td2a_eco_NLP_tf_idf_ngrams_LDA_word2vec_sur_des_extraits_litterair

June 24, 2022

1 2A.eco - Introduction au text mining

Dans ce TD, nous allons voir comment travailler avec du texte, à partir d'extraits de textes de trois auteurs, Edgar Allan Poe, (EAP), HP Lovecraft (HPL), et Mary Wollstonecraft Shelley (MWS).
Les données sont disponibles ici : spooky.csv.
Le but va être dans un premier temps de regarder dans le détail les termes les plus fréquents utilisés par les auteurs, de les représenter graphiquement puis on va ensuite essayer de prédire quel texte correspond à quel auteur à partir d'un modèle Word2Vec. Ce notebook librement inspiré de :

- https://www.kaggle.com/enerrio/scary-nlp-with-spacy-and-keras
- https://github.com/GU4243-ADS/spring2018-project1-ginnyqg
- https://www.kaggle.com/meiyizi/spooky-nlp-and-topic-modelling-tutorial/notebook

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

Parmi les concepts / packages que nous allons voir : - WordCloud - nltk - Keras - spacy

[2]:
```
%matplotlib inline
```

1.1 Les packages qu'on utilisera

[3]:
```
#! pip install wordcloud
#!pip install gensim
#!pip install pywaffle
#!pip install keras
#!pip install tensorflow
```

[4]:
```
#import nltk
#nltk.download('stopwords')
#nltk.download('punkt')
#nltk.download('genesis')
#nltk.download('wordnet')
```

[5]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
```

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from IPython.display import display
import base64
import string
import re
from collections import Counter
from time import time
# from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS as stopwords
from sklearn.metrics import log_loss
import matplotlib.pyplot as plt
from pywaffle import Waffle

from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF, LatentDirichletAllocation
```

[6]:
```
import nltk

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.collocations import *
try:
    stopwords = set(stopwords.words('english'))
except LookupError:
    import nltk
    nltk.download('stopwords')
    stopwords = set(stopwords.words('english'))
#stopwords
```

[7]:
```
import os
if not os.path.exists('spooky.csv'):
    from pyensae.datasource import download_data
    download_data('spooky.csv',
                  url='https://raw.githubusercontent.com/GU4243-ADS/
 ↪spring2018-project1-ginnyqg/master/data/')
```

1.2 Import des données

[8]:
```
import pandas as pd
train = pd.read_csv('spooky.csv',delimiter = ",",
                    skiprows=1, names=['ID', 'Text', 'Author'],
                    encoding='latin-1' ).set_index('ID')
```

[9]:
```
train.head()
```

[9]:

ID	Text	Author
id26305	This process, however, afforded me no means of…	EAP
id17569	It never once occurred to me that the fumbling…	HPL
id11008	In his left hand was a gold snuff box, from wh…	EAP
id27763	How lovely is spring As we looked from Windsor…	MWS
id12958	Finding nothing else, not even gold, the Super…	HPL

[10]:
```
# Delete the word 'id' from the ID columns
train.index = [id[2:] for id in train.index]
train.head()
```

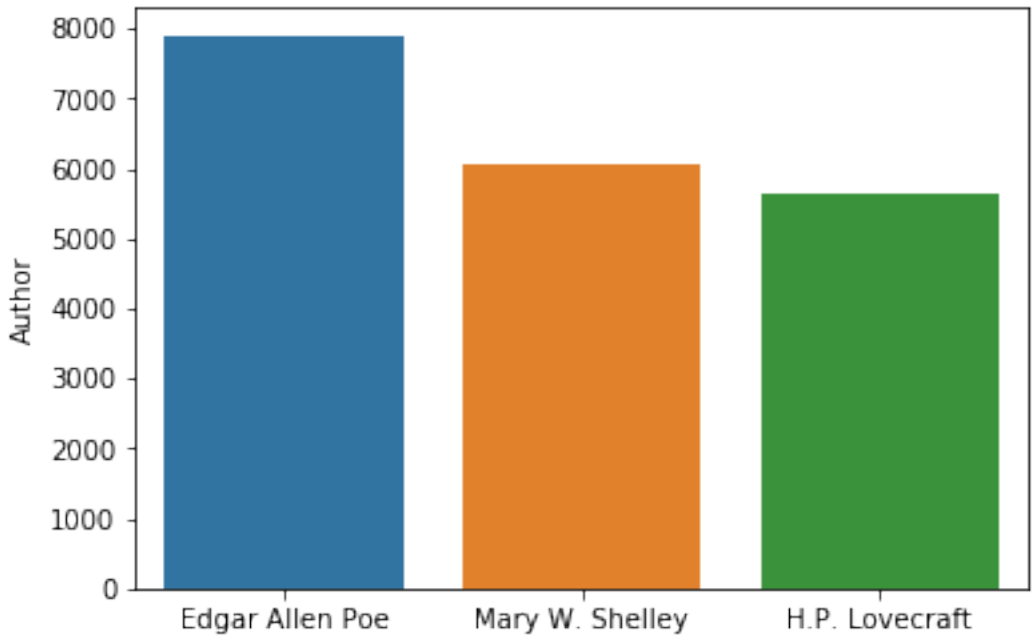
[10]:

	Text	Author
26305	This process, however, afforded me no means of…	EAP
17569	It never once occurred to me that the fumbling…	HPL
11008	In his left hand was a gold snuff box, from wh…	EAP
27763	How lovely is spring As we looked from Windsor…	MWS
12958	Finding nothing else, not even gold, the Super…	HPL

[11]:
```
print('Training sample:', train['Text'][0])
print('Author of sample:', train['Author'][0])
print('Training Data Shape:', train.shape)
```

```
Training sample: This process, however, afforded me no means of ascertaining the
dimensions of my dungeon; as I might make its circuit, and return to the point
whence I set out, without being aware of the fact; so perfectly uniform seemed
the wall.
Author of sample: EAP
Training Data Shape: (19579, 2)
```

[12]:
```
# Barplot of occurances of each author in the training dataset
sns.barplot(x=['Edgar Allen Poe', 'Mary W. Shelley', 'H.P. Lovecraft'],␣
 ↪y=train['Author'].value_counts());
```



On se rend compte que les extraits des 3 auteurs ne sont pas forcément équilibrés dans le jeu de données. Il faudra en tenir compte dans la prédiction.

1.3 Bag of Words, IF TDF

L'approche bag of words : on ne tient pas compte de l'ordre des mots, ni du contexte dans lequel ils interviennent (ou alors de manière très partielle, en étudiant par exemple le mot suivant).
L'idée est d'étudier la fréquence des mots d'un document et la surreprésentation des mots par rapport à un document de référence (appelé corpus). Cette approche un peu simpliste mais très efficace : on peut calculer des scores permettant par exemple de faire de classification automatique de document par thème, de comparer la similarité de deux documents. Elle est souvent utilisée en première analyse, et elle reste la référence pour l'analyse de textes mal structurés (tweets, dialogue tchat, etc.) Mot-clés : td-idf, indice de similarité cosine

1.3.1 Fréquence d'un mot

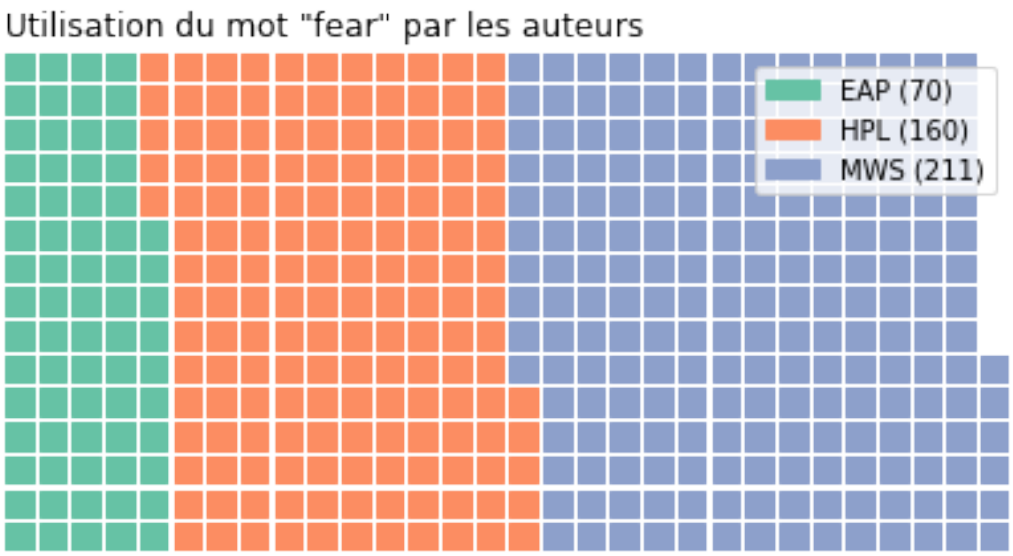
On débute par une approche très simple, compter le nombre d'occurence d'un mot qui nous intéresse, ici on prend l'exemple du mot le "fear"
Et on utilise au passage un package assez sympa PyWaffle

[13]:
```
train['word-fear'] = train['Text'].str.contains('fear').astype(int)
table = pd.pivot_table(train, values='word-fear', index=['Author'], aggfunc=np.sum)
data = table.to_dict()['word-fear']
data
```

[13]:
```
{'EAP': 70, 'HPL': 160, 'MWS': 211}
```

[14]:
```
fig = plt.figure(
    FigureClass=Waffle,
    rows=15,
    values=data,
    title={'label': 'Utilisation du mot "fear" par les auteurs', 'loc': 'left'},
    labels=["{0} ({1})".format(k, v) for k, v in data.items()],
);
```

```
c:\python372_x64\lib\site-packages\matplotlib\figure.py:2369: UserWarning: This
figure includes Axes that are not compatible with tight_layout, so results might
be incorrect.
  warnings.warn("This figure includes Axes that are not compatible "
```



1.3.2 WordCloud - premier exemple

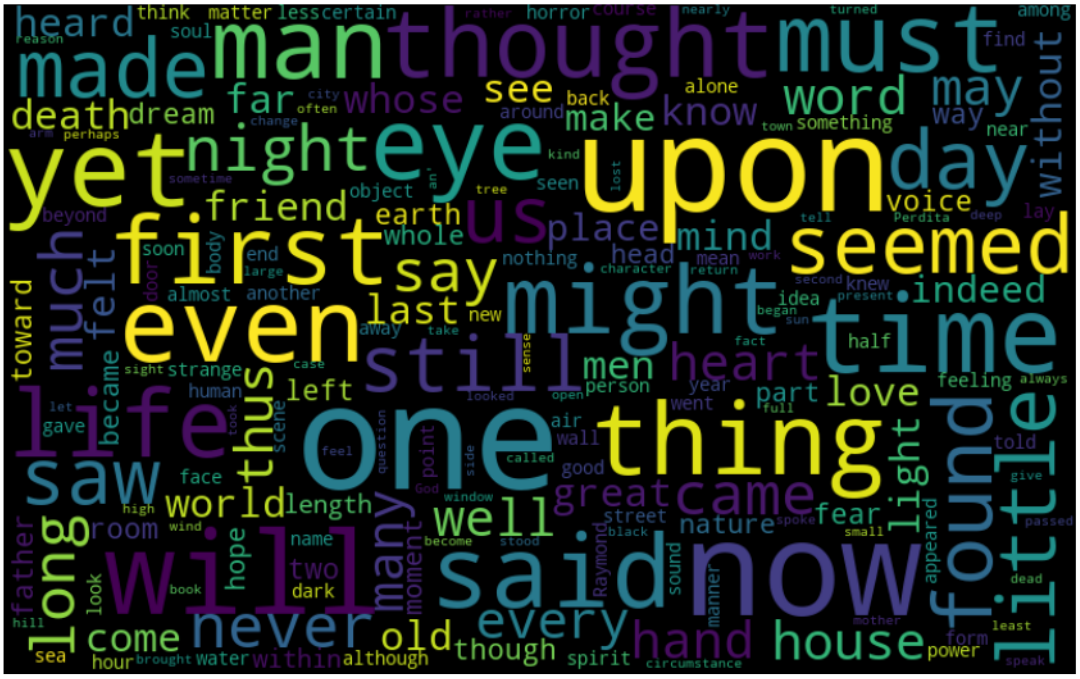
Ici, on va un cran plus loin, on prend tous les mots des textes et on cherche à représenter la fréquence des mots utilisés par les auteurs.
Pour cela, on va utiliser le package WordCloud - nuage de mots - qui prend en entrée une liste de mots.
Ce package peut être très largement personnalisé, par exemple en changeant la police (par défaut, Droid-SansMono)

[15]:
```
# Combine all training text into one large string
all_text = ' '.join([text for text in train['Text']])
print('Number of words in all_text:', len(all_text))
```

```
Number of words in all_text: 2938231
```

[16]:
```
# Word cloud for entire training dataset
# default width=400, height=200

wordcloud = WordCloud(width=800, height=500,
                      random_state=21, max_font_size=110).generate(all_text)
plt.figure(figsize=(15, 12))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off');
```



Si vous voulez changer la police, il faudra écrire cela ainsi :

```
wordcloud = WordCloud(font_path = "DoubleFeature21.ttf", width=800, height=500,
                      random_state=21, max_font_size=110).generate(all_text)
```

1.3.3 Fréquence des mots

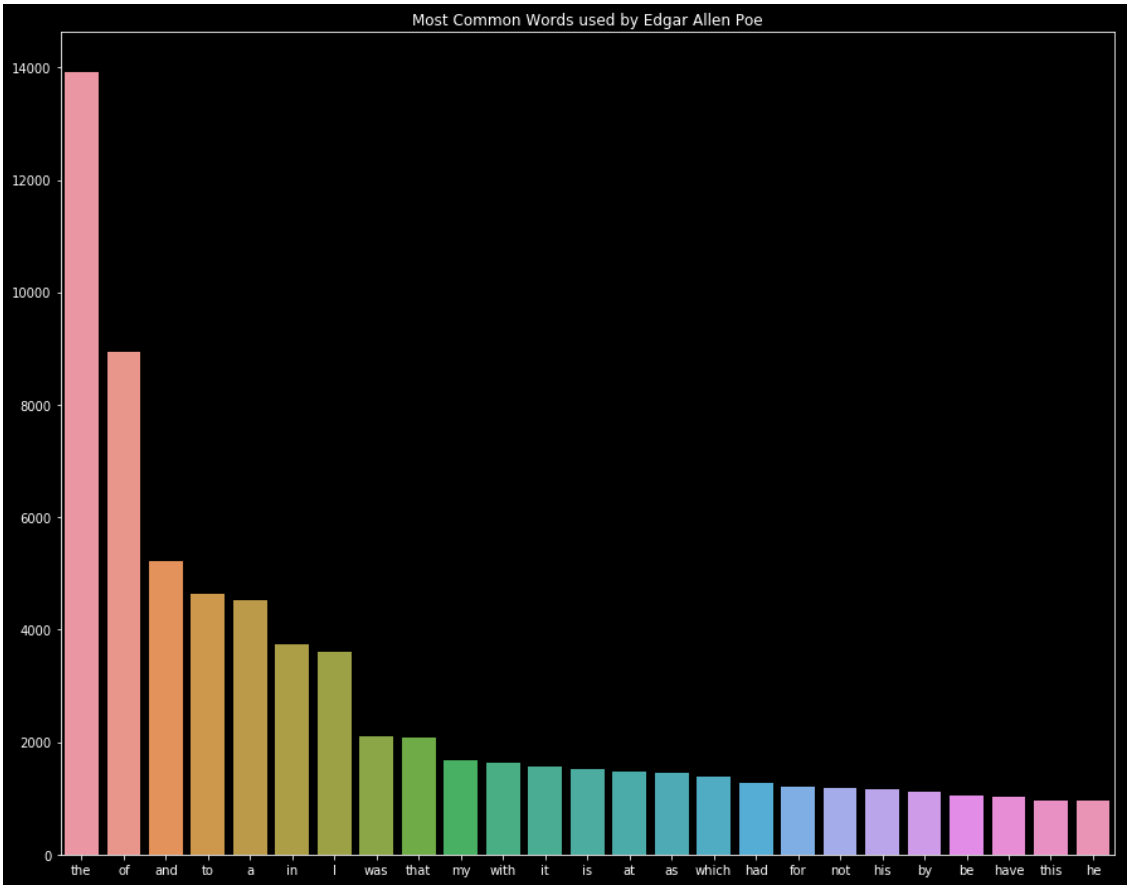
Une autre manière de voir la même chose, un histogramme avec le compte des mots les plus fréquents

[17]:
```
eap = train[train['Author'] == 'EAP']
eap_text = ' '.join(text for text in eap['Text'])
```

[18]:
```
eap_list = eap_text.split()
eap_counts = Counter(eap_list)
eap_common_words = [word[0] for word in eap_counts.most_common(25)]
eap_common_counts = [word[1] for word in eap_counts.most_common(25)]

plt.style.use('dark_background')
plt.figure(figsize=(15, 12))

sns.barplot(x=eap_common_words, y=eap_common_counts)
plt.title('Most Common Words used by Edgar Allen Poe');
```



Pas très instructif.... On a beaucoup de mots “commun”, comme “the”, “of”, ”my qui ne nous intéressent pas. On va donc devoir nettoyer les textes.

1.3.4 Nettoyage du texte, suppression de la ponctuation et des stopwords

Pour “nettoyer” le texte, on utilise les fonctions tokenizer de nltk.
C’est la tâche consistant à décomposer une chaîne de caractères en unités lexicales, aussi appelées tokens. Pour un ordinateur, une chaine de caractère est juste une suite de symboles, il n’est pas capable seul de déterminer quels sont les mots d’une phrase ; il n’y voit qu’une chaîne de caractères. Un processus de tokenization consiste donc à séparer ces mots, selon les espaces.

[19]:
```
eap_word_list = word_tokenize(eap_text)
eap_word_list = [x.lower() for x in eap_word_list]

# ensuite, on enlève les stopswords qui sont les mots comme if, then, the, and ...␣
 ↪ainsi que la ponctuation

eap_clean = [w.lower() for w in eap_word_list if w not in stopwords and w.isalpha()]
eap_clean = ' '.join(text.lower() for text in eap_clean)
```

[20]:
```
eap_clean[:200] + "..."
```

[20]:
```
'process however afforded means ascertaining dimensions dungeon might make
circuit return point whence set without aware fact perfectly uniform seemed wall
```

```
left hand gold snuff box capered hill cutting…'
```

Avec ce petit nettoyage, on arrive à un texte plus “pur” et plus intéressant en termes d’analyse lexicale

1.3.5 Question 1)

Reproduisez le wordcloud et le graphique des fréquences sur la base du texte nettoyé de Allan Edgar Poe, que remarquez-vous ?

1.3.6 Tf-IDF - le calcul de fréquence

Le calcul tf-idf (term frequency–inverse document frequency) permet de calculer un score de proximité entre un terme de recherche et un document (c’est ce que font les moteurs de recherche). La partie *tf* calcule une fonction croissante de la fréquence du terme de recherche dans le document à l’étude, la partie idf calcule une fonction inversement proportionnelle à la fréquence du terme dans l’ensemble des documents (ou corpus). Le score total, obtenu en multipliant les deux composantes, permet ainsi de donner un score d’autant plus élevé que le terme est surreprésenté dans un document (par rapport à l’ensemble des documents). Il existe plusieurs fonctions, qui pénalisent plus ou moins les documents longs, ou qui sont plus ou moins smooth.

[21]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words=stopwords)
tfs = tfidf.fit_transform(train['Text'])
```

[22]:
```
list(tfidf.vocabulary_.keys())[:10] + ['...']
```

[22]:
```
['process',
 'however',
 'afforded',
 'means',
 'ascertaining',
 'dimensions',
 'dungeon',
 'might',
 'make',
 'circuit',
 '…']
```

[23]:
```
feature_names = tfidf.get_feature_names()
corpus_index = [n for n in list(tfidf.vocabulary_.keys())]
import pandas as pd
df = pd.DataFrame(tfs.todense(), columns=feature_names)
#print(df)
```

[24]:
```
df.head()
```

[24]:

	aaem	ab	aback	abaft	abandon	abandoned	abandoning	abandonment	\
0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.253506	0.0	0.0	

	abaout	abased	…	zodiacal	zoilus	zokkar	zone	zones	zopyrus	zorry	\
0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

```
2     0.0     0.0  …       0.0     0.0     0.0   0.0    0.0      0.0    0.0
3     0.0     0.0  …       0.0     0.0     0.0   0.0    0.0      0.0    0.0
4     0.0     0.0  …       0.0     0.0     0.0   0.0    0.0      0.0    0.0

   zubmizzion  zuro   á¼
0         0.0   0.0  0.0
1         0.0   0.0  0.0
2         0.0   0.0  0.0
3         0.0   0.0  0.0
4         0.0   0.0  0.0

[5 rows x 24937 columns]
```

Si on repart de notre mot “fear”, on va chercher les extraits où le score TF-IDF est le plus élévé.

[25]:
```
for x in df["fear"].sort_values(ascending =False).head(n=10).index :
    print(train.iloc[x]["Text"])
```

```
We could not fear we did not.
"And now I do not fear death.
Be of heart and fear nothing.
I smiled, for what had I to fear?
Indeed I had no fear on her account.
I have not the slightest fear for the result.
At length, in an abrupt manner she asked, "Where is he?" "O, fear not," she
continued, "fear not that I should entertain hope Yet tell me, have you found
him?
"I fear you are right there," said the Prefect.
I went down to open it with a light heart, for what had I now to fear?
Do I fear, that my heart palpitates?
```

On remarque que les scores les plus élévés sont soient des extraits courts où le mot apparait une seule fois, et des extraits plus longs où le mot fear apprait plusieurs fois.

1.3.7 Question 2)

Qu’en est-il pour le mot “love” ?

1.4 Approche contextuelle : les n-gramms

L’approche contextuelle : on s’intéresse non seulement aux mots et à leur fréquence, mais aussi aux mots qui suivent. Cette approche est essentielle pour désambiguiser les homonymes. Elle permet aussi d’affiner les modèles “bag-of-words”. Le calcul de n-grams (bigrams pour les co-occurences de mots deux-à-deux, tri-grams pour les co-occurences trois-à-trois, etc.) constitue la méthode la plus simple pour tenir compte du contexte.
nltk offre des methodes pour tenir compte du contexte : pour ce faire, nous calculons les n-grams, c’est-à-dire l’ensemble des co-occurrences successives de mots deux-à-deux (bigrams), trois-à-trois (tri-grams), etc.
En général, on se contente de bi-grams, au mieux de tri-grams :

- les modèles de classification, analyse du sentiment, comparaison de documents, etc. qui comparent des n-grams avec n trop grands sont rapidement confrontés au problème de données sparse, cela réduit la capacité prédictive des modèles ;
- les performances décroissent très rapidement en fonction de n, et les coûts de stockage des données augmentent rapidement (environ n fois plus élevé que la base de donnée initiale).

1.4.1 Contexte du mot

[26]:
```
print("Nombre de caractères : ",len(eap_clean))
print('\n')

#Tokenisation naïve sur les espaces entre les mots => on obtient une liste de mots
tokens = eap_clean.split()

#On transforme cette liste en objet nltk "Text" (objet chaine de caractère qui␣
 ↪conserve la notion de tokens, et qui
#comprend  un certain nombre de méthodes utiles pour explorer les données.

text = nltk.Text(tokens)

#Comme par exemple "concordance" : montre les occurences d'un mot dans son contexte
print("Exemples d'occurences du terme 'fear' :")
text.concordance("fear")
print('\n')
```

```
Nombre de caractères :  696985


Exemples d'occurences du terme 'fear' :
Displaying 24 of 24 matches:
mitted suicide went open light heart fear reason priori diddle would diddle wi
loud quick unequal spoken apparently fear well anger said unintelligible words
 geneva seemed resolved give scruple fear wind night rain fell falling rain fa
ick darkness shutters close fastened fear robbers knew could see opening door
y ancient modes investigation smiled fear say mean average interval entire rew
vision need go details even went far fear occasioned much trouble might glad c
ed strong relish physical philosophy fear tinctured mind common error age mean
er started hourly dreams unutterable fear find hot breath thing upon face vast
lady seventy years age heard express fear never see marie observation attracte
d propeller must entirely remodelled fear serious accident mean steel rod vane
lliam legrand second place impressed fear indeed impossible make comprehended
t move question oinos freely without fear adaptation eyes behold earth adelaid
 given scream took arm attended home fear shall never see marie imagine madame
mentor came watched minutes somewhat fear wonder duc slips card horrible myste
unbent freely conclusion absurd much fear replied monsieur maillard becoming e
ind dreaded whip instantly converted fear summing seemed evident reasoner even
 spades whole insisted upon carrying fear seemed trusting either implements wi
 little doubt ultimately seeing pole fear right said prefect well talk facilis
urred occurred never occurred indeed fear account moment wild lurid light alon
nsieur maillard perhaps said legrand fear artist firm intention cut shop since
e three four smart raps hammer heart fear nothing daughter mademoiselle moissa
nt open reality thin replied entered fear unusual horror thing evident nucleus
tly inappropriate splendor slightest fear result face far turned toward stage
ervously around barriers iron hemmed fear mesmerized adding immediately afterw
```

1.4.2 Co-occurences - version 1

Analyse de la fréquence des termes d'intérêt :

[27]:
```
import io
f = io.StringIO()
try:
    text.collocations()
    print("Co-occurences fréquentes :")
    print("\n".join(f.getvalue().split("\n")[:3] + ["..."]))
except ValueError as e:
    print("Erreur...")
    print(e)
```

```
Erreur…
too many values to unpack (expected 2)
```

Si ces mots sont très fortement associés, les expressions sont également peu fréquentes. Il est donc parfois nécessaire d'appliquer des filtres, par exemple ignorer les bigrammes qui apparaissent moins de 5 fois dans le corpus.

1.4.3 Co-occurences - version 2

avec le code ci dessous, on peut choisir un peu mieux ce qui nous intéresse.

[28]:
```
finder = nltk.BigramCollocationFinder.from_words(text)

finder.apply_freq_filter(5)

bigram_measures = nltk.collocations.BigramAssocMeasures()

collocations = finder.nbest(bigram_measures.jaccard, 15)

for collocation in collocations:
    c = ' '.join(collocation)
    print(c)
```

```
mein gott
ourang outang
hans pfaall
brevet brigadier
du roule
bas bleu
ugh ugh
tea pot
gum elastic
hu hu
prodigies valor
ha ha
mille mille
chess player
drum dow
```

1.4.4 Co-occurences avec un mot spécifique

Ici, on va chercher quelques sont les termes fréquemment associés avec le mot "fear"

[29]:
```
try:
    nltk.corpus.genesis.words('english-web.txt')
```

```
except LookupError:
    import nltk
    nltk.download('genesis')
    nltk.corpus.genesis.words('english-web.txt')
```

[30]:
```
bigram_measures = nltk.collocations.BigramAssocMeasures()

# Ngrams with a specific name
name_filter = lambda *w: 'fear' not in w


## Bigrams
finder = BigramCollocationFinder.from_words(
            nltk.corpus.genesis.words('english-web.txt'))

# only bigrams that contain 'fear'
finder.apply_ngram_filter(name_filter)

# return the 100 n-grams with the highest PMI
print(finder.nbest(bigram_measures.likelihood_ratio,100))
```

```
[('fear', 'of'), ('fear', 'God'), ('I', 'fear'), ('the', 'fear'), ('The',
'fear'), ('fear', 'him'), ('you', 'fear')]
```

1.4.5 Question 3)

Retouvez les bi-gramms les plus fréquents avec le mot “love”

1.5 LDA

Le modèle Latent Dirichlet Allocation (LDA) est un modèle probabiliste génératif qui permet de décrire des collections de documents de texte ou d’autres types de données discrètes. LDA fait partie d’une catégorie de modèles appelés “topic models”, qui cherchent à découvrir des structures thématiques cachées dans des vastes archives de documents.
Ceci permet d’obtenir des méthodes efficaces pour le traitement et l’organisation des documents de ces archives: organisation automatique des documents par sujet, recherche, compréhension et analyse du texte, ou même résumer des textes.
Aujourd’hui, ce genre de méthodes s’utilisent fréquemment dans le web, par exemple pour analyser des ensemble d’articles d’actualité, les regrouper par sujet, faire de la recommendation d’articles, etc.

1.5.1 Spécification du modèle

[31]:
```
try:
    nltk.download('wordnet')
except LookupError:
    import nltk
    nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\xavie\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[32]:
```
lemm = WordNetLemmatizer()

# un exemple de ce que la notion de lemmatize veut dire

print("The lemmatized form of leaves is: {}".format(lemm.lemmatize("women")))
print("The lemmatized form of leaves is: {}".format(lemm.lemmatize("daughters")))
print("The lemmatized form of leaves is: {}".format(lemm.lemmatize("leaves")))
```

```
The lemmatized form of leaves is: woman
The lemmatized form of leaves is: daughter
The lemmatized form of leaves is: leaf
```

[33]:
```
lemm = WordNetLemmatizer()

class LemmaCountVectorizer(CountVectorizer):
    def build_analyzer(self):
        analyzer = super(LemmaCountVectorizer, self).build_analyzer()
        return lambda doc: (lemm.lemmatize(w) for w in analyzer(doc))
```

[34]:
```
# Storing the entire training text in a list

text = list(train.Text.values)

# Calling our overwritten Count vectorizer
tf_vectorizer = LemmaCountVectorizer(max_df=0.95,
                                     min_df=2,
                                     stop_words='english',
                                     decode_error='ignore')
tf = tf_vectorizer.fit_transform(text)
```

[35]:
```
lda = LatentDirichletAllocation(n_components=11, max_iter=5,
                                learning_method = 'online',
                                learning_offset = 50.,
                                random_state = 0)
```

[36]:
```
lda.fit(tf)
```

[36]:
```
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
                          evaluate_every=-1, learning_decay=0.7,
                          learning_method='online', learning_offset=50.0,
                          max_doc_update_iter=100, max_iter=5,
                          mean_change_tol=0.001, n_components=11, n_jobs=None,
                          perp_tol=0.1, random_state=0, topic_word_prior=None,
                          total_samples=1000000.0, verbose=0)
```

1.5.2 Représentation des résultats

[37]:
```
def print_top_words(model, feature_names, n_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print("Topic #%d:" % topic_idx)
        print(" ".join([feature_names[i]
                        for i in topic.argsort()[:-n_top_words - 1:-1]]))
```

```
    print()
```

[38]:
```
n_top_words = 40
print("\nTopics in LDA model: ")
tf_feature_names = tf_vectorizer.get_feature_names()
print_top_words(lda, tf_feature_names, n_top_words)
```

```
Topics in LDA model:
Topic #0:
knew near raymond did hope death morning stood eye think little ground read
attention know felt sense say creature sat gave table bed misery pleasure heart
like till lay loved heavy high spirit desire rock hour father light presence
lord
Topic #1:
make mr matter mean point fact called return present set person certain said
reason case question result impossible self really doubt knowledge force
ordinary merely altogether real duty discovery principle purpose course quite
particular having second term perceive proportion machine
Topic #2:
left way place sea half far thing wall window ancient like possible land hill
taken wood water strange black street motion right small world great short box
long order little received spot having took city river shape cut said wind
Topic #3:
night thought house old earth man soon away young length men great god foot room
beauty town fear people degree beneath oh remained know seen lady country happy
replied street held long passion idea number woman come high certain home
Topic #4:
time dream did said long year friend went father love day men family speak child
dear poor old happiness care came sure idris passed place death mother strange
sister remember world girl account silence visit little company public state
companion
Topic #5:
thing man say life manner said moon wild little secret like death great age
appearance kind space mind given did year change tell word west nature observed
just ye general come believe gave thrown power air better regard sir resolved
Topic #6:
saw door stone nearly perdita large terrible sleep tear thousand wish room
surface instant later seen condition eye tried miserable caused elizabeth die
position length mystery chair dark nose view succeeded probably known measure
marked physical complete forced remembered affair
Topic #7:
eye came head let horror open suddenly deep form face hand close come arm
appeared terror hideous mere gone lip wonder light vision grave new figure lost
countenance darkness beheld sound escape hair fall shadow influence extreme
chamber effort mouth
Topic #8:
shall good day feel sight sun soul look longer nature star evening grew world
idea imagination work cloud cast arose study mind heart sky threw arrived true
evil said melancholy continued glance hand situation clear feeling relief
approached city lake
Topic #9:
heard word moment life body voice time dead hand character fell like heart
```

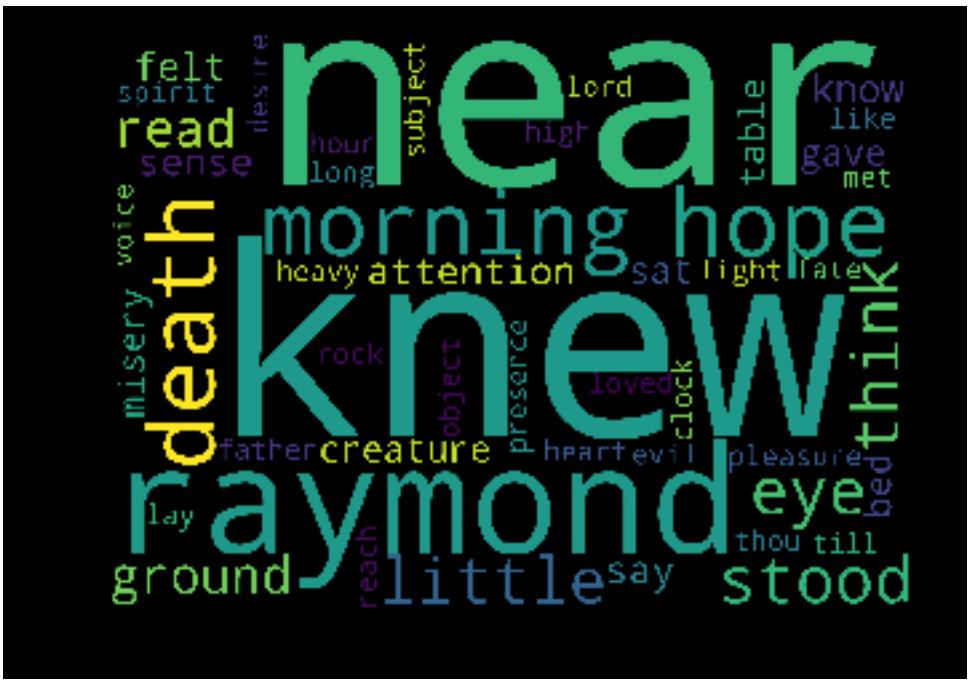
```
turned adrian day felt best reached gentle friend ear rest sound tree necessary
live blood wind road ship water tone hour servant closed seen sympathy hung bear
Topic #10:
fear memory month leave week course danger enemy hardly wished observation
article le glimpse attended charge truly magnificent engaged beast wealth ve sit
locked bob toil interrupted curiously oil universal la motive luxury et solid
perpetual plan cruel boat pull
```

[39]:
```
first_topic = lda.components_[0]
second_topic = lda.components_[1]
```

[40]:
```
first_topic_words = [tf_feature_names[i] for i in first_topic.argsort()[:-50 - 1 :-1]]
second_topic_words = [tf_feature_names[i] for i in second_topic.argsort()[:-50 - 1 :
↪-1]]
```

Generating the wordcloud with the values under the category dataframe...

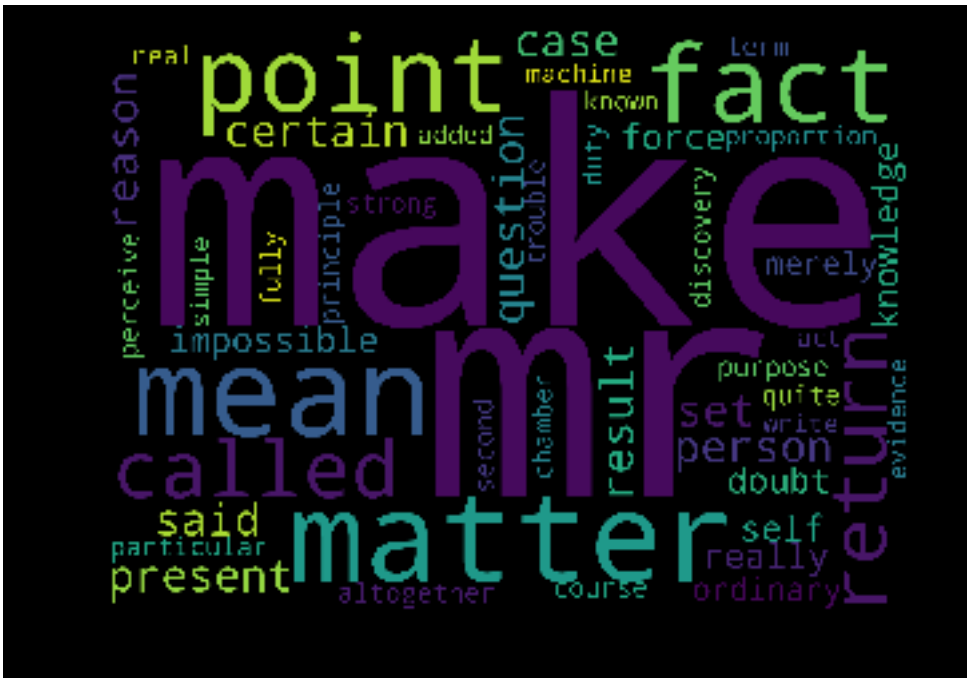
[41]:
```
firstcloud = WordCloud(stopwords=stopwords,
                       background_color='black',
                       width=2500,
                       height=1800
                       ).generate(" ".join(first_topic_words))
plt.imshow(firstcloud)
plt.axis('off');
```



Generating the wordcloud with the values under the category dataframe...

[42]:
```
second_topic_words = [tf_feature_names[i] for i in second_topic.argsort()[:-50 - 1 :
↪-1]]
```

```
second_cloud = WordCloud(
                          stopwords=stopwords,
                          background_color='black',
                          width=2500,
                          height=1800
                         ).generate(" ".join(second_topic_words))
plt.imshow(second_cloud)
plt.axis('off');
```



Dernière façon de visualiser les résultats avec le module pyldavis.

[43]:
```
import pyLDAvis
import pyLDAvis.sklearn
pyLDAvis.enable_notebook()
```

C'est assez long...

[44]:
```
pyLDAvis.sklearn.prepare(lda, tf, tf_vectorizer)
```

```
c:\python372_x64\lib\site-packages\pyLDAvis\_prepare.py:257: FutureWarning:
Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  return pd.concat([default_term_info] + list(topic_dfs))
```

[44]:
```
PreparedData(topic_coordinates=              x         y  topics  cluster
Freq
topic
3      0.083418  0.093247       1        1  13.966553
5      0.125122  0.097281       2        1  11.190887
2      0.070879  0.153648       3        1  11.166555
0      0.140329  0.071879       4        1  10.935812
9      0.112714 -0.161896       5        1   9.971051
4      0.149845 -0.091162       6        1   9.355816
7     -0.074913  0.002756       7        1   8.767889
1     -0.127920  0.087840       8        1   7.801239
8     -0.035444 -0.294589       9        1   7.197809
6     -0.214033  0.088394      10        1   6.324431
10    -0.229997 -0.047397      11        1   3.321958, topic_info=      Category
Freq    Term       Total  loglift  logprob
12320  Default  821.000000    time  821.000000  30.0000  30.0000
10599  Default  488.000000     saw  488.000000  29.0000  29.0000
12223  Default  688.000000   thing  688.000000  28.0000  28.0000
7456   Default  743.000000     man  743.000000  27.0000  27.0000
8153   Default  547.000000   night  547.000000  26.0000  26.0000
…        …          …      …          …       …       …
7916   Topic11   25.666850  motive   26.520360   3.3719  -5.6375
7369   Topic11   24.676136  luxury   25.529647   3.3706  -5.6769
7866   Topic11  121.532274   month  139.424660   3.2673  -4.0826
4687   Topic11  140.643108    fear  287.716489   2.6889  -3.9365
2666   Topic11   91.508401  course  208.720573   2.5801  -4.3663

[539 rows x 6 columns], token_table=       Topic      Freq      Term
term
23         6  0.982310      able
73         8  0.984806  accident
162        8  0.989697     added
218        5  0.996050    adrian
255       10  0.972772    affair
…      …        …       …
13741      2  0.260289      year
13741      5  0.083579      year
13741      6  0.477594      year
13752      2  0.995314       yes
13766      1  0.996449     young

[728 rows x 3 columns], R=30, lambda_step=0.01, plot_opts={'xlab': 'PC1',
'ylab': 'PC2'}, topic_order=[4, 6, 3, 1, 10, 5, 8, 2, 9, 7, 11])
```

1.6 Prédiction - Modélisation en utilisant Word2vec et Keras

Derniere partie du TD, prédire les auteurs à partir des textes qu'on aura dans la base de test.
Pour cela, on utilise les modules spacy, Word2vec et Keras de Python. Le but ici n'est pas de connaitre par coeur toutes les fonctions utilisées, mais de comprendre comment le code fonctionne, afin de pouvoir vous en reservir si vous en avez besoin.

[45]:
```
try:
    import spacy
    nlp = spacy.load('en_core_web_sm')
```

```
except Exception as e:
    # Un message d'erreur peut apparaître.
    # cymem.cymem.Pool has the wrong size, try recompiling. Expected 64, got 48
    # Sous Windows:
    # pip install http://www.xavierdupre.fr/enseignement/setup/cymem-1.31.
 ↪2-cp37-cp37m-win_amd64.whl
    # pip install http://www.xavierdupre.fr/enseignement/setup/thinc-6.10.
 ↪2-cp37-cp37m-win_amd64.whl
    # pip install http://www.xavierdupre.fr/enseignement/setup/spacy-2.0.
 ↪12-cp37-cp37m-win_amd64.whl
    print(e)
```

```
c:\python372_x64\lib\site-packages\thinc\neural\train.py:7: DeprecationWarning:
Using or importing the ABCs from 'collections' instead of from 'collections.abc'
is deprecated, and in 3.8 it will stop working
  from .optimizers import Adam, linear_decay
c:\python372_x64\lib\site-packages\thinc\check.py:4: DeprecationWarning: Using
or importing the ABCs from 'collections' instead of from 'collections.abc' is
deprecated, and in 3.8 it will stop working
  from collections import Sequence, Sized, Iterable, Callable
c:\python372_x64\lib\site-packages\thinc\check.py:4: DeprecationWarning: Using
or importing the ABCs from 'collections' instead of from 'collections.abc' is
deprecated, and in 3.8 it will stop working
  from collections import Sequence, Sized, Iterable, Callable
```

1.6.1 Mise en forme des données

On nettoye rapidement le texte sur la base entière, en supprime la ponctuation, les stopwords, avec une autre méthode que celle vue plus haut

[46]:
```
punctuations = string.punctuation

# Define function to cleanup text by removing personal pronouns, stopwords, and␣
 ↪puncuation
def cleanup_text(docs, logging=False):
    texts = []
    counter = 1
    for doc in docs:
        if counter % 1000 == 0 and logging:
            print("Processed %d out of %d documents." % (counter, len(docs)))
        counter += 1
        doc = nlp(doc, disable=['parser', 'ner'])
        tokens = [tok.lemma_.lower().strip() for tok in doc if tok.lemma_ != '-PRON-']
        tokens = [tok for tok in tokens if tok not in stopwords and tok not in␣
 ↪punctuations]
        tokens = ' '.join(tokens)
        texts.append(tokens)
    return pd.Series(texts)
```

[47]:
```
# Cleanup text and make sure it retains original shape
print('Original training data shape: ', train['Text'].shape)
train_cleaned = cleanup_text(train['Text'], logging=True)
print('Cleaned up training data shape: ', train_cleaned.shape)
```

```
Original training data shape:  (19579,)
Processed 1000 out of 19579 documents.
Processed 2000 out of 19579 documents.
Processed 3000 out of 19579 documents.
Processed 4000 out of 19579 documents.
Processed 5000 out of 19579 documents.
Processed 6000 out of 19579 documents.
Processed 7000 out of 19579 documents.
Processed 8000 out of 19579 documents.
Processed 9000 out of 19579 documents.
Processed 10000 out of 19579 documents.
Processed 11000 out of 19579 documents.
Processed 12000 out of 19579 documents.
Processed 13000 out of 19579 documents.
Processed 14000 out of 19579 documents.
Processed 15000 out of 19579 documents.
Processed 16000 out of 19579 documents.
Processed 17000 out of 19579 documents.
Processed 18000 out of 19579 documents.
Processed 19000 out of 19579 documents.
Cleaned up training data shape:  (19579,)
```

On “parse”, c’est à dire qu’on parcourt, tous les textes pour les mettre en forme avant de passer dans le modèle

[48]:
```
# Parse documents and print some info
print('Parsing documents...')

start = time()

train_vec = []
for doc in nlp.pipe(train_cleaned, batch_size=500):
    if doc.has_vector:
        train_vec.append(doc.vector)
    # If doc doesn't have a vector, then fill it with zeros.
    else:
        train_vec.append(np.zeros((128,), dtype="float32"))

train_vec = np.array(train_vec)

end = time()
```

```
Parsing documents…
```

[49]:
```
print('Total time passed parsing documents: {} seconds'.format(end - start))
print('Total number of documents parsed: {}'.format(len(train_vec)))
print('Number of words in first document: ', len(train['Text'][0]))
print('Number of words in second document: ', len(train['Text'][1]))
print('Size of vector embeddings: ', train_vec.shape)
print('Shape of vectors embeddings matrix: ', train_vec.shape)
```

```
Total time passed parsing documents: 61.13515377044678 seconds
Total number of documents parsed: 19579
Number of words in first document:  231
```

```
Number of words in second document:  71
Size of vector embeddings:  (19579,)
Shape of vectors embeddings matrix:  (19579,)
```

[50]:
```
all_text = pd.DataFrame(train, columns=['Text'])

print('Number of total text documents:', len(all_text))
```

```
Number of total text documents: 19579
```

[51]:
```
# Define function to preprocess text for a word2vec model
def cleanup_text_word2vec(docs, logging=False):
    sentences = []
    counter = 1
    for doc in docs:
        if counter % 1000 == 0 and logging:
            print("Processed %d out of %d documents" % (counter, len(docs)))
        # Disable tagger so that lemma_ of personal pronouns (I, me, etc) don't getted␣
 ↪marked as "-PRON-"
        doc = nlp(doc, disable=['tagger'])
        # Grab lemmatized form of words and make lowercase
        doc = " ".join([tok.lemma_.lower() for tok in doc])
        # Split into sentences based on punctuation
        doc = re.split("[\.?!;] ", doc)
        # Remove commas, periods, and other punctuation (mostly commas)
        doc = [re.sub("[\.,;:!?]", "", sent) for sent in doc]
        # Split into words
        doc = [sent.split() for sent in doc]
        sentences += doc
        counter += 1
    return sentences
```

```
<>:13: DeprecationWarning: invalid escape sequence \.
<>:15: DeprecationWarning: invalid escape sequence \.
<>:13: DeprecationWarning: invalid escape sequence \.
<>:15: DeprecationWarning: invalid escape sequence \.
<>:13: DeprecationWarning: invalid escape sequence \.
<>:15: DeprecationWarning: invalid escape sequence \.
<ipython-input-58-9d1583af632e>:13: DeprecationWarning: invalid escape sequence
\.
  doc = re.split("[\.?!;] ", doc)
<ipython-input-58-9d1583af632e>:15: DeprecationWarning: invalid escape sequence
\.
  doc = [re.sub("[\.,;:!?]", "", sent) for sent in doc]
```

!!! la cellule ci dessous prend du temps, c'est normal, on est en train de parser 15000 textes.

[52]:
```
train_cleaned_word2vec = cleanup_text_word2vec(all_text['Text'], logging=True)
print('Cleaned up training data size (i.e. number of sentences): ',␣
 ↪len(train_cleaned_word2vec))
```

```
Processed 1000 out of 19579 documents
Processed 2000 out of 19579 documents
```

```
Processed 3000 out of 19579 documents
Processed 4000 out of 19579 documents
Processed 5000 out of 19579 documents
Processed 6000 out of 19579 documents
Processed 7000 out of 19579 documents
Processed 8000 out of 19579 documents
Processed 9000 out of 19579 documents
Processed 10000 out of 19579 documents
Processed 11000 out of 19579 documents
Processed 12000 out of 19579 documents
Processed 13000 out of 19579 documents
Processed 14000 out of 19579 documents
Processed 15000 out of 19579 documents
Processed 16000 out of 19579 documents
Processed 17000 out of 19579 documents
Processed 18000 out of 19579 documents
Processed 19000 out of 19579 documents
Cleaned up training data size (i.e. number of sentences):  26930
```

1.6.2 Le modèle Word2Vec

Ici on lui indique par combien de vecteur il faut représenter l'ensemble des mots.

[53]:
```
from gensim.models.word2vec import Word2Vec

text_dim = 300

print("Training Word2Vec model...")

wordvec_model = Word2Vec(train_cleaned_word2vec, size=text_dim, window=5, min_count=3,␣
 ↪workers=4, sg=1)

print("Word2Vec model created.")
print("%d unique words represented by %d dimensional vectors" % (len(wordvec_model.wv.
 ↪vocab), text_dim))
```

```
c:\python372_x64\lib\site-packages\scipy\sparse\sparsetools.py:21:
DeprecationWarning: `scipy.sparse.sparsetools` is deprecated!
scipy.sparse.sparsetools is a private module for scipy.sparse, and should not be
used.
  _deprecated()
```

```
Training Word2Vec model…
Word2Vec model created.
9195 unique words represented by 300 dimensional vectors
```

On peut ainsi demander au modèle quels sont les mots les plus similaires à d'autres d'après ces données : par exemple

[54]:
```
print(wordvec_model.wv.most_similar(positive=['woman','mother']))
print(wordvec_model.wv.most_similar(positive=['love']))
print(wordvec_model.wv.most_similar(positive=['fear']))
```

```
[('son', 0.9320223331451416), ('lady', 0.9270102977752686), ('youth',
0.9190002679824829), ('child', 0.9032077789306641), ('wife',
```

```
0.9024365544319153), ('nurse', 0.9020218253135681), ('girl',
0.8976081609725952), ('dye', 0.8891721963882446), ('physician',
0.8882074356079102), ('daughter', 0.8830063343048096)]
[('affection', 0.9054282903671265), ('thy', 0.8841571807861328), ('grief',
0.8792158365249634), ('sympathy', 0.8759533166885376), ('tender',
0.8607128858566284), ('sweet', 0.8601766228675842), ('kindness',
0.8561927676200867), ('lover', 0.8538906574249268), ('creature',
0.8527849912643433), ('passion', 0.8490328788757324)]
[('hope', 0.9003583192825317), ('toil', 0.8973568677902222), ('madness',
0.8961610794067383), ('overcome', 0.8856801986694336), ('forget',
0.8853425979614258), ('untill', 0.8850491046905518), ('woodville',
0.8848880529403687), ('freedom', 0.88478684425354), ('sympathize',
0.8840764760971069), ('revive', 0.8812857866287231)]
```

[55]:
```
# Define function to create word vectors given a cleaned piece of text.
def create_average_vec(doc):
    average = np.zeros((text_dim,), dtype='float32')
    num_words = 0.
    for word in doc.split():
        if word in wordvec_model.wv.vocab:
            average = np.add(average, wordvec_model[word])
            num_words += 1.
    if num_words != 0.:
        average = np.divide(average, num_words)
    return average
```

[56]:
```
# Counting the number of empty strings are in train_cleaned
count = 0
for i in range(len(train_cleaned)):
    if train_cleaned[i] == "":
        print("index:", i)
        count += 1
print(count)
```

```
index: 477
index: 1023
index: 3601
index: 8654
4
```

[57]:
```
# Create word vectors
import warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore", DeprecationWarning)

    train_cleaned_vec = np.zeros((train.shape[0], text_dim), dtype="float32")  # 19579
 ↪x 300
    for i in range(len(train_cleaned)):
        train_cleaned_vec[i] = create_average_vec(train_cleaned[i])

print("Train word vector shape:", train_cleaned_vec.shape)
```

```
Train word vector shape: (19579, 300)
```

[58]:
```
from sklearn.preprocessing import label_binarize

# Transform labels into one hot encoded format.
y_train_ohe = label_binarize(train['Author'], classes=['EAP', 'HPL', 'MWS'])
print('y_train_ohe shape: {}'.format(y_train_ohe.shape))
print('y_train_ohe samples:')
print(y_train_ohe[:5])
```

```
y_train_ohe shape: (19579, 3)
y_train_ohe samples:
[[1 0 0]
 [0 1 0]
 [1 0 0]
 [0 0 1]
 [0 1 0]]
```

[59]:
```
from sklearn.model_selection import train_test_split

# If using spaCy word vectors
# X_train, X_test, y_train, y_test = train_test_split(train_vec, y_train_ohe,␣
 ↪test_size=0.2, random_state=21)
# If using Word2Vec word vectors
X_train, X_test, y_train, y_test = train_test_split(train_cleaned_vec, y_train_ohe,␣
 ↪test_size=0.2, random_state=21)

print('X_train size: {}'.format(X_train.shape))
print('X_test size: {}'.format(X_test.shape))
print('y_train size: {}'.format(y_train.shape))
print('y_test size: {}'.format(y_test.shape))
```

```
X_train size: (15663, 300)
X_test size: (3916, 300)
y_train size: (15663, 3)
y_test size: (3916, 3)
```

1.6.3 Identification d'un auteur

`y_train` contient un entier correspond à un des trois auteurs considérés.

[60]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore", DeprecationWarning)

    from keras.models import Sequential, Model
    from keras.layers import Dense, Dropout, Input, LSTM, Embedding, Bidirectional,␣
 ↪Flatten
    from keras.layers import Conv1D, MaxPooling1D, GlobalMaxPooling1D
    from keras.optimizers import SGD

def build_model():
    model = Sequential()
        # Densely Connected Neural Network (Multi-Layer Perceptron)
    model.add(Dense(512, activation='relu', kernel_initializer='he_normal',␣
 ↪input_dim=300))
```

```
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
    model.add(Dropout(0.2))
    model.add(Dense(3, activation='softmax'))
    return model
```

```
Using TensorFlow backend.
c:\python372_x64\lib\site-packages\tensorflow\python\framework\dtypes.py:526:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
c:\python372_x64\lib\site-packages\tensorflow\python\framework\dtypes.py:527:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
c:\python372_x64\lib\site-packages\tensorflow\python\framework\dtypes.py:528:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
c:\python372_x64\lib\site-packages\tensorflow\python\framework\dtypes.py:529:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
c:\python372_x64\lib\site-packages\tensorflow\python\framework\dtypes.py:530:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
c:\python372_x64\lib\site-packages\tensorflow\python\framework\dtypes.py:535:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

Définit un modèle keras.

[61]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore", DeprecationWarning)

    model = build_model()  #('mlp')
    model.summary()
```

```
WARNING:tensorflow:From c:\python372_x64\lib\site-
packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
```

```
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From c:\python372_x64\lib\site-
packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               154112
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 1539
=================================================================
Total params: 943,619
Trainable params: 943,619
Non-trainable params: 0
_________________________________________________________________
```

Compilaton du modèle.

[62]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore", DeprecationWarning)

    sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['acc'])
```

Une fois le modèle spécifié, il faut encore lui indiquer combien de fois il devrait être appliqué aux données. C'est la notion d'epochs > "generally defined as"one pass over the entire dataset".

[63]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore", DeprecationWarning)

    # Define number of epochs
    epochs = 50

    # Fit the model to the training data
    estimator = model.fit(X_train, y_train,
                          validation_split=0.2,
```

```
                              epochs=epochs, batch_size=128, verbose=100)
```

```
WARNING:tensorflow:From c:\python372_x64\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Train on 12530 samples, validate on 3133 samples
Epoch 1/50
Epoch 2/50
Epoch 3/50
Epoch 4/50
Epoch 5/50
Epoch 6/50
Epoch 7/50
Epoch 8/50
Epoch 9/50
Epoch 10/50
Epoch 11/50
Epoch 12/50
Epoch 13/50
Epoch 14/50
Epoch 15/50
Epoch 16/50
Epoch 17/50
Epoch 18/50
Epoch 19/50
Epoch 20/50
Epoch 21/50
Epoch 22/50
Epoch 23/50
Epoch 24/50
Epoch 25/50
Epoch 26/50
Epoch 27/50
Epoch 28/50
Epoch 29/50
Epoch 30/50
Epoch 31/50
Epoch 32/50
Epoch 33/50
Epoch 34/50
Epoch 35/50
Epoch 36/50
Epoch 37/50
Epoch 38/50
Epoch 39/50
Epoch 40/50
Epoch 41/50
Epoch 42/50
Epoch 43/50
Epoch 44/50
Epoch 45/50
```

```
Epoch 46/50
Epoch 47/50
Epoch 48/50
Epoch 49/50
Epoch 50/50
```

[64]:
```
print("Training accuracy: %.2f%% / Validation accuracy: %.2f%%" %
      (100*estimator.history['acc'][-1], 100*estimator.history['val_acc'][-1]))
```

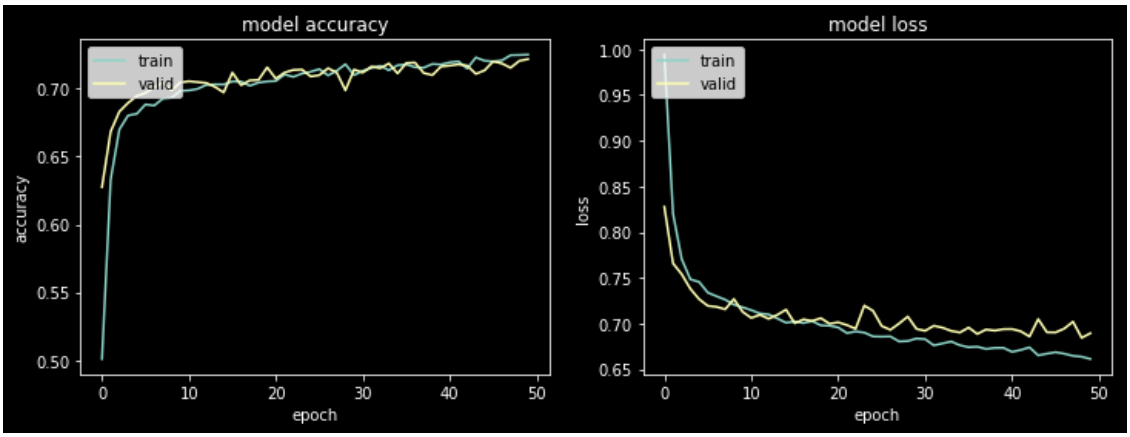
```
Training accuracy: 72.50% / Validation accuracy: 72.17%
```

[65]:
```
# Plot model accuracy over epochs
fig, ax = plt.subplots(1, 2, figsize=(12,4))

sns.reset_orig()   # Reset seaborn settings to get rid of black background
ax[0].plot(estimator.history['acc'])
ax[0].plot(estimator.history['val_acc'])
ax[0].set_title('model accuracy')
ax[0].set_ylabel('accuracy')
ax[0].set_xlabel('epoch')
ax[0].legend(['train', 'valid'], loc='upper left')

# Plot model loss over epochs
ax[1].plot(estimator.history['loss'])
ax[1].plot(estimator.history['val_loss'])
ax[1].set_title('model loss')
ax[1].set_ylabel('loss')
ax[1].set_xlabel('epoch')
ax[1].legend(['train', 'valid'], loc='upper left');
```

```
c:\python372_x64\lib\site-packages\matplotlib\__init__.py:855:
MatplotlibDeprecationWarning:
examples.directory is deprecated; in the future, examples will be found relative
to the 'datapath' directory.
  "found relative to the 'datapath' directory.".format(key))
c:\python372_x64\lib\site-packages\matplotlib\__init__.py:846:
MatplotlibDeprecationWarning:
The text.latex.unicode rcparam was deprecated in Matplotlib 2.2 and will be
removed in 3.1.
  "2.2", name=key, obj_type="rcparam", addendum=addendum)
```



[66]:
```
# Make predictions
predicted_prob = model.predict(X_test)
print(predicted_prob.shape)
```

```
(3916, 3)
```

[67]:
```
predicted_prob, y_test
```

[67]:
```
(array([[0.06372661, 0.02075616, 0.9155173 ],
        [0.2703639 , 0.19288836, 0.5367478 ],
        [0.23529688, 0.11483311, 0.64987004],
        …,
        [0.01442218, 0.00138307, 0.98419476],
        [0.24399275, 0.17094396, 0.58506334],
        [0.10251086, 0.15703583, 0.7404533 ]], dtype=float32),
 array([[0, 0, 1],
        [0, 0, 1],
        [0, 0, 1],
        …,
        [0, 0, 1],
        [1, 0, 0],
        [0, 0, 1]]))
```

[68]:

[69]: