

# 1 Préambule

## 1.1 Imports

```
[1]: # setting up sys.path for relative imports
from pathlib import Path
import sys
project_root = str(Path(sys.path[0]).parents[1].absolute())
if project_root not in sys.path:
    sys.path.append(project_root)

[201]: # imports and customization of display
import os
from functools import partial
from collections import defaultdict
import re
import numpy as np
from scipy.stats import linregress
import pandas as pd
pd.options.display.min_rows = 6
pd.options.display.width=108
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import clone
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.pipeline import Pipeline
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap
from sklearn.preprocessing import StandardScaler
from gensim.models import word2vec

from matplotlib import pyplot as plt
import matplotlib.patches as mpatch
import matplotlib.cm as cm
import matplotlib.colors as colors
from matplotlib.colors import Normalize
import matplotlib.ticker as mtick
import seaborn as sns

from src.pimest import ContentGetter
from src.pimest import PathGetter
from src.pimest import PDFContentParser
from src.pimest import BlockSplitter
from src.pimest import SimilaritySelector
from src.pimest import custom_accuracy
from src.pimest import text_sim_score
from src.pimest import text_similarity
from src.pimest import build_text_processor
```

## 1.2 Acquisition des données de la ground truth

```
[4]: ground_truth_df = pd.read_csv(Path('..') / '..' / 'ground_truth' / 'manually_labelled_ground_truth.csv',
                                   sep=';',
                                   encoding='latin-1',
                                   index_col='uid')
ground_truth_uids = list(ground_truth_df.index)

acqui_pipe = Pipeline([('PathGetter', PathGetter(ground_truth_uids=ground_truth_uids,
                                                train_set_path=Path('..') / '..' / 'ground_truth',
                                                ground_truth_path=Path('..') / '..' / 'ground_truth',
                                                )),
                      ('ContentGetter', ContentGetter(missing_file='to_nan')),
                      ('ContentParser', PDFContentParser(none_content='to_empty'))],
```

```

],
verbose=True)

texts_df = acqui_pipe.fit_transform(ground_truth_df)
texts_df['ingredients'] = texts_df['ingredients'].fillna('')
texts_df

```

```

[Pipeline] ... (step 1 of 3) Processing PathGetter, total= 0.1s
[Pipeline] ... (step 2 of 3) Processing ContentGetter, total= 0.1s
Launching 8 processes.
[Pipeline] ... (step 3 of 3) Processing ContentParser, total= 39.1s

```

```

[4]:
                                     designation \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70    Concentré liquide Asian en bouteille 980 ml CHEF
d183e914-db2f-4e2f-863a-a3b2d054c0b8    Pain burger curry 80 g CREATIV BURGER
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8    Macaroni en sachet 500 g PANZANI
...
e67341d8-350f-46f4-9154-4dbbb8035621    PRÉPARATION POUR CRÈME BRÛLÉE BIO 6L
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3    Céréales instantanées en poudre saveur caramel...
0faad739-ea8c-4f03-b62e-51ee592a0546    FARINE DE BLÉ TYPE 45, 10KG

                                     ingredients \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70    Eau, maltodextrine, sel, arômes, sucre, arôme ...
d183e914-db2f-4e2f-863a-a3b2d054c0b8    Farine de blé T65, eau, levure, vinaigre de ci...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8    - 100% Semoule de BLE dur de qualité supérieur...
...
e67341d8-350f-46f4-9154-4dbbb8035621    Sucre roux de canne* (64%), amidon de maïs*, ...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3    Farine 87,1 % (Blé (GLUTEN), Blé hydrolysé (GL...
0faad739-ea8c-4f03-b62e-51ee592a0546    Farine de blé T45

                                     path \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70    ../../ground_truth/a0492df6-9c76-4303-8813-65e...
d183e914-db2f-4e2f-863a-a3b2d054c0b8    ../../ground_truth/d183e914-db2f-4e2f-863a-a3b...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8    ../../ground_truth/ab48a1ed-7a3d-4686-bb6d-ab4...
...
e67341d8-350f-46f4-9154-4dbbb8035621    ../../ground_truth/e67341d8-350f-46f4-9154-4db...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3    ../../ground_truth/a8f6f672-20ac-4ff8-a8f2-3bc...
0faad739-ea8c-4f03-b62e-51ee592a0546    ../../ground_truth/0faad739-ea8c-4f03-b62e-51e...

                                     content \
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70    b'%PDF-1.5\r\n%\xb5\xb5\xb5\b5\r\n1 0 obj\r\n...
d183e914-db2f-4e2f-863a-a3b2d054c0b8    b'%PDF-1.5\r%\xe2\xe3\xcf\xd3\r\n4 0 obj\r<</L...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8    b'%PDF-1.4\r%\xc7\xec\x8f\xa2\r\n5 0 obj\r<</Len...
...
e67341d8-350f-46f4-9154-4dbbb8035621    b'%PDF-1.7\r\n%\xb5\xb5\xb5\b5\r\n1 0 obj\r\n...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3    b'%PDF-1.5\r\n%\xb5\xb5\xb5\b5\r\n1 0 obj\r\n...
0faad739-ea8c-4f03-b62e-51ee592a0546    b'%PDF-1.5\r\n%\xb5\xb5\xb5\b5\r\n1 0 obj\r\n...

                                     text
uid
a0492df6-9c76-4303-8813-65ec5ccbfa70    Concentré Liquide Asian CHEF® \n\nBouteille de...
d183e914-db2f-4e2f-863a-a3b2d054c0b8
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8    Direction Qualité \n\n \n\n \n\nPATES ALIMENTA...
...
e67341d8-350f-46f4-9154-4dbbb8035621    FICHE TECHNIQUE \n\nCREME BRÛLÉE 6L \n\nREF : ...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3    81 rue de Sans Souci - CS13754 - 69576 Limones...
0faad739-ea8c-4f03-b62e-51ee592a0546    \n1050/10502066400 \n\n10502055300/1050202520...

```

[500 rows x 5 columns]

On fusionne les corpus

```
[5]: target_df = texts_df['ingredients'].rename('text').to_frame()
target_df.loc[:, 'source'] = 'target'
content_df = texts_df['text'].to_frame()
content_df['source'] = 'content'
corpus_df = pd.concat([target_df, content_df])
corpus_df = corpus_df.reset_index().set_index(['source', 'uid'])
corpus_df['text'].fillna('', inplace=True)
corpus_df
```

```
[5]:
```

	source	uid	text
target	a0492df6-9c76-4303-8813-65ec5ccbfa70		Eau, maltodextrine, sel, arômes, sucre, arôme ...
	d183e914-db2f-4e2f-863a-a3b2d054c0b8		Farine de blé T65, eau, levure, vinaigre de ci...
	ab48a1ed-7a3d-4686-bb6d-ab4f367cada8		- 100% Semoule de BLE dur de qualité supérieur...
content	e67341d8-350f-46f4-9154-4dbbb8035621		FICHE TECHNIQUE \n\nCREME BRÛLÉE 6L \n\nREF : ...
	a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3		81 rue de Sans Souci - CS13754 - 69576 Limones...
	0faad739-ea8c-4f03-b62e-51ee592a0546		\n1050/10502066400 \n\n10502055300/1050202520...

[1000 rows x 4 columns]

## 2 Analyses

### 2.1 Analyse des longueurs des textes

```
[6]: lengths = corpus_df['text'].apply(len)
print(
lengths.reset_index()
.groupby('source')
.describe()
.rename({'content': 'Texte des documents',
'target': "Listes d'ingrédients"}))
.to_latex(Path('.') / 'tbls' / 'text_lengths.tex',
bold_rows=True,
column_format='lccccccc',
index_names=False,
)
lengths.reset_index().groupby('source').describe()
```

```
\begin{tabular}{lccccccc}
\toprule
{} & count & mean & std & min & 25\% & 50\% & 75\% & max \\
\midrule
\textbf{Texte des documents} & 500.0 & 3937.630 & 3280.860732 & 0.0 & 2173.75 & 3247.5 & 5034.25 & 37322.0 \\
\textbf{Listes d'ingrédients} & 500.0 & 199.686 & 457.044723 & 0.0 & 43.00 & 122.0 & 250.75 & 7963.0 \\
\bottomrule
\end{tabular}
```

```
[6]:
```

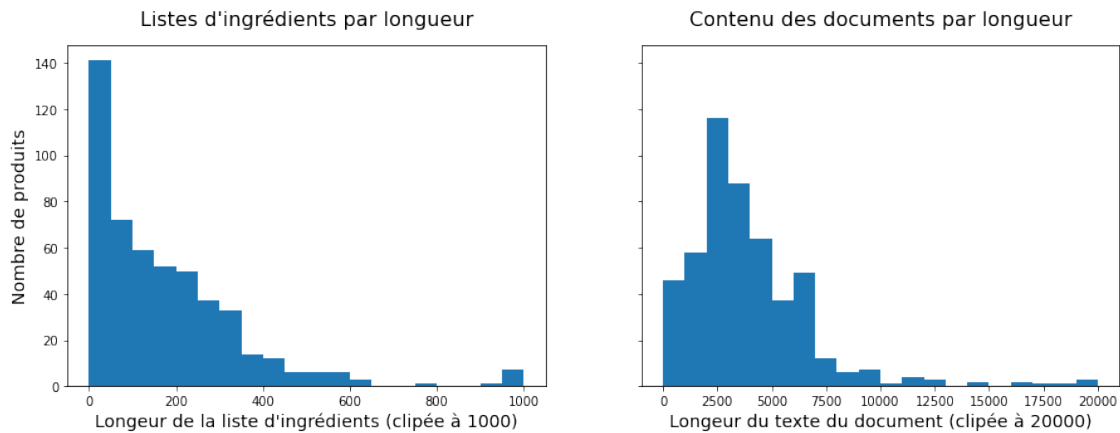
	text count	mean	std	min	25%	50%	75%	max
source								
content	500.0	3937.630	3280.860732	0.0	2173.75	3247.5	5034.25	37322.0
target	500.0	199.686	457.044723	0.0	43.00	122.0	250.75	7963.0

```
[7]: fig, axs = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
axs[0].hist(lengths.loc['target'].clip(0, 1000), bins=20)
axs[0].set_title("Listes d'ingrédients par longueur", fontsize=16, pad=15)
axs[0].set_ylabel("Nombre de produits", fontsize=14)
axs[0].set_xlabel("Longueur de la liste d'ingrédients (clippée à 1000)", fontsize=14)
```

```

axs[1].hist(lengths.loc['content'].clip(0, 20000), bins=20)
axs[1].set_title("Contenu des documents par longueur", fontsize=16, pad=15)
axs[1].set_xlabel("Longueur du texte du document (clipée à 20000)", fontsize=14)
# fig.savefig(Path('.') / 'img' / 'text_lengths.png', bbox_inches='tight')

```

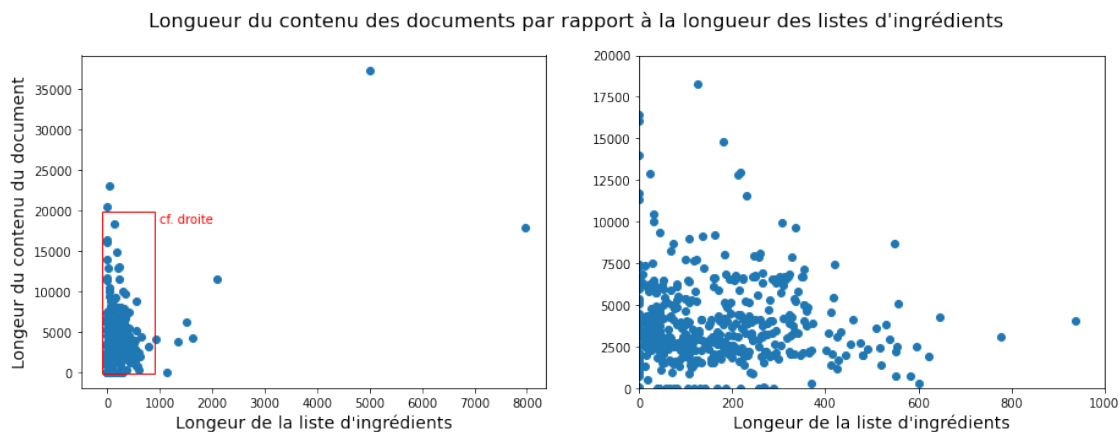


```

[8]: fig, axs = plt.subplots(ncols=2, figsize=(15, 5))

axs[0].scatter(x=lengths.loc['target'], y=lengths.loc['content'])
axs[0].set_xlabel("Longueur de la liste d'ingrédients", fontsize=14)
axs[0].set_ylabel("Longueur du contenu du document", fontsize=14)
axs[0].add_patch(mpatch.Rectangle((-100, -150), 1000, 20000, fill=False, color='red'))
axs[0].annotate('cf. droite', (1000, 18500), color='red')
axs[1].scatter(x=lengths.loc['target'], y=lengths.loc['content'])
axs[1].set_xlabel("Longueur de la liste d'ingrédients", fontsize=14)
axs[1].set_xlim(0, 1000)
axs[1].set_ylim(0, 20000)
fig.suptitle("Longueur du contenu des documents par rapport à la longueur des listes d'ingrédients",
             ↵, fontsize=16)
# fig.savefig(Path('.') / 'img' / 'text_lengths_2.png', bbox_inches='tight')

```



```

[9]: linregress(x=lengths.loc['target'], y=lengths.loc['content']).rvalue ** 2

```

[9]: 0.139326761283587

## 2.2 Analyse des mots

```
[76]: stop_words = {'de', 'et', 'la', 'en', 'du', 'les', 'des', 'le', 'dans', 'ce', 'un', 'pour',  
                  'par', 'sur', 'au', 'dont', 'of', 'and', 'pas', 'est', 'ou', 'que'}  
print(stop_words)
```

{'par', 'pas', 'dans', 'des', 'les', 'pour', 'ce', 'est', 'de', 'of', 'un', 'and', 'la', 'sur', 'du', 'en', 'au', 'ou', 'et', 'que', 'dont', 'le'}

```
[11]: tfidf_corpus = TfidfVectorizer(strip_accents='unicode',  
                                     lowercase=True,  
                                     stop_words=stop_words,  
                                     ngram_range=(1, 1),  
                                     max_df=1.0,  
                                     #min_df=0.0, A TESTER !  
                                     binary=False,  
                                     norm=None, # set to l1 or l2  
                                     use_idf=False,  
                                     smooth_idf=False,  
                                     sublinear_tf=False,  
                                     )  
vectorized_corpus = tfidf_corpus.fit_transform(corpus_df['text'])  
tokenizer = tfidf_corpus.build_analyzer()
```

Comptage des mots du vocabulaire du corpus :

```
[12]: inverse_corpus_voc = {val: key for key, val in tfidf_corpus.vocabulary_.items()}  
word_counts = np.asarray(vectorized_corpus.sum(axis=0)).squeeze()  
print(f'Corpus vocabulary size is ', len(tfidf_corpus.vocabulary_), '\n')  
print('Most frequent words in vocabulary are:')  
most_freq = dict()  
for idx in word_counts.argsort()[::-1][:20]:  
    most_freq[inverse_corpus_voc[idx].ljust(12)] = int(word_counts[idx])  
    print(f'{inverse_corpus_voc[idx].ljust(12)}: {word_counts[idx]:5} occurrences')  
  
str_cpt_corpus = [f'{word.ljust(12)}: {most_freq[word]} occurrences \\\\' for word in most_freq.keys()]
```

Corpus vocabulary size is 15465

Most frequent words in vocabulary are:

produit	: 2198.0 occurrences
non	: 2164.0 occurrences
produits	: 1508.0 occurrences
10	: 1404.0 occurrences
kg	: 1290.0 occurrences
base	: 1119.0 occurrences
sel	: 1041.0 occurrences
poids	: 1021.0 occurrences
100	: 971.0 occurrences
palette	: 933.0 occurrences
ingredients	: 917.0 occurrences
date	: 904.0 occurrences
sucre	: 841.0 occurrences
12	: 803.0 occurrences
code	: 801.0 occurrences
lait	: 733.0 occurrences
absence	: 697.0 occurrences
fiche	: 688.0 occurrences
the	: 658.0 occurrences
france	: 657.0 occurrences

Constitution du vocabulaire des listes d'ingrédients

```
[13]: tfidf_ingred = clone(tfidf_corpus)  
vectorized_ingred = tfidf_ingred.fit_transform(corpus_df.loc['target', 'text'])
```

```
[14]: inverse_ingred_voc = {val: key for key, val in tfidf_ingred.vocabulary_.items()}
word_counts = np.asarray(vectorized_ingred.sum(axis=0)).squeeze()
print(f'Ingredient vocabulary size is ', len(tfidf_ingred.vocabulary_), '\n')
print('Most frequent words in vocabulary are:')
most_freq = dict()
for idx in word_counts.argsort()[::-1][:20]:
    most_freq[inverse_ingred_voc[idx].ljust(12)] = int(word_counts[idx])
    print(f'{inverse_ingred_voc[idx].ljust(10)}: {word_counts[idx]:5} occurrences')

str_cpt_corpus_2 = [f'{word.ljust(12)}: {most_freq[word]} occurrences &' for word in most_freq.keys()]
str_latex = '\n'.join([str2 + ' ' + str1 for str1, str2 in zip(str_cpt_corpus, str_cpt_corpus_2)])
#print(str_latex)
```

Ingredient vocabulary size is 1324

Most frequent words in vocabulary are:

```
sucre      : 363.0 occurrences
acide      : 255.0 occurrences
sel        : 240.0 occurrences
sirop      : 180.0 occurrences
eau        : 178.0 occurrences
poudre     : 176.0 occurrences
arome      : 175.0 occurrences
lait       : 171.0 occurrences
ble        : 171.0 occurrences
huile      : 146.0 occurrences
citrique   : 132.0 occurrences
farine     : 128.0 occurrences
amidon     : 125.0 occurrences
glucose    : 124.0 occurrences
cacao      : 122.0 occurrences
extrait    : 117.0 occurrences
acidifiant : 114.0 occurrences
aromes     : 98.0 occurrences
soja       : 96.0 occurrences
concentre  : 92.0 occurrences
```

On vérifie s'il existe des mots qui sont inclus dans les listes d'ingrédients mais pas dans les documents :

```
[15]: ingred_only = {word for word in tfidf_ingred.vocabulary_ if word not in tfidf_corpus.vocabulary_}
      ingred_only
```

```
[15]: set()
```

Constitution d'une nouvelle feature qui sont les textes tokenisés via la méthode du CountVectorizer.

```
[16]: corpus_df['tokenized'] = corpus_df['text'].apply(tokenizer)
      corpus_df
```

```
[16]:
```

	text \
source uid	
target a0492df6-9c76-4303-8813-65ec5ccbfa70	Eau, maltodextrine, sel, arômes, sucre, arôme ...
d183e914-db2f-4e2f-863a-a3b2d054c0b8	Farine de blé T65, eau, levure, vinaigre de ci...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8	- 100% Semoule de BLE dur de qualité supérieur...
...	...
content e67341d8-350f-46f4-9154-4dbbb8035621	FICHE TECHNIQUE \n\nCREME BRÛLÉE 6L \n\nREF : ...
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3	81 rue de Sans Souci - CS13754 - 69576 Limones...
0faad739-ea8c-4f03-b62e-51ee592a0546	\n1050/10502066400 \n\n10502055300/1050202520...
...	...
source uid	
target a0492df6-9c76-4303-8813-65ec5ccbfa70	[eau, maltodextrine, sel, aromes, sucre, arome...
d183e914-db2f-4e2f-863a-a3b2d054c0b8	[farine, ble, t65, eau, levure, vinaigre, cidr...
ab48a1ed-7a3d-4686-bb6d-ab4f367cada8	[100, semoule, ble, dur, qualite, superieure, ...
...	...
content e67341d8-350f-46f4-9154-4dbbb8035621	[fiche, technique, creme, brulee, 6l, ref, nap...

```
a8f6f672-20ac-4ff8-a8f2-3bc4306c8df3 [81, rue, sans, souci, cs13754, 69576, limones...
0faad739-ea8c-4f03-b62e-51ee592a0546 [1050, 10502066400, 10502055300, 10502025200, ...
```

```
[1000 rows x 2 columns]
```

### 3 Analyse de données

## 4 Représentation des mots

### 4.1 Document frequency

1- Déjà, on calcule une métrique qui permet de dire si un mot est plutôt un mot de type ingrédients ou un mot issu du corpus.

Représentation simple : on regarde simplement la document frequency de chacun des mots dans le corpus des ingrédients.

```
[17]: doc_freq_counter = clone(tfidf_ingred)
doc_freq_counter.set_params(binary=True).fit(corpus_df.loc['target', 'text'])
binary_ingred_counts = doc_freq_counter.transform(corpus_df.loc[:, 'text'])

[18]: doc_freq_ingred = binary_ingred_counts[:500]
doc_freq_ingred = np.array(doc_freq_ingred.sum(axis=0)).reshape(-1) / 500
print('shape of doc_freq_ingred :', doc_freq_ingred.shape)
doc_freq_ingred_translator = defaultdict(lambda: 0., {word: doc_freq_ingred[tfidf_ingred.vocabulary_[word]]
                                     for word in tfidf_ingred.vocabulary_.keys()})

print(doc_freq_ingred_translator['sucre'])
print(doc_freq_ingred_translator['taboulllleeeeeeeeeiiiiiii'])
print(doc_freq_ingred_translator['maltodextrine'])
```

```
shape of doc_freq_ingred : (1324,)
0.448
0.0
0.064
```

On constitue un array qui nous servira plus tard, avec le document frequency dans les ingrédients de tous les mots utilisés dans le contenu des documents.

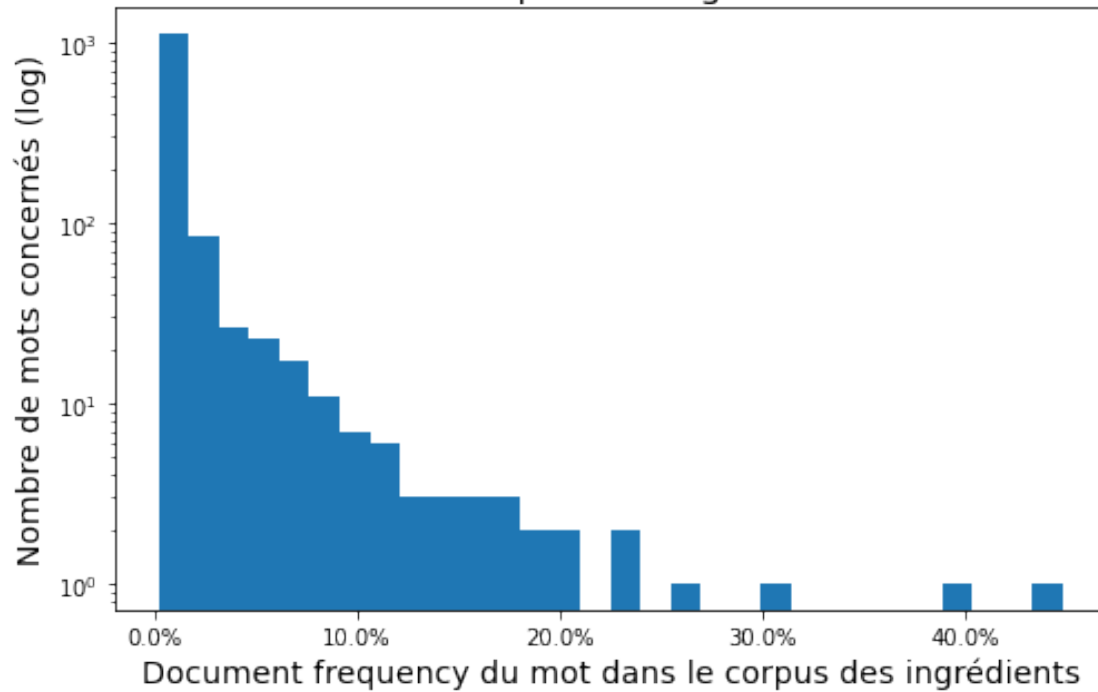
```
[19]: corpus_doc_freq_ingred = np.zeros(shape=(len(tfidf_corpus.vocabulary_)))
for i in range(len(tfidf_corpus.vocabulary_)):
    corpus_doc_freq_ingred[i] = doc_freq_ingred_translator[inverse_corpus_voc[i]]
corpus_doc_freq_ingred
```

```
[19]: array([0.004, 0.    , 0.    , ..., 0.    , 0.    , 0.    ])
```

On peut sortir les top scorer de la doc frequency :

```
[28]: fig, ax = plt.subplots(figsize=(8, 5))
ax.hist(doc_freq_ingred, bins=30, log=True)
ax.xaxis.set_major_formatter(mtick.PercentFormatter(xmax=1.))
ax.set_xlabel("Document frequency du mot dans le corpus des ingrédients", fontsize=14)
ax.set_ylabel("Nombre de mots concernés (log)", fontsize=14)
fig.suptitle("Répartition des mots par documents frequency dans le corpus des ingrédients", fontsize=16)
# fig.savefig(Path('.') / 'img' / 'doc_freq_of_ingreds.png', bbox_inches='tight')
```

## Répartition des mots par documents frequency dans le corpus des ingrédients



```
[22]: for word_idx in doc_freq_ingred.argsort()[::-1][:20]:
      print(inverse_ingred_voc[word_idx], '&', f'{doc_freq_ingred[word_idx]*100:.2f}\\%', '\\\\')

      for word_idx in doc_freq_ingred.argsort()[:20]:
      print(inverse_ingred_voc[word_idx], '&', f'{doc_freq_ingred[word_idx]*100:.2f}\\%', '\\\\')
```

```
sucre & 44.80\\% \\
sel & 40.00\\% \\
eau & 30.20\\% \\
acide & 26.20\\% \\
arome & 23.00\\% \\
huile & 23.00\\% \\
ble & 20.60\\% \\
amidon & 20.00\\% \\
aromes & 19.00\\% \\
sirop & 18.40\\% \\
citrique & 18.00\\% \\
lait & 17.20\\% \\
acidifiant & 17.00\\% \\
farine & 16.40\\% \\
extrait & 16.20\\% \\
glucose & 16.00\\% \\
concentre & 14.80\\% \\
jus & 14.00\\% \\
poudre & 14.00\\% \\
colza & 13.20\\% \\
ferrique & 0.20\\% \\
d3 & 0.20\\% \\
datte & 0.20\\% \\
dattes & 0.20\\% \\
debris & 0.20\\% \\
decafeine & 0.20\\% \\
deciree & 0.20\\% \\
```



```

deconseille & 0.20\% \\  

nectar & 0.20\% \\  

necessaire & 0.20\% \\  

ne & 0.20\% \\  

delta & 0.20\% \\  

naturelles & 0.20\% \\  

denoyautees & 0.20\% \\  

dentier & 0.20\% \\  

depellicule & 0.20\% \\  

cysteine & 0.20\% \\  

negra & 0.20\% \\  

new & 0.20\% \\  

nicotinamide & 0.20\% \\  


```

```

[309]: # looking for first word in documents but not in ingredients
for word_idx in np.array(vectorized_corpus[500:].sum(axis=0)).reshape(15465).argsort()[::-1]:
    print(inverse_corpus_voc[word_idx])
    if inverse_corpus_voc[word_idx] not in tfidf_ingred.vocabulary_:
        print('trouvé !')
        break

```

```

produit
non
produits
10
kg
base
poids
trouvé !

```

```

[75]: ingred_scores.shape

```

```

[75]: (1324,)

```

```

[291]: ingred_scores = np.log2(doc_freq_ingred * 500 + 1)
sample = doc_freq_ingred.argsort()[::-1][[0, 10, 100, 250, 702]]
sample_scores = ingred_scores[sample]
labels = [inverse_ingred_voc[word_idx] + f'\n{ingred_scores[word_idx]:.2f}' for word_idx in sample] +_
↳ ['poids\n0.00']
labels

```

```

[291]: ['sucre\n7.81',
       'citrique\n6.51',
       'issus\n4.39',
       'si\n2.81',
       'plantations\n1.58',
       'poids\n0.00']

```

```

[292]: gradient = np.linspace(0, 8, 256)
gradient = np.vstack((gradient, gradient, gradient))

fig, ax = plt.subplots(figsize=(8, 3))

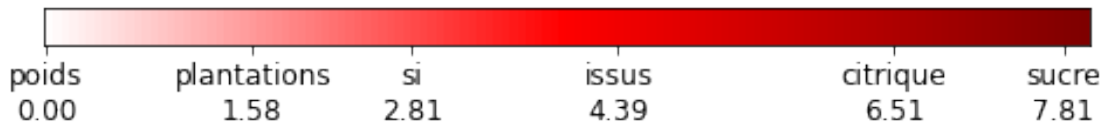
ax.imshow(gradient, aspect=3, cmap='seismic', vmin=-8, vmax=8)
ax.set_xticks(np.hstack((255 / 8 * sample_scores, np.array([0]))))
ax.set_xticklabels(labels, fontsize = 12)
#ax.set_xlim(0, 50)
ax.set_yticks([])
# fig.savefig(Path('.') / 'img' / 'scores_bar.png', bbox_inches='tight')

```

```

[292]: []

```



On calcule les document frequencies des mots issus des documents, et on compte sur les corpus.

```
[293]: doc_freq_counter_2 = clone(tfidf_corpus)
doc_freq_counter_2.set_params(binary=False).fit(corpus_df.loc[:, 'text'])
word_counts = np.array(doc_freq_counter_2.transform(corpus_df.loc[:, 'text']).todense())
delta_doc_counts = (word_counts[500:] - word_counts[:500])
binary_delta = np.where(delta_doc_counts > 0., np.ones(delta_doc_counts.shape), np.zeros(delta_doc_counts.
↳ shape))
docs_scores = np.log2(binary_delta.sum(axis=0) + 1)
```

```
[294]: docs_scores[tfidf_corpus.vocabulary_['produits']]
```

```
[294]: 8.005624549193879
```

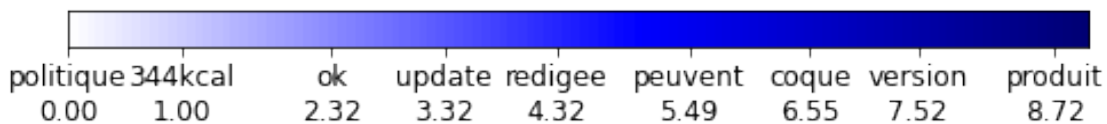
```
[295]: sample = docs_scores.argsort()[::-1][[0, 90, 251, 550, 1200, 2200, 3803, 10005]]
sample_scores = docs_scores[sample]
labels = [inverse_corpus_voc[word_idx] + f'\n{docs_scores[word_idx]:.2f}' for word_idx in sample]
labels
```

```
[295]: ['produit\n8.72',
'version\n7.52',
'coque\n6.55',
'peuvent\n5.49',
'redigee\n4.32',
'update\n3.32',
'ok\n2.32',
'344kcal\n1.00']
```

```
[298]: gradient = np.linspace(0, 8, 256)
gradient = np.vstack((gradient, gradient, gradient))

fig, ax = plt.subplots(figsize=(8, 3))

ax.imshow(gradient, aspect=3, cmap='seismic_r', vmin=-9, vmax=9)
ax.set_xticks(np.hstack((255 / 9 * sample_scores, np.array([0]))))
ax.set_xticklabels(labels + ["politique\n0.00"], fontsize = 12)
#ax.set_xlim(0, 50)
ax.set_yticks([])
# fig.savefig(Path('.') / 'img' / 'corpus_score_bar.png', bbox_inches='tight')
```



Et maintenant on calcule le score relatif de chacun des mots.

```
[299]: # new computation on ingredients, on corpus vocabulary indexing
binary_ingred = np.where(word_counts[:500] > 0., np.ones(word_counts[:500].shape), np.zeros(word_counts[:
↳ 500].shape))
ingred_scores = np.log2(binary_ingred.sum(axis=0) + 1)
# check computation is coherent
```

```
print(ingred_scores[tfidf_corpus.vocabulary_['sucre']])
```

7.813781191217037

```
[300]: sample = np.hstack([relative_scores.argsort()[[0, 90, 251, 1200, 3000, 8007, 15464]], [14079]])
sample_scores = relative_scores[sample]
labels = [inverse_corpus_voc[word_idx] + f'\n{relative_scores[word_idx]:.2f}' for word_idx in sample]
labels
```

```
[300]: ['glucides\n-8.67',
'800\n-7.04',
'tout\n-5.93',
'presence\n-4.09',
'fraude\n-2.58',
'proteges\n-1.00',
'acidifiants\n3.91',
'sucre\n1.51']
```

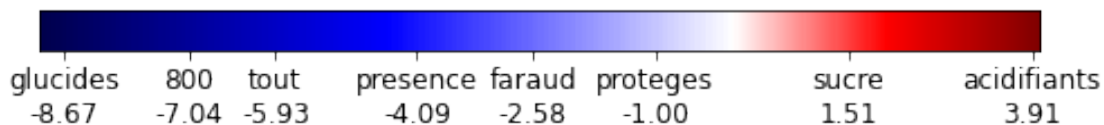
```
[301]: mmin, mmax, resolution = -9, 4, 1000
divnorm = colors.TwoSlopeNorm(vmin=mmin, vcenter=0, vmax=mmax)
gradient = np.linspace(mmin, mmax, resolution)
gradient = np.vstack([gradient, gradient])
# gradient = np.vstack([sorted(relative_scores), sorted(relative_scores)])

def convert_to_width(num):
    return(int(resolution * (num - mmin) / (mmax - mmin)))

fig, ax = plt.subplots(figsize=(8, 3))

ax.imshow(gradient, aspect=20, cmap='seismic', norm=divnorm) #, vmin=-9, vmax=9)
ax.set_xticks(list(map(convert_to_width, sample_scores)))
# ax.set_xticks(255 / 18 * sample_scores)
ax.set_xticklabels(labels, fontsize = 12)
#ax.set_xlim(0, 50)
ax.set_yticks([])
# fig.savefig(Path('.') / 'img' / 'relative_score_bar.png', bbox_inches='tight')
```

```
[301]: []
```



## 4.2 Word embeddings

### 4.2.1 Word2Vec

On calcule les embeddings avec Word2Vec

```
[302]: word2vec_model = word2vec.Word2Vec(corpus_df['tokenized'], min_count=1)
```

```
[303]: for i, word in enumerate(tfidf_corpus.vocabulary_.keys()):
print('-----\n', word, ': ', word2vec_model.wv[word][:5])
if i > 4:
break
```

```
-----
eau : [-1.0830716 -0.46610487 -0.43349764 0.55493397 1.1007366 ]
-----
```

```

maltodextrine : [ 0.01011625  0.15969634 -0.67031556  0.5559474  0.4602319 ]
-----
sel : [-0.8459699 -0.21097712 -0.6416249  0.84254736  0.9729415 ]
-----
aromes : [ 0.22419034  0.23973037 -1.4658073  1.1386586  1.0591131 ]
-----
sucre : [-0.74801284  0.1217097 -2.0248985  1.5143404  1.4470899 ]
-----
arome : [ 0.08999626  0.10789929 -2.2472398  1.5559452  1.5740927 ]

```

```

[265]: ndarray_words_embeddings = np.vstack([word2vec_model.wv[inverse_corpus_voc[i]]
                                             for i in range(len(tfidf_corpus.vocabulary_))])
ndarray_words_embeddings

```

```

[265]: array([[ -1.9143574e+00, -1.0786384e+00, -1.6952591e+00, ...,
                2.1197143e-04,  6.8014598e-01, -9.4786072e-01],
              [-1.0596310e+00, -5.1570371e-02, -2.4971814e+00, ...,
                -4.9154687e-01,  1.3782319e+00, -1.2871372e+00],
              [-2.1143033e-01, -1.3124047e-01, -2.7956560e-01, ...,
                -6.6144560e-03,  1.6432673e-02, -5.8027157e-03],
              ...,
              [-4.8991539e-02, -1.6540296e-02, -6.0264323e-02, ...,
                3.1673540e-03, -9.9934675e-03, -1.2510464e-02],
              [-8.2935727e-01, -6.4514387e-01, -8.7146807e-01, ...,
                -1.7320616e-02,  3.3239821e-01, -3.0926281e-01],
              [-2.5584301e-01, -1.5305966e-01, -5.1036954e-01, ...,
                -1.5918270e-02, -3.0415934e-02, -2.4762219e-01]], dtype=float32)

```

On effectue une PCA :

```

[266]: scaled_words_embeddings = StandardScaler().fit_transform(ndarray_words_embeddings)
scaled_words_embeddings

```

```

[266]: array([[ -10.750973 , -7.621986 , -6.1712112 , ...,  0.1407848 ,
                4.966931 , -4.985649 ],
              [ -5.818584 , -0.29638907, -9.341663 , ..., -5.478642 ,
                9.752907 , -6.8654494 ],
              [ -0.92385375, -0.8646387 , -0.5741748 , ...,  0.06277785,
                0.4166085 ,  0.23393212],
              ...,
              [  0.01353529, -0.04653592,  0.29284707, ...,  0.17455655,
                0.23543474,  0.19676708],
              [ -4.489738 , -4.5300717 , -2.9142997 , ..., -0.05956358,
                2.5828223 , -1.4474237 ],
              [ -1.180147 , -1.0202649 , -1.4866732 , ..., -0.04353869,
                0.09542128, -1.1058966 ]], dtype=float32)

```

```

[267]: PCA_model = PCA(n_components=10)
PCA_words_embeddings = PCA_model.fit_transform(scaled_words_embeddings)
print(PCA_words_embeddings.shape)

```

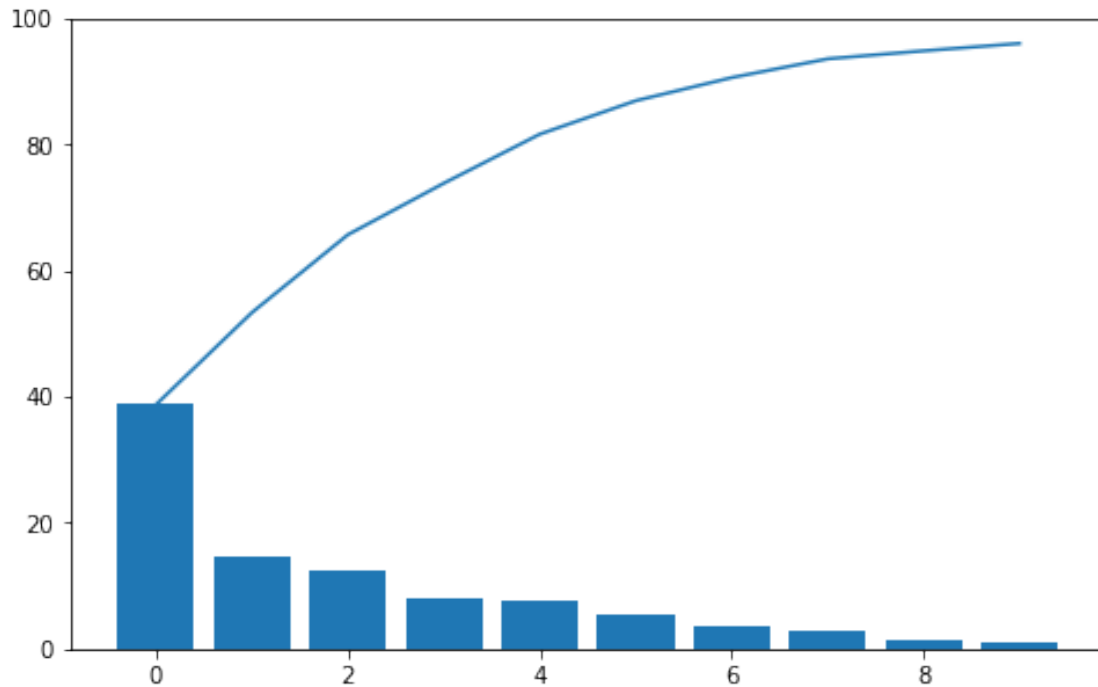
(15465, 10)

```

[268]: fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(PCA_model.explained_variance_ratio_.cumsum()*100)
ax.bar(x=range(10), height=PCA_model.explained_variance_ratio_*100)
ax.set_ylim(0, 100)

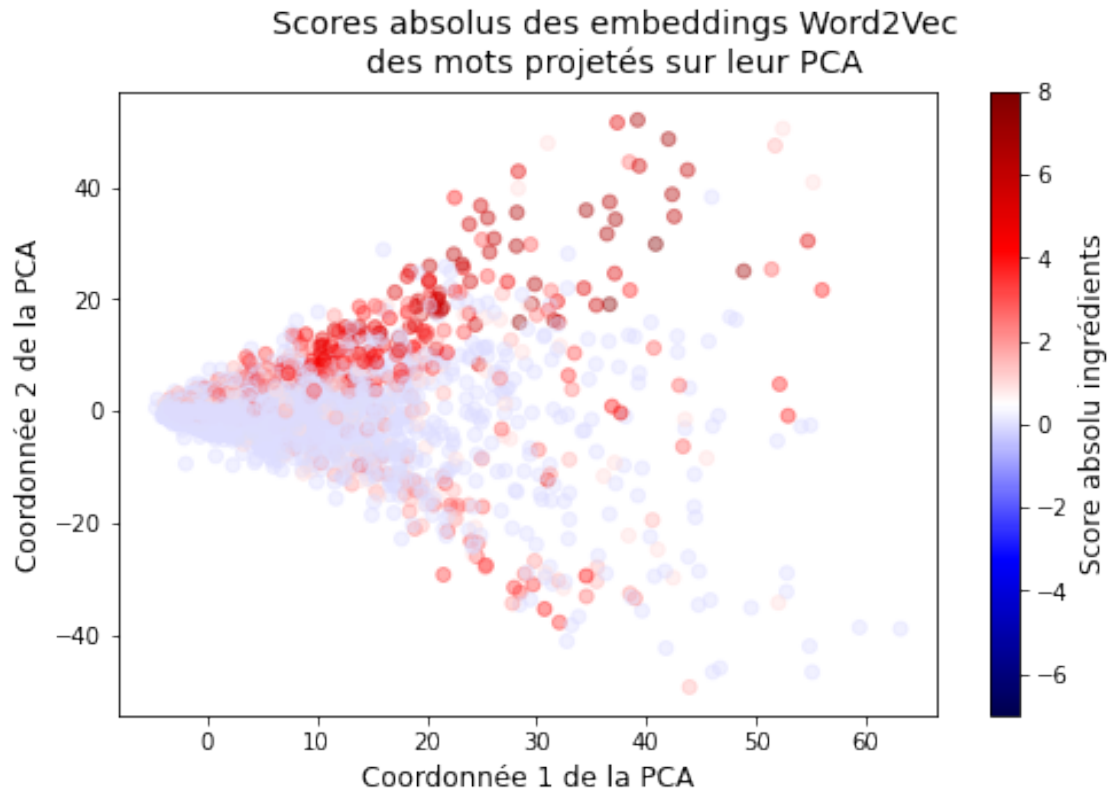
```

[268]: (0.0, 100.0)



On la dessine :

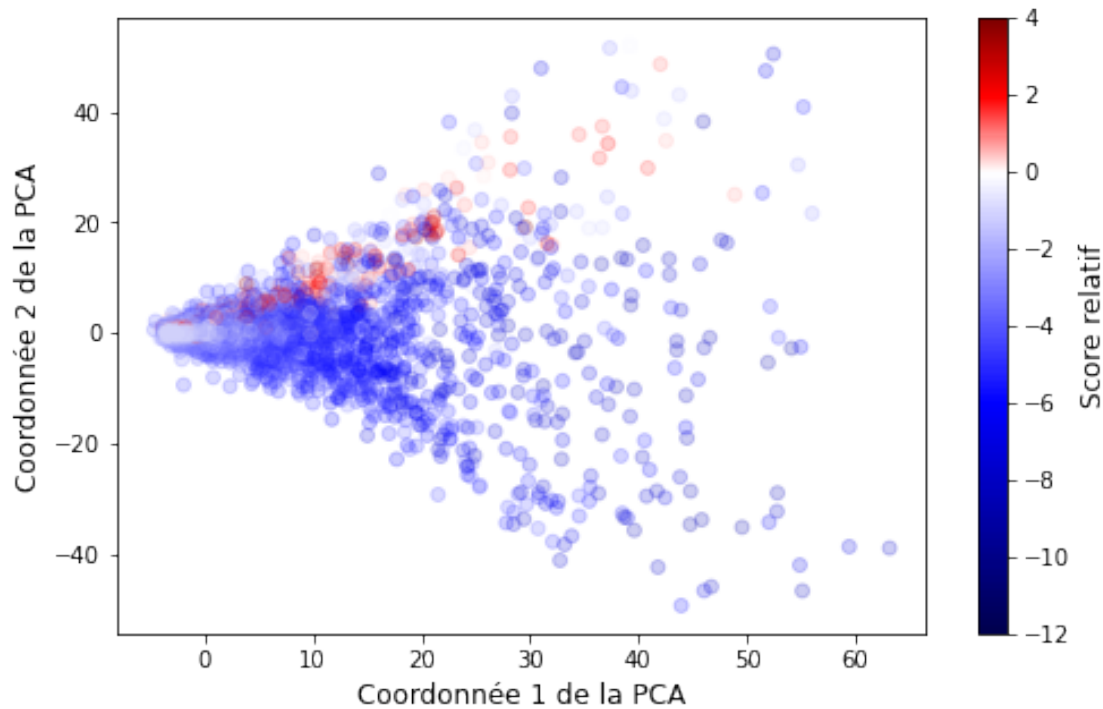
```
[304]: fig, ax = plt.subplots(figsize=(8, 5))
scat = ax.scatter(x=PCA_words_embeddings[:, 0],
                  y=PCA_words_embeddings[:, 1],
                  alpha=0.4,
                  cmap='seismic',
                  vmin=-7,
                  vmax=8,
                  c=np.log2(500 * corpus_doc_freq_ingred + 1),
                  )
ScalMappable = cm.ScalarMappable(norm=Normalize(vmin=-7, vmax=8, clip=True), cmap = 'seismic')
cb = fig.colorbar(ScalMappable, ax=ax)
fig.axes[1].set_ylabel('Score absolu ingrédients', fontsize=12)
fig.axes[0].set_xlabel('Coordonnée 1 de la PCA', fontsize=12)
fig.axes[0].set_ylabel('Coordonnée 2 de la PCA', fontsize=12)
fig.suptitle('Scores absolus des embeddings Word2Vec\ndes mots projetés sur leur PCA', fontsize=14)
fig.savefig(Path('.') / 'img' / 'word2vec_PCA.png', bbox_inches='tight')
```



```
[305]: mmin, mmax, resolution = -12, 4, 1000
divnorm = colors.TwoSlopeNorm(vmin=mmin, vcenter=0, vmax=mmax)

fig, ax = plt.subplots(figsize=(8, 5))
scat = ax.scatter(x=PCA_words_embeddings[:, 0],
                  y=PCA_words_embeddings[:, 1],
                  alpha=0.2,
                  cmap='seismic',
                  norm=divnorm,
                  c=relative_scores,
                  )
ScalMappable = cm.ScalarMappable(norm=divnorm, cmap = 'seismic')
cb = fig.colorbar(ScalMappable, ax=ax)
fig.axes[1].set_ylabel('Score relatif', fontsize=12)
fig.axes[0].set_xlabel('Coordonnée 1 de la PCA', fontsize=12)
fig.axes[0].set_ylabel('Coordonnée 2 de la PCA', fontsize=12)
fig.suptitle('Scores relatifs des Word2Vec des mots projetés sur leur PCA', fontsize=14)
fig.savefig(Path('.') / 'img' / 'word2vec_PCA_relative.png', bbox_inches='tight')
```

## Scores relatifs des Word2Vec des mots projetés sur leur PCA

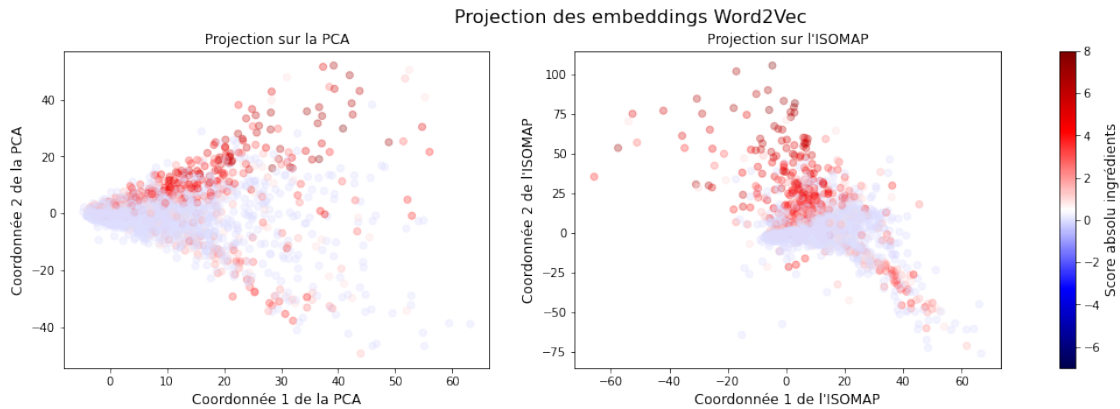


```
[275]: isomap_model = Isomap(n_neighbors=5, n_components=2, n_jobs=8)
isomap_words_embeddings = isomap_model.fit_transform(scaled_words_embeddings)
print(isomap_words_embeddings.shape)
```

(15465, 2)

```
[309]: fig, axs = plt.subplots(ncols=2, figsize=(18, 5))
scat = axs[0].scatter(x=PCA_words_embeddings[:, 0],
                    y=PCA_words_embeddings[:, 1],
                    alpha=0.3,
                    cmap='seismic',
                    vmin=-7,
                    vmax=8,
                    c=np.log2(500 * corpus_doc_freq_ingred + 1),
                    )
scat = axs[1].scatter(x=isomap_words_embeddings[:, 0],
                    y=isomap_words_embeddings[:, 1],
                    alpha=0.3,
                    cmap='seismic',
                    vmin=-7,
                    vmax=8,
                    c=np.log2(500 * corpus_doc_freq_ingred + 1),
                    )
ScalMappable = cm.ScalarMappable(norm=Normalize(vmin=-7, vmax=8, clip=True), cmap = 'seismic')
cb = fig.colorbar(ScalMappable, ax=axs)
fig.axes[2].set_ylabel("Score absolu ingrédients", fontsize=12)
fig.axes[0].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
fig.axes[0].set_ylabel("Coordonnée 2 de la PCA", fontsize=12)
fig.axes[0].set_title("Projection sur la PCA", fontsize=12)
fig.axes[1].set_xlabel("Coordonnée 1 de l'ISOMAP", fontsize=12)
fig.axes[1].set_ylabel("Coordonnée 2 de l'ISOMAP", fontsize=12)
```

```
fig.axes[1].set_title("Projection sur l'ISOMAP", fontsize=12)
fig.suptitle("Projection des embeddings Word2Vec", fontsize=16)
# fig.savefig(Path('.') / 'img' / 'word2vec_projection.png', bbox_inches='tight')
```

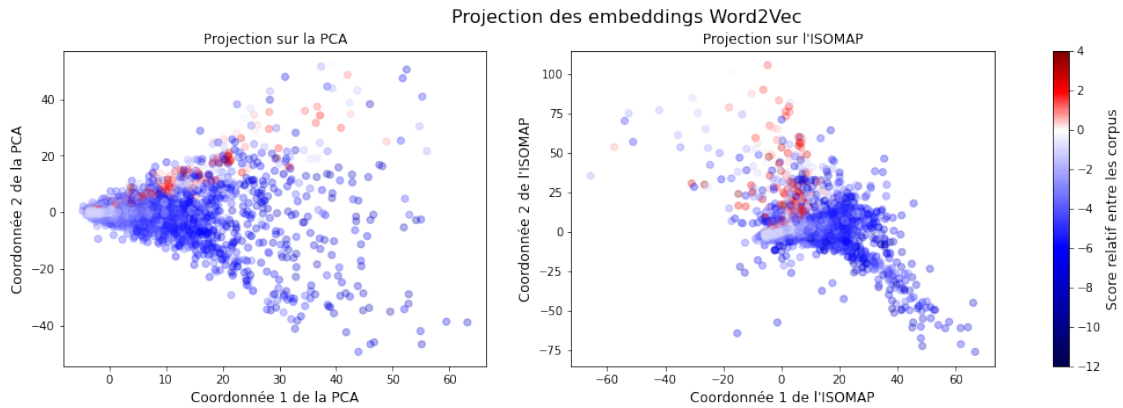


```
[310]: mmin, mmax, resolution = -12, 4, 1000
divnorm = colors.TwoSlopeNorm(vmin=mmin, vcenter=0, vmax=mmax)

fig, axs = plt.subplots(ncols=2, figsize=(18, 5))
scat = axs[0].scatter(x=PCA_words_embeddings[:, 0],
                    y=PCA_words_embeddings[:, 1],
                    alpha=0.3,
                    cmap='seismic',
                    norm=divnorm,
                    c=relative_scores,
                    )
scat = axs[1].scatter(x=isomap_words_embeddings[:, 0],
                    y=isomap_words_embeddings[:, 1],
                    alpha=0.3,
                    cmap='seismic',
                    norm=divnorm,
                    c=relative_scores,
                    )

ScalMappable = cm.ScalarMappable(norm=divnorm, cmap = 'seismic')
cb = fig.colorbar(ScalMappable, ax=axs)
fig.axes[2].set_ylabel("Score relatif entre les corpus", fontsize=12)
fig.axes[0].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
fig.axes[0].set_ylabel("Coordonnée 2 de la PCA", fontsize=12)
fig.axes[0].set_title("Projection sur la PCA", fontsize=12)
fig.axes[1].set_xlabel("Coordonnée 1 de l'ISOMAP", fontsize=12)
fig.axes[1].set_ylabel("Coordonnée 2 de l'ISOMAP", fontsize=12)
fig.axes[1].set_title("Projection sur l'ISOMAP", fontsize=12)
fig.suptitle("Projection des embeddings Word2Vec", fontsize=16)
# fig.savefig(Path('.') / 'img' / 'word2vec_projection_relative.png', bbox_inches='tight')
```





## 4.3 Représentations des textes

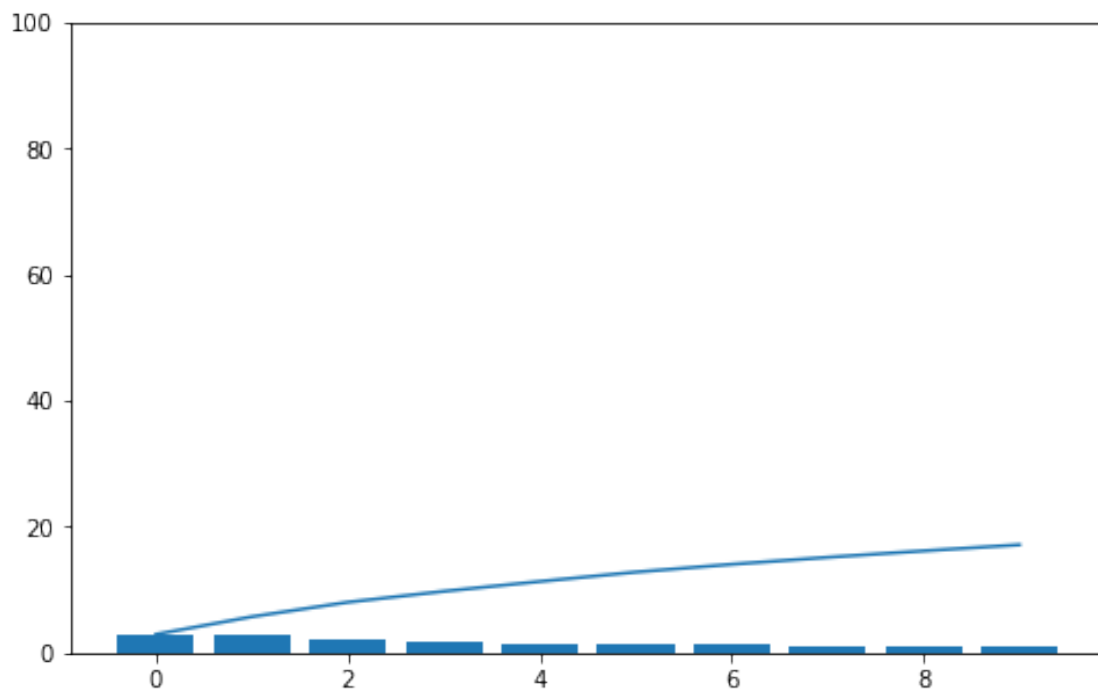
### 4.3.1 Bag of Words : comptes

Je fais une PCA sur les vecteurs de comptes. Mais d'abord, il faut standardiser.

```
[320]: vectorized_std = StandardScaler().fit_transform(np.array(vectorized_corpus.todense()))
PCA_model = PCA(n_components=10)
decomp = PCA_model.fit_transform(vectorized_std)
```

```
[321]: fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(PCA_model.explained_variance_ratio_.cumsum()*100)
ax.bar(x=range(10), height=PCA_model.explained_variance_ratio_*100)
ax.set_ylim(0, 100)
```

```
[321]: (0.0, 100.0)
```



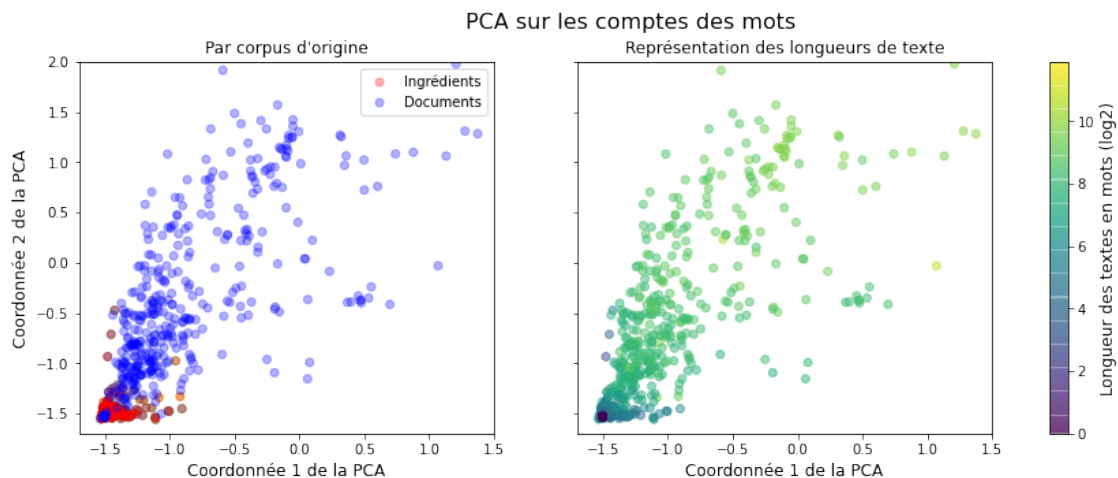
```
[322]: fig, ax = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
handles = []

ax[0].scatter(x=decomp[:500, 0],
              y=decomp[:500, 1],
              c=np.log2(np.array(vectorized_corpus[:500,:].sum(axis=1)).flatten() + 1),
              cmap='viridis',
              alpha=0.5)

handles.append(ax[0].scatter(x=decomp[:500, 0],
                             y=decomp[:500, 1],
                             c=['red']*500,
                             alpha=0.3))
handles.append(ax[0].scatter(x=decomp[500:, 0],
                             y=decomp[500:, 1],
                             c=['blue']*500,
                             alpha=0.3))

handles.append(ax[1].scatter(x=decomp[:, 0],
                             y=decomp[:, 1],
                             c=np.log2(np.array(vectorized_corpus[:, :].sum(axis=1)).flatten() + 1),
                             cmap='viridis',
                             alpha=0.5))

ax[0].set_xlim(-1.7, 1.5)
ax[0].set_ylim(-1.7, 2)
ax[1].set_xlim(-1.7, 1.5)
ax[1].set_ylim(-1.7, 2)
ax[0].legend(handles[:-1], ['Ingrédients', 'Documents'])
cb = fig.colorbar(handles[-1], ax=ax)
fig.axes[2].set_ylabel("Longueur des textes en mots (log2)", fontsize=12)
fig.suptitle("PCA sur les comptes des mots", fontsize=16)
ax[0].set_ylabel("Coordonnée 2 de la PCA", fontsize=12)
ax[0].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
ax[0].set_title("Par corpus d'origine", fontsize=12)
ax[1].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
ax[1].set_title("Représentation des longueurs de texte", fontsize=12)
# fig.savefig(Path('.') / 'img' / 'PCA_counts.png', bbox_inches='tight')
```



### 4.3.2 tSVD

Je commence par faire un tSVD sur le corpus, en différenciant les ingrédients du texte des fiches techniques. Dans ce premier cas, on a les comptes des mots dans la vectorisation.

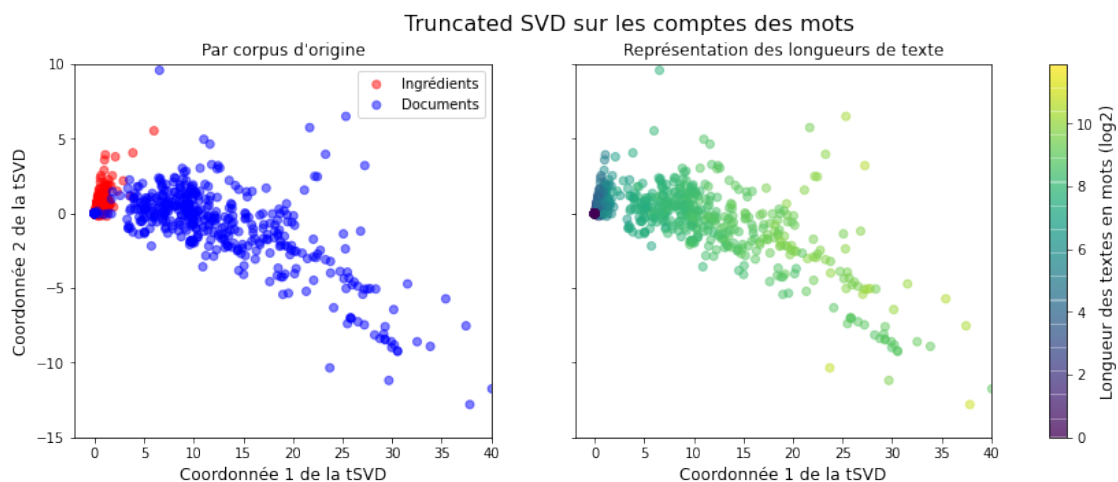
```
[323]: vectorized_corpus.todense()
```

```
[323]: matrix([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
               [ 0.,  0.,  0., ...,  0.,  0.,  0.],
               [ 0.,  0.,  0., ...,  0.,  0.,  0.],
               ...,
               [ 1.,  0.,  0., ...,  0.,  0.,  0.],
               [ 2.,  1.,  0., ...,  0.,  0.,  0.],
               [11.,  0.,  0., ...,  0.,  0.,  0.]])
```

```
[324]: transformer = TruncatedSVD(n_components=100)
projected = transformer.fit_transform(vectorized_corpus)
```

```
[325]: fig, ax = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
handles = []
handles.append(ax[0].scatter(x=projected[:500, 0], y=projected[:500, 1], c=['red']*500, alpha=0.5))
handles.append(ax[0].scatter(x=projected[500:, 0], y=projected[500:, 1], c=['blue']*500, alpha=0.5))
handles.append(ax[1].scatter(x=projected[:, 0],
                             y=projected[:, 1],
                             c=np.log2(np.array(vectorized_corpus[:, :].sum(axis=1)).flatten() + 1),
                             cmap='viridis',
                             alpha=0.5))

ax[0].set_ylim(-15, 10)
ax[0].set_xlim(-2, 40)
ax[1].set_ylim(-15, 10)
ax[1].set_xlim(-2, 40)
ax[0].legend(handles[:-1], ['Ingrédients', 'Documents'])
cb = fig.colorbar(handles[-1], ax=ax)
fig.axes[2].set_ylabel("Longueur des textes en mots (log2)", fontsize=12)
fig.suptitle("Truncated SVD sur les comptes des mots", fontsize=16)
ax[0].set_ylabel("Coordonnée 2 de la tSVD", fontsize=12)
ax[0].set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax[0].set_title("Par corpus d'origine", fontsize=12)
ax[1].set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax[1].set_title("Représentation des longueurs de texte", fontsize=12)
# fig.savefig(Path('.') / 'img' / 'tSVD_counts.png', bbox_inches='tight')
```



### 4.3.3 PCA : fréquences

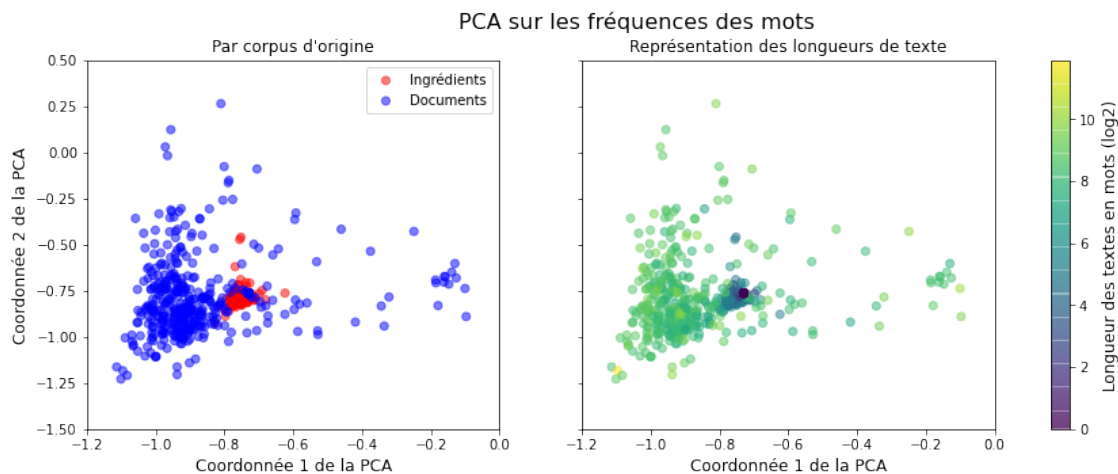
En utilisant cette fois la term frequency. PCA

```
[326]: tf_corpus_model = clone(tfidf_corpus)
tf_corpus = tf_corpus_model.set_params(norm='l1').fit_transform(corpus_df.loc[:, 'text'])

[327]: vectorized_std = StandardScaler().fit_transform(np.array(tf_corpus.todense()))
decomp = PCA(n_components=10).fit_transform(vectorized_std)

[332]: fig, ax = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
handles = []
handles.append(ax[0].scatter(x=decomp[:500, 0], y=decomp[:500, 1], c=['red']*500, alpha=0.5))
handles.append(ax[0].scatter(x=decomp[500:, 0], y=decomp[500:, 1], c=['blue']*500, alpha=0.5))
handles.append(ax[1].scatter(x=decomp[:, 0],
                             y=decomp[:, 1],
                             c=np.log2(np.array(vectorized_corpus[:, :].sum(axis=1)).flatten() + 1),
                             alpha=0.5,
                             cmap='viridis'))

ax[0].set_xlim(-1.2, 0)
ax[0].set_ylim(-1.5, 0.5)
ax[1].set_xlim(-1.2, 0)
ax[1].set_ylim(-1.5, 0.5)
ax[0].legend(handles[: -1], ['Ingrédients', 'Documents'])
cb = fig.colorbar(handles[-1], ax=ax)
fig.axes[2].set_ylabel("Longueur des textes en mots (log2)", fontsize=12)
fig.suptitle("PCA sur les fréquences des mots", fontsize=16)
ax[0].set_ylabel("Coordonnée 2 de la PCA", fontsize=12)
ax[0].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
ax[0].set_title("Par corpus d'origine", fontsize=12)
ax[1].set_xlabel("Coordonnée 1 de la PCA", fontsize=12)
ax[1].set_title("Représentation des longueurs de texte", fontsize=12)
fig.savefig(Path('.') / 'img' / 'PCA_freq.png', bbox_inches='tight')
```



### 4.3.4 tSVD : fréquences

Voilà.

```
[329]: transformer = TruncatedSVD(n_components=100)
projected = transformer.fit_transform(tf_corpus)

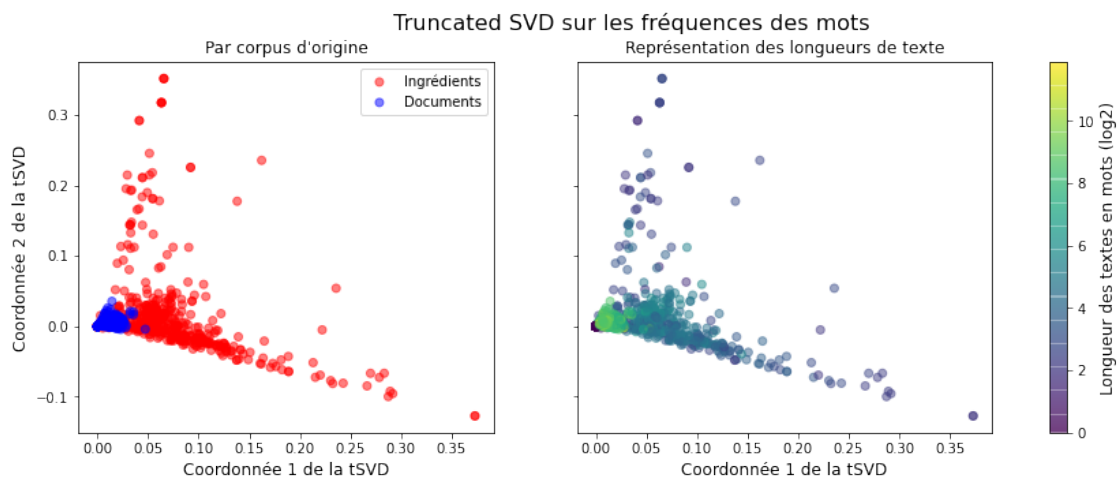
[330]: fig, ax = plt.subplots(ncols=2, figsize=(15, 5), sharey=True)
handles = []
handles.append(ax[0].scatter(x=projected[:500, 0], y=projected[:500, 1], c=['red']*500, alpha=0.5))
```

```

handles.append(ax[0].scatter(x=projected[500:, 0], y=projected[500:, 1], c=['blue']*500, alpha=0.5))
handles.append(ax[1].scatter(x=projected[:, 0],
                             y=projected[:, 1],
                             c=np.log2(np.array(vectorized_corpus[:, :].sum(axis=1)).flatten() + 1),
                             alpha=0.5,
                             cmap='viridis'))

# ax.set_ylim(-15, 10)
# ax.set_xlim(-2, 40)
ax[0].legend(handles, ['Ingrédients', 'Documents'])
cb = fig.colorbar(handles[-1], ax=ax)
fig.axes[2].set_ylabel("Longueur des textes en mots (log2)", fontsize=12)
fig.suptitle("Truncated SVD sur les fréquences des mots", fontsize=16)
ax[0].set_ylabel("Coordonnée 2 de la tSVD", fontsize=12)
ax[0].set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax[0].set_title("Par corpus d'origine", fontsize=12)
ax[1].set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax[1].set_title("Représentation des longueurs de texte", fontsize=12)
# fig.savefig(Path('.') / 'img' / 'tSVD_freq.png', bbox_inches='tight')

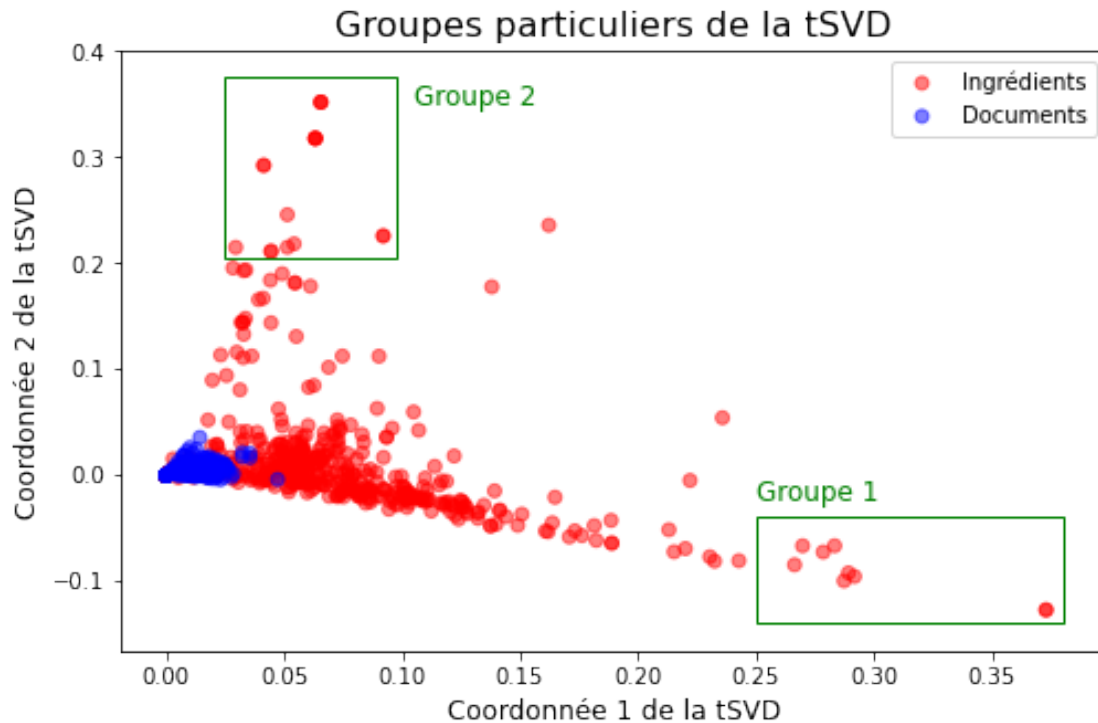
```



```

[433]: fig, ax = plt.subplots(figsize=(8, 5))
handles = []
handles.append(ax.scatter(x=projected[:500, 0], y=projected[:500, 1], c=['red']*500, alpha=0.5))
handles.append(ax.scatter(x=projected[500:, 0], y=projected[500:, 1], c=['blue']*500, alpha=0.5))
ax.add_patch(mpatch.Rectangle((0.25, -0.14), 0.13, 0.1, fill=False, color='green'))
ax.add_patch(mpatch.Rectangle((0.025, 0.205), 0.073, 0.17, fill=False, color='green'))
ax.annotate('Groupe 1', (0.25, -0.025), fontsize=12, color='green')
ax.annotate('Groupe 2', (0.105, 0.35), fontsize=12, color='green')
ax.legend(handles, ['Ingrédients', 'Documents'])
ax.set_ylabel("Coordonnée 2 de la tSVD", fontsize=12)
ax.set_xlabel("Coordonnée 1 de la tSVD", fontsize=12)
ax.set_title("Groupes particuliers de la tSVD", fontsize=16)
# fig.savefig(Path('.') / 'img' / 'tSVD_freq_groups.png', bbox_inches='tight')

```



```
[434]: sample = pd.concat([corpus_df.reset_index(), pd.DataFrame(projected[:, [0, 1]], columns=['x', 'y'])],
↳axis=1)
```

```
[435]: idx1 = ((projected[:, 0] > 0.25) &
              (projected[:, 0] < (0.25+0.13)) &
              (projected[:, 1] < (-0.14+0.1)) &
              (projected[:, 1] > -0.14))
idx2 = ((projected[:, 0] > 0.025) &
        (projected[:, 0] < (0.025+0.073)) &
        (projected[:, 1] < (0.205+0.17)) &
        (projected[:, 1] > 0.205))

sample.loc[idx1, 'group'] = 'Groupe 1'
sample.loc[idx2, 'group'] = 'Groupe 2'
with pd.option_context("max_colwidth", 100000):
    tex_str = (sample.loc[idx1 | idx2]
               .set_index('group')
               .sort_index()
               .loc[:, ['text', 'x', 'y']]
               .rename({'text': 'Texte'}, axis=1)
               .to_latex(multirow=True,
                          column_format='llcc',
                          index_names=False,
                          )
               .replace(r'\textbackslash n', r' \newline '))
    print(tex_str)
# with open(Path('.') / 'tbls' / 'tSVD_sample.tex', mode='w') as file:
#     file.write(tex_str)
```

```
\begin{tabular}{llcc}
\toprule
{} & Texte & x & y \\
\midrule
Groupe 1 & Thon, eau, sel & 0.373079 & -0.127052
```

Groupe 1 &	Pois chiches, eau, sel. &	0.266423 &	-0.084501 \\	
Groupe 1 &	Haricots beurre, eau, sel &	0.283409 &	-0.066561 \\	
Groupe 1 &	Câpres, eau, vinaigre, sel &	0.291819 &	-0.095479 \\	
Groupe 1 &	Thon albacore, eau, sel &	0.287423 &	-0.099627 \\	
Groupe 1 &	Eau, haricots verts, sel. &	0.278588 &	-0.072474 \\	
Groupe 1 &	Pommes de terre, eau, sel. &	0.289291 &	-0.092162 \\	
Groupe 1 &	Carottes rondelles, eau, sel, sucre &	0.269849 &	-0.066600 \\	
Groupe 1 &	Thon, eau, sel &	0.373079 &	-0.127052 \\	
Groupe 2 &	- 100\% Semoule de blé dur de qualité supérieure &	0.065551 &	0.351800 \\	
Groupe 2 &	100\% haricots blancs &	0.051373 &	0.214921 \\	
Groupe 2 &	100\% Semoule de blé dur de qualité supérieure &	0.065551 &	0.351800 \\	
Groupe 2 &	Semoule de blé dur* \newline *issu de l'agriculture biologique &	0.054211 &		
0.218192 \\				
Groupe 2 &	Tilleul (100\%). &	0.041323 &	0.292199 \\	
Groupe 2 &	100\% semoule de blé dur de qualité supérieure &	0.065551 &	0.351800 \\	
Groupe 2 &	Farine de blé T45 &	0.092002 &	0.225689 \\	
Groupe 2 &	- Semoule de blé dur de qualité supérieure \newline - 30\% oeufs frais &	0.044446 &		
0.211188 \\				
Groupe 2 &	- 100\% Semoule de blé dur de qualité courante \newline - Contient du gluten &	0.051337 &		
0.245649 \\				
Groupe 2 &	SEMOULE DE BLE' DUR de qualité supérieure &	0.063201 &	0.317653 \\	
Groupe 2 &	100\% Arabica &	0.041324 &	0.292222 \\	
Groupe 2 &	agar-agar 100\% &	0.029466 &	0.214845 \\	
Groupe 2 &	SEMOULE DE BLE' DUR de qualité supérieure &	0.063201 &	0.317653 \\	
Groupe 2 &	SEMOULE DE BLE' DUR de qualité supérieure &	0.063201 &	0.317653 \\	
Groupe 2 &	SEMOULE DE BLE' DUR de qualité supérieure &	0.063201 &	0.317653 \\	
Groupe 2 &	- Semoule de blé dur de qualité supérieure \newline - 30\% oeufs frais &	0.044446 &		
0.211188 \\				
Groupe 2 &	Farine de blé T45 &	0.092002 &	0.225689 \\	
\bottomrule				
\end{tabular}				

[ ]: