenize(text):
        fdist7[word] += 1

#=========================================

fdist8 = FreqDist()
for text in df["abstract_fr_proc_8"]:
    for word in word_tokenize(text):
        fdist8[word] += 1


plt.figure(figsize=(20, 5))
fdist1.plot(100)
```

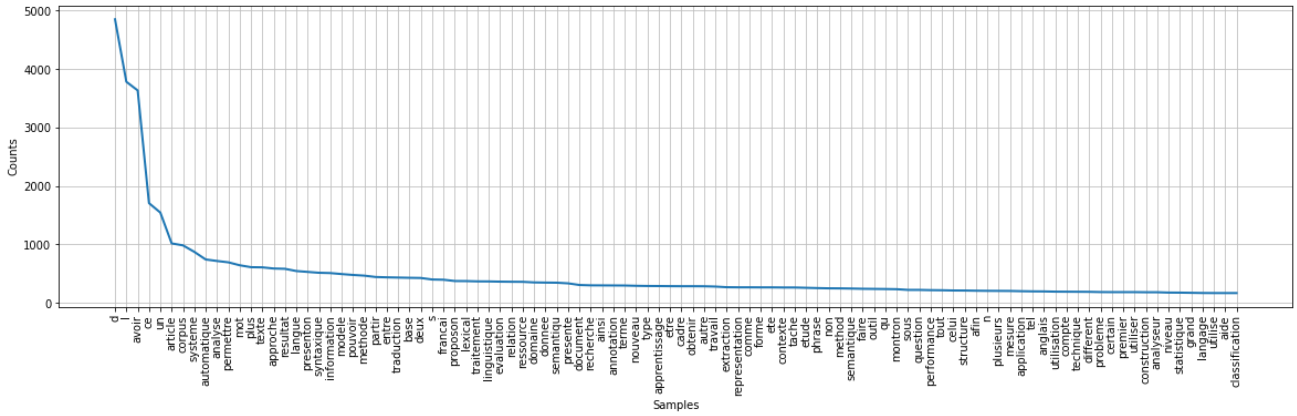



```
plt.figure(figsize=(20, 5))
fdist2.plot(100)
```

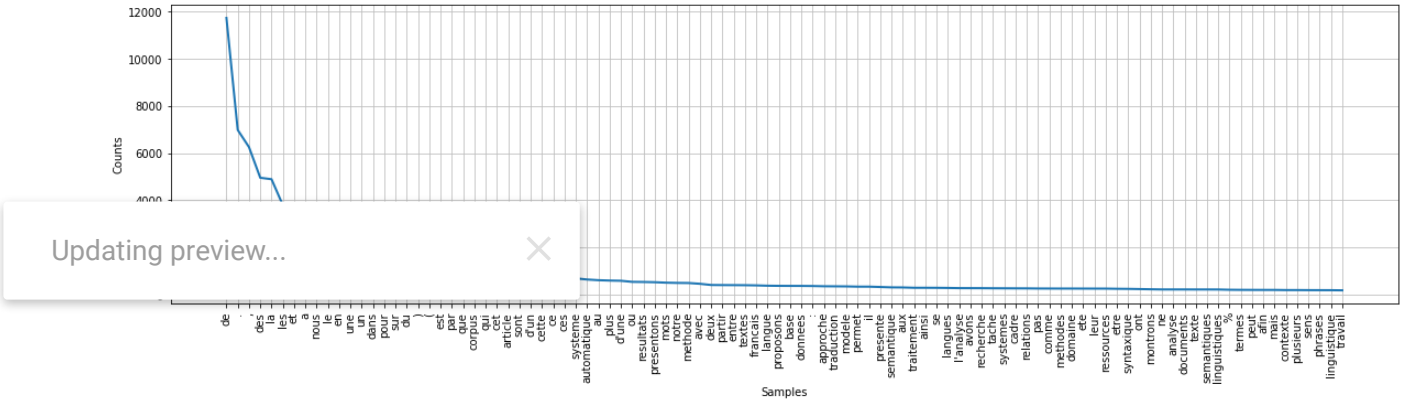
Updating preview...



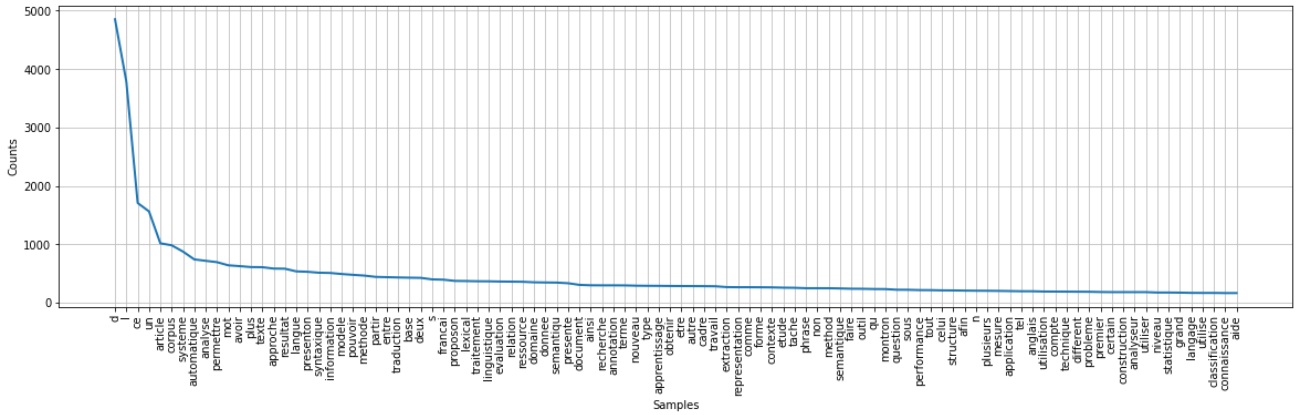

```
plt.figure(figsize=(20, 5))
fdist3.plot(100)
```



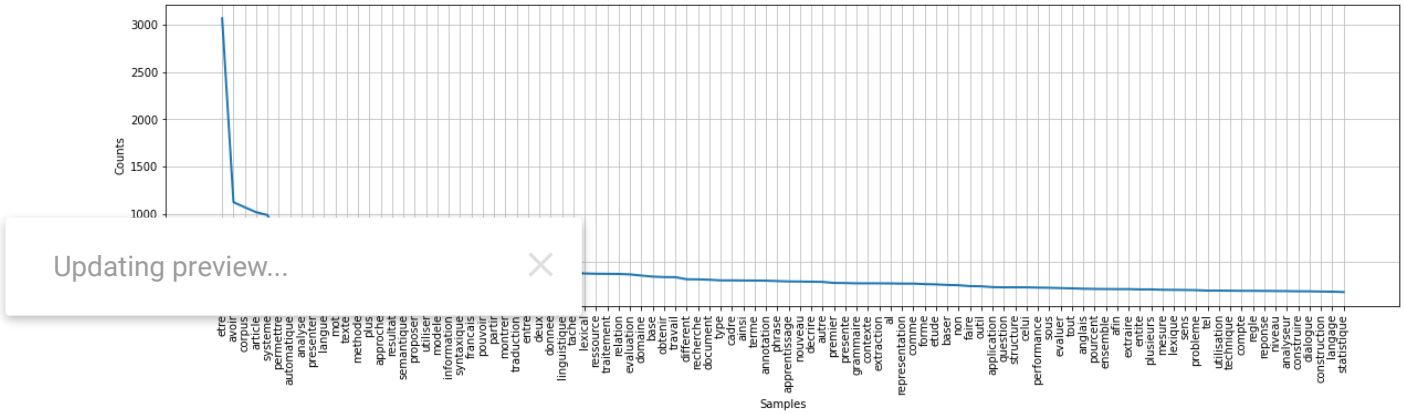

```
plt.figure(figsize=(20, 5))
fdist4.plot(100)
```



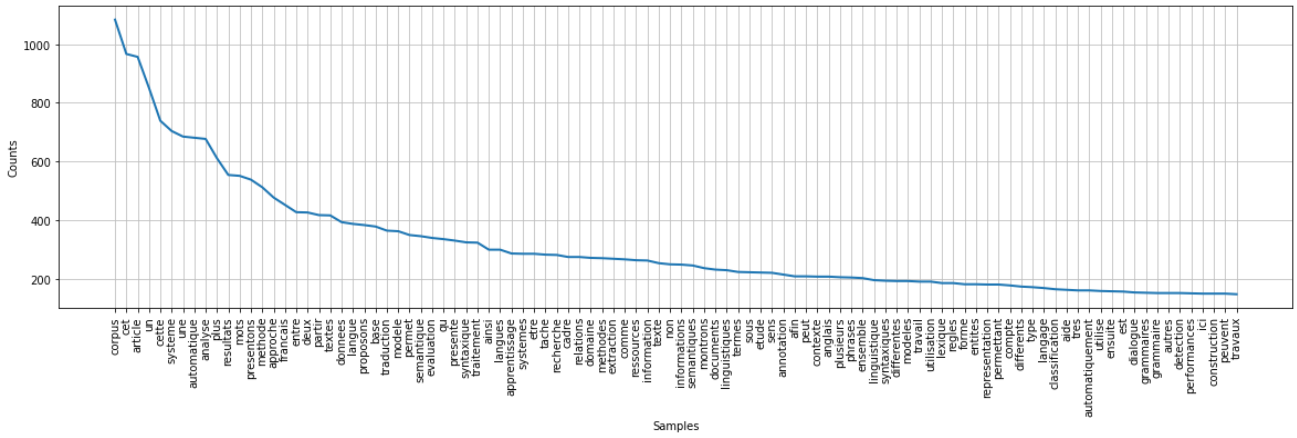

```
plt.figure(figsize=(20, 5))
fdist5.plot(100)
```



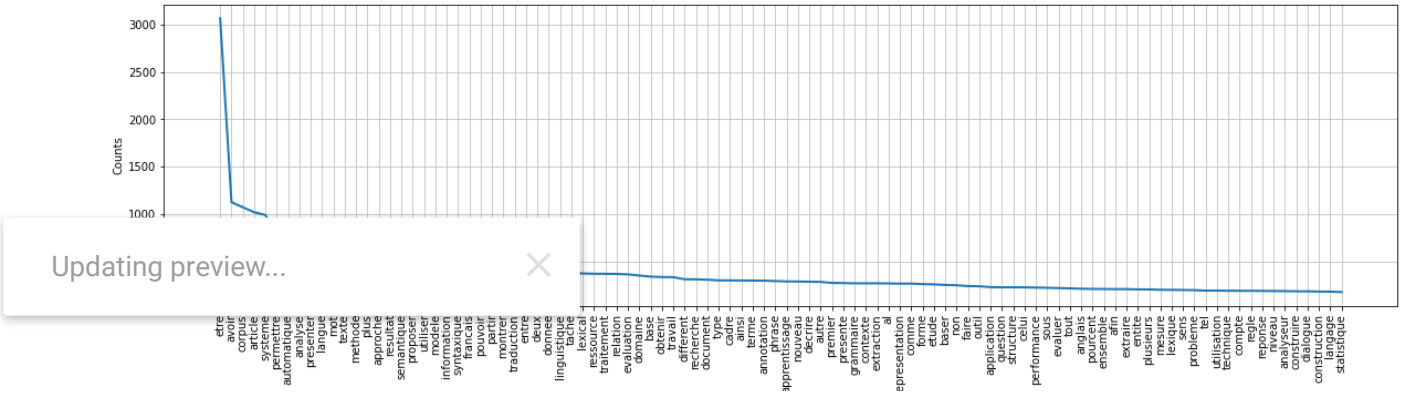

```
plt.figure(figsize=(20, 5))
fdist6.plot(100)
```




```
plt.figure(figsize=(20, 5))
fdist7.plot(100)
```




```
plt.figure(figsize=(20, 5))
fdist8.plot(100)
```



D'après les graphiques, on remarque que les processing les plus pertinents sont les deux premiers, l'avant avant dernier et le dernier. En effet, on ne retrouve pas les mots communs de la langue français ou des lettres comme d. On gardera le dernier, soit avec le processing 8 pour la suite.

Part of speech

```
#On utilise un part of speech français trouver à l'url suivant:
#https://nlp.stanford.edu/software/tagger.shtml
import nltk
from nltk.tag.stanford import StanfordPOSTagger
pos_tagger = StanfordPOSTagger("/content/drive/MyDrive/Advanced_ml/stanford-postagger-full
def pos_tag(sentence):
    tokens = nltk.word_tokenize(sentence)
    tags = pos_tagger.tag(tokens)
    return tags
```

```
/usr/local/lib/python3.6/dist-packages/nltk/tag/stanford.py:149: DeprecationWarning:
The StanfordTokenizer will be deprecated in version 3.2.5.
Please use nltk.tag.corenlp.CoreNLPPOSTagger or nltk.tag.corenlp.CoreNLPNERTagger ins
  super(StanfordPOSTagger, self).__init__(*args, **kwargs)
```

```
#Pour le processing 8 on essaye (risque de ne pas être interresant à cause du lemmatizer)
#On se limite a 50 textes pour l'instant (le temps de calcul est très long pour le part of
part_of_speech_8 = []
fdist_PoS_8 = FreqDist()
i = 0
for text in df["abstract_fr_proc_8"]:
    for word in pos_tag(text):
      fdist_PoS_8[word[1]]+=1
    i += 1
    if i == 50:
      break

fdist_PoS_8
```

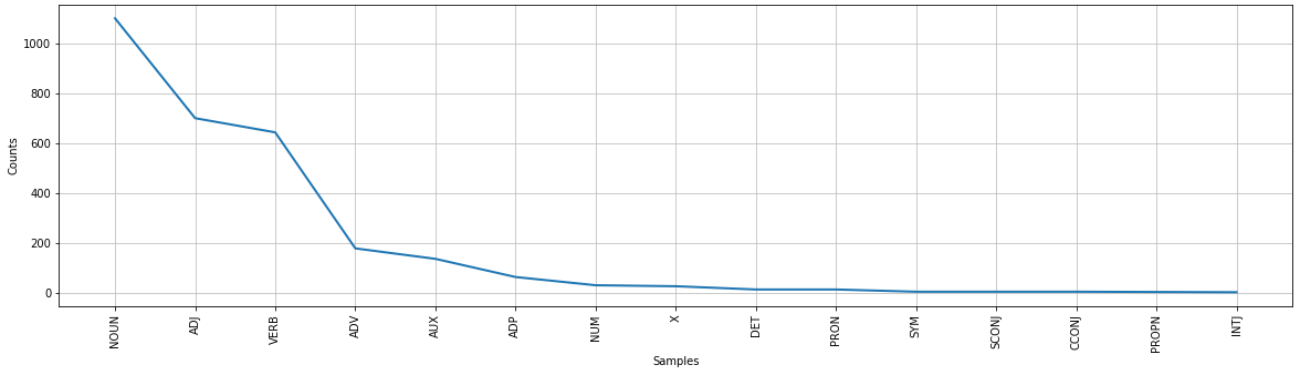
```
FreqDist({'ADJ': 700,
```

```
'ADP': 62,
'ADV': 177,
'AUX': 135,
'CCONJ': 3,
'DET': 12,
'INTJ': 1,
'NOUN': 1101,
```

Updating preview...

```
'SCONJ': 3,
'SYM': 3,
'VERB': 643,
'X': 25})
```

```
from matplotlib import pyplot as plt
plt.figure(figsize=(20, 5))
fdist_PoS_8.plot(100)
```



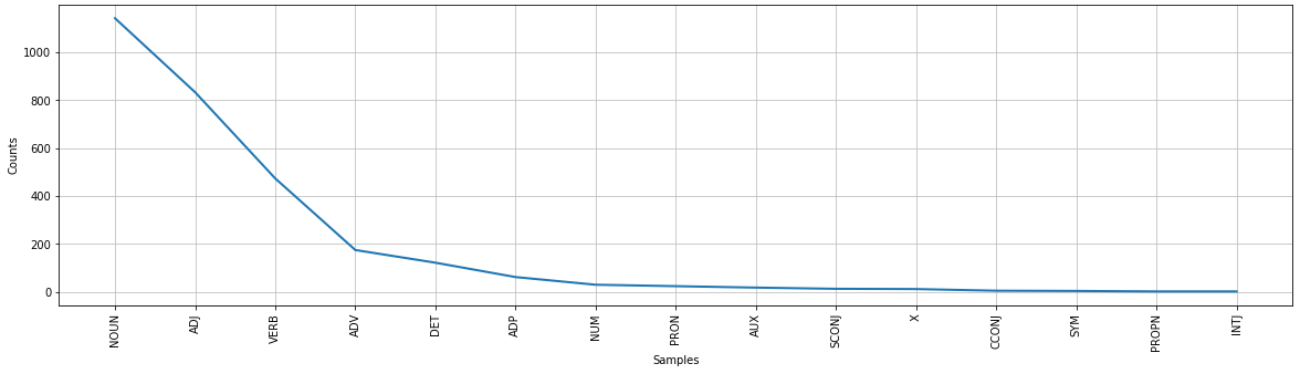
On le voit ici il y a énormement de verbe (assez rare) ce qui est du a la lemmatization
Comparons avec un processing sans lemmatization

```
#Pour observer une différence entre avec ou sans le lemmatizer nous allons utiliser le pro
#On se limite aussi au 50 premier texte
part_of_speech_7 = []
fdist_PoS_7 = FreqDist()
i = 0
for text in df["abstract_fr_proc_7"]:
    #print(pos_tag(text))
    for word in pos_tag(text):
      fdist_PoS_7[word[1]]+=1
    i += 1
    if i == 50:
      break
fdist_PoS_7
```

```
FreqDist({'ADJ': 833,
          'ADP': 61,
          'ADV': 174,
          'AUX': 17,
          'CCONJ': 4,
          'DET': 121,
          'PRON': 23,
          'PROPN': 1,
          'SCONJ': 12,
          'SYM': 3,
          'VERB': 473,
          'X': 11})
```

Updating preview...

```
from matplotlib import pyplot as plt
plt.figure(figsize=(20, 5))
fdist_PoS_7.plot(100)
```



On le voit ici: on a plus de noms (300 en plus) et moins de verbes (200 en moins)
Nous garderons quand même notre processing 8, le part of speech étant peu utile pour la suite, mais il était intéressant de nuancer nos études des différents processing que nous avons effectué

N-Gram

Dans cette partie nous allons voir quels sont les unigrams, bigrams et trigrams les plus fréquents dans les données.

```
#Pour faire ce traitement nous avons besoin de réunir tous les abstract et de les word tok
```

```
tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
text_grams = []
for text in df["abstract_fr_proc_8"]:
  tokens = word_tokenize(text)
  print(tokens)
  for token in tokens:
    text_grams.append(token)
```

Updating preview...

```
ontexte', 'avoir', 'conduire', 'enrichir', 'systeme
', 'central', 'alignement', 'automatique', 'celui',
['developpement', 'outil', 'tal', 'dialecte', 'arabe', 'heurte', 'absence', 'resso
['incompletude', 'lexical', 'etre', 'probleme', 'recurrent', 'lorsque', 'cherche',
['dernier', 'decennie', 'accroissement', 'volume', 'donnee', 'avoir', 'rendre', 'd
['segmentation', 'texte', 'unite', 'discursif', 'minimal', 'udm', 'avoir', 'but',
['faiblesse', 'systeme', 'traduction', 'statistique', 'etre', 'caractere', 'ad', '
['explorer', 'maintenir', 'documentation', 'technique', 'etre', 'tache', 'difficil
['article', 'presenter', 'methode', 'segmentation', 'page', 'web', 'bloc', 'texte'
['simplification', 'lexical', 'consiste', 'remplacer', 'mot', 'phrase', 'equivalen
['article', 'presente', 'method', 'generatif', 'prediction', 'structure', 'semanti
['cadre', 'projet', 'asfalda', 'comporte', 'phase', 'annotation', 'semantique', 'f
['devant', 'collection', 'massif', 'heterogene', 'donnee', 'systeme', 'ri', 'devoi
['cadre', 'projet', 'recherche', 'avoir', 'but', 'implementation', 'outil', 'simpl
['analyse', 'syntaxique', 'semantique', 'langages', 'non', 'canonique', 'etre', 'p
['article', 'presente', 'methode', 'avoir', 'objectif', 'minimiser', 'apport', 'ex
['article', 'aborder', 'problematique', 'fonctionnement', 'temporalite', 'langue',
['article', 'presente', 'probleme', 'association', 'entre', 'enoncer', 'langage',
['article', 'propose', 'methode', 'regroupement', 'structure', 'derivation', 'lexi
['papier', 'traiter', 'resume', 'automatique', 'conversation', 'parler', 'spontane
['etude', 'interesser', 'traduction', 'assister', 'ordinateur', 'tao', 'objectif',
['article', 'presenter', 'demarche', 'induction', 'grammaire', 'propriete', 'gp',
['technique', 'actuel', 'traduction', 'automatique', 'permettre', 'produire', 'tra
['article', 'montrer', 'comment', 'utilisation', 'conjoint', 'technique', 'alignem
['presenter', 'projet', 'collaboratif', 'cours', 'mener', 'universite', 'grenoble'
['communaute', 'terminologie', 'etre', 'essentielle', 'car', 'permettre', 'decrire
['baser', 'calcul', 'entropie', 'conditionnel', 'bonami', 'boye', 'paraitre', 'pro
['traitement', 'informatique', 'construction', 'verbe', 'support', 'prendre', 'pho
['sous', 'categorisation', 'argument', 'introduire', 'preposition', 'avoir', 'etre
['outil', 'etiquetage', 'automatique', 'etre', 'plus', 'moins', 'robuste', 'concer
['correction', 'donnee', 'textuel', 'obtenir', 'reconnaissance', 'optique', 'carac
['presenter', 'travail', 'nouveau', 'systeme', 'voyellation', 'automatique', 'text
['article', 'proposer', 'evaluation', 'cadre', 'utilisateur', 'citron', 'systeme',
['none']
['travail', 'presenter', 'approche', 'afin', 'etiqueter', 'large', 'collection', '
['article', 'presente', 'plateforme', 'dedier', 'evaluation', 'difficulte', 'texte
['mot', 'diese', 'hash', 'tags', 'etre', 'moyen', 'naturel', 'lier', 'entre', 'dif
['article', 'aborde', 'question', 'expression', 'attitude', 'affect', 'jugement',
['article', 'presente', 'bibliotheque', 'python', 'appelee', 'kng', 'permettre', '
['objectif', 'etre', 'comparer', 'deux', 'outil', 'analyse', 'corpus', 'texte', 'b

['concordancier', 'jouer', 'depuis', 'longtemps', 'role', 'important', 'analyse',
['presente', 'etude', 'apprentissage', 'viser', 'montrer', 'contexte', 'local', 'c
['presenter', 'etude', 'comparatif', 'impact', 'nature', 'taille', 'corpus', 'appr
['reconnaissance', 'entite', 'nommer', 'ren', 'langue', 'amazigh', 'etre', 'pre',
['article', 'employer', 'topic', 'modeling', 'explorer', 'chemin', 'vers', 'detect
['article', 'propose', 'approche', 'formalisation', 'grammaire', 'langue', 'signe'
['langue', 'signe', 'etre', 'langue', 'naturel', 'utiliser', 'communaute', 'sourd'
['tache', 'resume', 'multi', 'lingue', 'vis', 'concevoir', 'systeme', 'resume', 't
['presenter', 'article', 'adaptation', 'outil', 'resume', 'automatique', 'rezim',
['article', 'aborde', 'probleme', 'extraction', 'donnee', 'oral', 'multi', 'annote
['article', 'propose', 'analyse', 'critique', 'norme', 'timeml', 'lumiere', 'exper
['nombreux', 'information', 'clinique', 'etre', 'contenu', 'texte', 'dossier', 'el
['partir', 'schema', 'annotation', 'dependance', 'syntaxique', 'surface', 'corpus'
```

```
['article', 'presente', 'essai', 'application', 'analyse', 'argumentatif', 'text',
['presenter', 'ensemble', 'exemple', 'lexicographique', 'integrer', 'reseau', 'lex
['tal', 'plus', 'particulierement', 'analyse', 'semantique', 'information', 'coule
['traitement', 'automatique', 'langue', 'ressource', 'lexico', 'semantique', 'avoi
['article', 'presente', 'deux', 'ressource', 'tal', 'distribuer', 'sous', 'licence
```

Updating preview...

```
ms))
bigrams
```

```
[('considerer', 'travail'),
 ('travail', 'tache'),
 ('tache', 'traitement'),
 ('traitement', 'automatique'),
 ('automatique', 'viser'),
 ('viser', 'construire'),
 ('construire', 'partir'),
 ('partir', 'texte'),
 ('texte', 'issu'),
 ('issu', 'corpus'),
 ('corpus', 'constat'),
 ('constat', 'accident'),
 ('accident', 'route'),
 ('route', 'interpretation'),
 ('interpretation', 'compatible'),
 ('compatible', 'dernier'),
 ('dernier', 'proposer'),
 ('proposer', 'illustration'),
 ('illustration', 'sous'),
 ('sous', 'forme'),
 ('forme', 'sequence'),
 ('sequence', 'image'),
 ('image', 'fixe'),
 ('fixe', 'recherche'),
 ('recherche', 'etre'),
 ('etre', 'fruit'),
 ('fruit', 'collaboration'),
 ('collaboration', 'entre'),
 ('entre', 'laboratoire'),
 ('laboratoire', 'universitaire'),
 ('universitaire', 'entreprise'),
 ('entreprise', 'prendre'),
 ('prendre', 'appui'),
 ('appui', 'modele'),
 ('modele', 'grammaire'),
 ('grammaire', 'applicatif'),
 ('applicatif', 'cognitif'),
 ('cognitif', 'vise'),
 ('vise', 'particulier'),
 ('particulier', 'expliquer'),
 ('expliquer', 'certain'),
 ('certain', 'niveau'),
 ('niveau', 'cognitif'),
 ('cognitif', 'transfert'),
 ('transfert', 'entre'),
 ('entre', 'representation'),
 ('representation', 'image'),
 ('image', 'verbal'),
 ('verbal', 'revue'),
```

```
        ('revue', 'question'),
        ('question', 'relatif'),
        ('relatif', 'transcription'),
        ('transcription', 'automatique'),
        ('automatique', 'verbal'),
        ('verbal', 'image'),
        ('image', 'renvoyer'),
```

Updating preview...

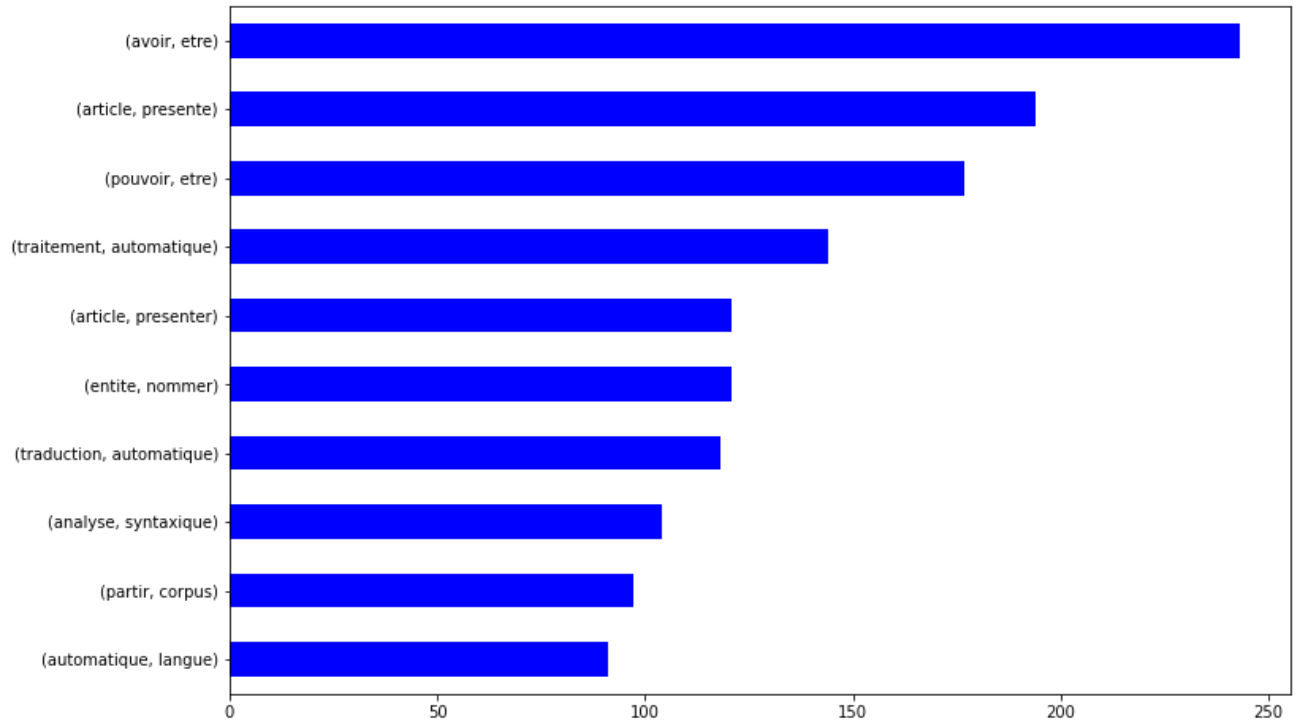
```
),
```

```
bigram_series = pd.Series(bigrams).value_counts()[:10]
bigram_series
```

```
(avoir, etre)               243
(article, presente)         194
(pouvoir, etre)             177
(traitement, automatique)   144
(entite, nommer)            121
(article, presenter)        121
(traduction, automatique)   118
(analyse, syntaxique)       104
(partir, corpus)             97
(automatique, langue)        91
dtype: int64
```

```
bigram_series.sort_values().plot.barh(color='blue', figsize=(12, 8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb67a67c748>
```



Le bigram nous donne comme duo le plus utilisé (avoir et être), sachant que le texte est lemmatizé, il peu s'agir d'un particpe passé ou autre chose.
Ce bigram ne donne que peux d'info : deuxième couple (article et présente) et assez interresant pour le coup
Essayons le trigram


```
#Essayons le trigrams
trigrams = list(nltk.trigrams(text_grams))
trigrams
```

```
trigram_series = pd.Series(trigrams).value_counts()[:10]
trigram_series
```

```
(traitement, automatique, langue)         88
(systeme, question, reponse)              37
(systeme, traduction, automatique)        32
(reconnaissance, entite, nommer)          31
(pouvoir, etre, utiliser)                 25
(traduction, automatique, statistique)    25
(article, presente, methode)              24
(grammaire, arbre, adjoint)               22
(traitement, automatique, langage)        22
(etiquetage, morpho, syntaxique)          22
dtype: int64
```

```
trigram_series.sort_values().plot.barh(color='blue', figsize=(12, 8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb67a608a20>
```

Le trigram est beaucoup plus pertinent en effet toutes lesa ssociations de mots montrent un des sujets bien précis et sont porteurs d'infomration

On remarque de plus que le trigram est moins sensible au lemmmatizer

mots


```
ngrams = list(nltk.ngrams(text_grams,5))
ngrams
```

```
ngram_series = pd.Series(ngrams).value_counts()[:10]
ngram_series
```

```
(none, none, none, none, none)                          7
(extraction, lexique, bilingue, partir, corpus)         6
(traitement, automatique, langue, tal, article)         4
(apprentissage, langue, assister, ordinateur, alao)     3
(traitement, automatique, langue, article, presente)    3
(ameliore, aussi, bien, qualite, alignement)            3
(cadre, projet, intitule, oreodule, systeme)            3
(domaine, traitement, automatique, langage, naturel)    3
(avoir, etre, realiser, cadre, projet)                  3
(traitement, automatique, langue, naturel, taln)        3
dtype: int64
```

```
ngram_series.sort_values().plot.barh(color='blue', figsize=(12, 8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb679eb2780>
```

Le ngrams montrent une lacune du aux fait qu'il y at des corpus vide: en effet l'ocurence qui revient le plus est l'association de none 5 fois
En revanche toutes les autres associations sont très intéressantes puiqu'elles révèlent des

Dirichlet

Dans la dernière partie on procède à une allocation de Dirichlet latente pour mettre en évidence des pacquets de mots que l'on retrouve dans les publications. Pour faire cela on va utiliser la librairie scikit-learn.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import numpy as np


corpus = df[["abstract_fr_proc_8"]]


corpus = corpus["abstract_fr_proc_8"].to_list()


vect = CountVectorizer(max_features=10000, max_df=0.15)
X = vect.fit_transform(corpus)


lda = LatentDirichletAllocation(n_components=10, learning_method="batch", max_iter=25, ran
corpus_topics = lda.fit_transform(X)


print(f"lda.components_.shape {lda.components_.shape}")
```

```
lda.components_.shape (10, 7572)
```

```
sorting = np.argsort(lda.components_, axis=1)[:, ::-1]
features_names = np.array(vect.get_feature_names())
```

On a récupéré cette fonction sur un github pour afficher les pacquets de mots.

```
def print_topics(topics, feature_names, sorting, topics_per_chunk=6,
                 n_words=20):
    for i in range(0, len(topics), topics_per_chunk):
        # for each chunk:
        these_topics = topics[i: i + topics_per_chunk]
        # maybe we have less than topics_per_chunk left
        len_this_chunk = len(these_topics)
        # print topic headers
        print(("topic {:<8}" * len_this_chunk).format(*these_topics))
```

```
        print(("-------- {0:<5}" * len_this_chunk).format(""))
        # print top n_words frequent words
        for i in range(n_words):
            try:
                print(("{:<14}" * len_this_chunk).format(
                    *feature_names[sorting[these_topics, i]]))
```

Updating preview...

```
        print("\n")
```

Dans la représentation d'en-dessous, on remarque une certaine logique dans certains pacquets, comme dans le topic 5, alors que dans d'autres cela est difficilement compréhensible. Un axe d'amélioration aurait été d'utiliser un k-means pour déterminer le nombre de topics pour voir qu'elle saurait la composition des topics.

```
print_topics(topics=range(10), feature_names=features_names,
                          sorting=sorting, topics_per_chunk=5, n_words=10)
```

```
     topic 0       topic 1       topic 2       topic 3       topic 4
     --------      --------      --------      --------      --------
     pourcent      traduction    arabe         conversation  domaine
     phrase        statistique   resume        oral          annotation
     tweet         alignement    annotation    parole        lexique
     performance   bilingue      langage       apprentissage ressource
     reconnaissanceanglais       utilisateur   etude         recherche
     classificationqualite       medical       segmentation  extraction
     analyseur     parallele     etudier       annotation    pourcent
     mesure        phrase        document      erreur        terme
     schema        traduire      regle         baser         outil
     entite        baser         recherche     evaluer       apprentissage


     topic 5       topic 6       topic 7       topic 8       topic 9
     --------      --------      --------      --------      --------
     annotation    dialogue      nan           grammaire     lexical
     relation      grammaire     discours      arbre         relation
     structure     representationarabe         nouveau       question
     representationstructure     objectif      arabe         document
     opinion       formalisme    expression    lexical       sens
     classificationenonce        structure     algorithme    reponse
     forme         analyseur     relation      phrase        ressource
     comme         langage       niveau        analyseur     terme
     certain       dependance    regle         pronom        contexte
     recherche     homme         typologie     derivation    type
```

TF-IDF

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
#Nous allons réutiliser text_grams puisqu'il s'agit de tout nos corpus (les abstract) word
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(text_grams)
feature_names = vectorizer.get_feature_names()
```


Nous allons essayer le modèle k-means qui nécessitait le tf-idf

```
from sklearn.cluster import KMeans
from sklearn import cluster


words = vectorizer.get_feature_names()
kmeans = KMeans(n_clusters = 10, n_init = 17,  max_iter = 200)
#fit the data
kmeans.fit(vectors)
#this loop transforms the numbers back into words
common_words = kmeans.cluster_centers_.argsort()[:,-1:-10:-1]
clusters_words = []
nb_clusters = []
for num, centroid in enumerate(common_words):
    print(str(num) + ' : ' + ', '.join(words[word] for word in centroid))
    clusters_words.append( ', '.join(words[word] for word in centroid))
```

```
df_clusters = pd.DataFrame({ 'clusters_words':clusters_words})
```

```
0 : corpus, article, systeme, automatique, presenter, langue, mot, texte, methode
1 : semantique, zpar, emprunter, encadrement, enchaine, enchainee, enchainement, ench
2 : apprentissage, zpar, encourageant, enchainee, enchainement, enchainer, enchasser,
3 : aide, zpar, encore, enchaine, enchainee, enchainement, enchainer, enchasser, encl
4 : etre, zpar, encontre, encadrement, enchaine, enchainee, enchainement, enchainer,
5 : permettre, zpar, encore, enchaine, enchainee, enchainement, enchainer, enchasser,
6 : erreur, zpar, encore, enchaine, enchainee, enchainement, enchainer, enchasser, er
7 : relation, zpar, encodeur, en, encadrement, enchaine, enchainee, enchainement, enc
8 : analyse, zpar, encourageant, enchainee, enchainement, enchainer, enchasser, encle
9 : avoir, zpar, enchainee, enchainement, enchainer, enchasser, enclencher, enclitiqu
```

On peut remarquer des anomalies dans les cluster (comme le mot zpar)
Mais on peut remarquer aussi que beaucoup de mot de la famille "enchainer" sont présents dans différents clusters, ce mot seul n'ayant que peu se sens ils pourrait être pertinent de les jeter pour la suite
En conclusion le premier cluster est interresant mais tous les autres ont l'air erronés par beaucoup de mots proches (pas dans le sens mais par leur orthographe).
Ce k-means n'est donc pas très concluant

```
df_clusters
```

	clusters_words
0	corpus, article, systeme, automatique, present...
1	semantique, zpar, emprunter, encadrement, ench...
	ant, enchainee, ...
	chainee, encha...
4	etre, zpar, encontre, encadrement, enchaine, e...
5	permettre, zpar, encore, enchaine, enchainee, ...
6	erreur, zpar, encore, enchaine, enchainee, enc...
7	relation, zpar, encodeur, en, encadrement, enc...
8	analyse, zpar, encourageant, enchainee, enchai...
9	avoir, zpar, enchainee, enchainement, enchaine...

Updating preview...