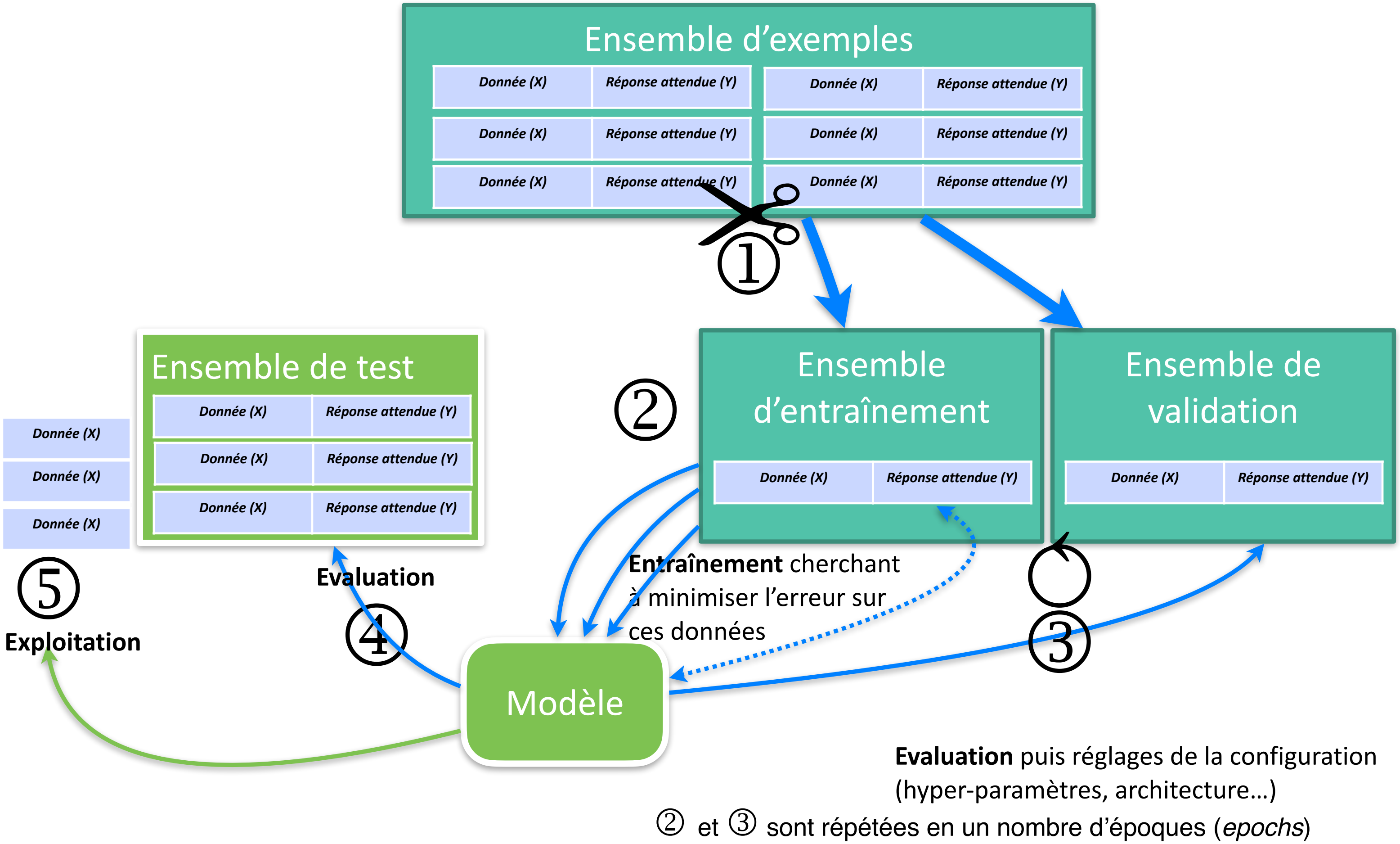
APPRENTISSAGE AUTOMATIQUE POUR LA CLASSIFICATION TEXTUELLE

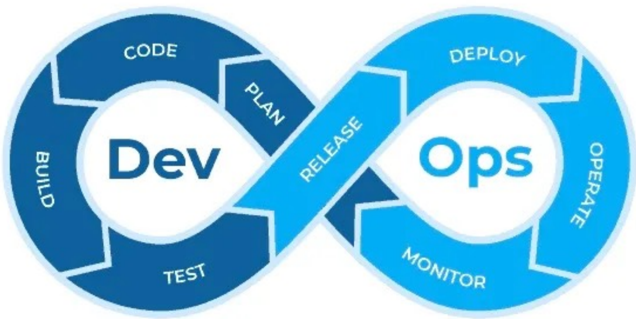
Atelier ANF TDM - Exploration documentaire et extraction d'information
octobre 2022

Patrice Bellot
Aix-Marseille Université - CNRS (LIS-INS2I)

patrice.bellot@univ-amu.fr

Apprentissage automatique, modèle, évaluation



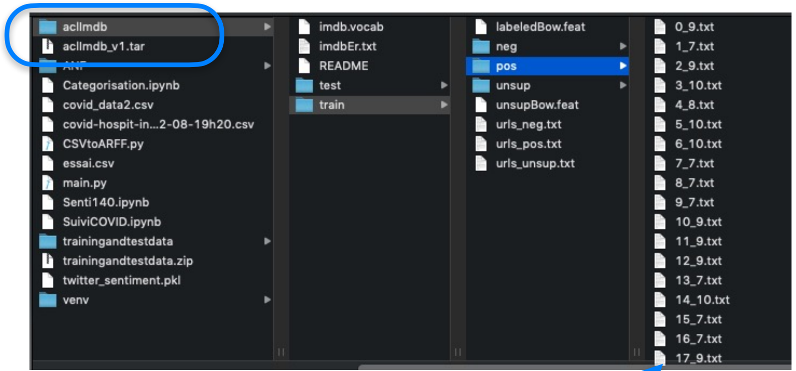


https://datascientest.com/mlops-le-devops-applique-aux-projets-de-machine-learning

ANALYSE DE SENTIMENT (POLARITE) SUR DES CRITIQUES DE FILMS

Large Movie Review Dataset

http://ai.stanford.edu/~amaas/data/sentiment/



Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* (pp. 142-150).

4.3.2 IMDB Review Dataset

We constructed a collection of 50,000 reviews from IMDB, allowing no more than 30 reviews per movie. The constructed dataset contains an even number of positive and negative reviews, so randomly guessing yields 50% accuracy. Following previous work on polarity classification, we consider only highly polarized reviews. A negative review has a score ≤ 4 out of 10, and a positive review has a score ≥ 7 out of 10. Neutral reviews are not included in the dataset. In the interest of providing a benchmark for future work in this area, we release this dataset to the public.[2]

Corpus d'entraînement (train) : 12 500 positives, 12 500 négatives

```
This is a complex film that explores the effects of Fordist and Taylorist modes of industrial capitalist production on human relations. There are const
 references to assembly line production, where workers are treated as cogs in a machine, overseen by managers wielding clipboards, controlling how much
ir the workers leave exposed, and firing workers (Stanley) who meet all criteria (as his supervisor says, are always on time, are hard workers, do good
rk) but who may in some unspecified future make a mistake. <br /><br />This system destroys families - Stanley has to send his father to a nursing home
here he quickly dies) after Stanley loses his job. Iris' daughter is a single teen mother who drops out of high school to take a job in the plant. Refe
ces are made to the fact that now, with declining wages, both partners need to work, the implication being that there's nobody left at home to care for
e kids. Iris' husband is dead from an illness, and with the multiple references in the film about the costs of medical care, the viewer must wonder if
might have lived with better and more costly care. Iris' brother in law gets abusive after yet another unsuccessful day at the unemployment office when
s wife yells at him for buying a beer with her savings instead of leaving it for her face lift and/or teeth job (even the working class with no stake i
onventional bourgeois notions of perfection and beauty buy into them). The one reference to race in the film is through a black factory line worker who
husband is in jail (presumably, he's also black, and black men suffer disproportionally high incarceration rates). She remarks that he, like her, "is d
g time" - her family is composed of a prisoner and a wage slave.<br /><br />Stanley, however, still believes in human relations and is therefore for mo
of the film outside of the system of Fordist capitalism. He cares for his father in spite of the fact that it was his father's traveling salesman job t
 resulted in his illiteracy - he has not yet reduced human relations to a purely instrumental contract, as Iris' brother in law does (suggesting that h
married the wrong sister"). He does not, as Iris says, conform to the work-eat-sleep routine of everyone else; rather, he uses technology and the techn
es of industrial production in an artisanal and creative way, in a sort of Bauhaus ideal. This was the dream of early modernists and 1920's socialists
```

Le plus simple : utiliser des modules existants

- 0,8

```
from textblob import TextBlob

print(TextBlob("I hate that movie").sentiment.polarity)

texte = "This is a gem. As a Film Four production - the anticipated quality was indeed delivered. Shot with great style that reminded me some Errol Morris films, well arranged and simply gripping. It's long yet horrifying to the point it's excruciating. We know something bad happened (one can guess by the lack of participation of a person in the interviews) but we are compelled to see it, a bit like a car accident in slow motion. The story spans most conceivable aspects and unlike some documentaries did not try and refrain from showing the grimmer sides of the stories, as also dealing with the guilt of the people Don left behind him, wondering why they didn't stop him in time. It took me a few hours to get out of the melancholy that gripped me after seeing this very-well made documentary."
print(TextBlob(texte).sentiment.polarity)
```

- 0,054

…. quelle analyse ? quelle performance en moyenne ? comment l'améliorer ?

Pré-traitements du corpus

La première étape consiste à intégrer l'ensemble des critiques annotées (polarité négative ou positive) en un seul fichier au format CSV qui pourra être stocké en mémoire par un DataFrame (extension Pandas de Python).

```
# Conversion du corpus d'origine en un fichier .csv

import pandas as pd
import os

repertoire_depart = '/Users/Patrice/PycharmProjects/ANF2021/aclImdb'

labels = {'pos':1, 'neg' : 0}
df = pd.DataFrame()
for f in ('test', 'train'):
    for l in ('pos', 'neg'):
        path = os.path.join(repertoire_depart, f, l)
        for fichier in os.listdir(path):
            with open(os.path.join(path, fichier), 'r', encoding='utf-8') as infile:
                txt = infile.read()
            df = df.append([[txt, labels[l]]], ignore_index=True)
df.columns=['review', 'polarity']

df.to_csv('movie_data.csv', index=False, encoding='utf-8')
df.head()
```

	review	polarity
0	Based on an actual story, John Boorman shows t...	1
1	This is a gem. As a Film Four production - the...	1
2	I really like this show. It has drama, romance...	1
3	This is the best 3-D experience Disney has at ...	1
4	Of the Korean movies I've seen, only three had...	1

taille du fichier movie_dataset.csv : 65,9 Mo (50 000 lignes, 14 millions de *tokens*, 194 758 mots différents) les tokens les plus fréquents :

```
['the', ',', '.', 'a', 'and', 'of', 'to', 'is', '/', '>', '<', 'br', 'in', 'I', 'it', 'that', "'s", 'this', 'was', 'The', 'as',
 'with', 'movie', 'for', 'film', ')', '(', 'but', "''", "n't", '``', 'on', 'you', 'are', 'not', 'have', 'his', 'be', '!', 'he',
 'one', 'at', 'by', 'an', 'all', 'who', 'they', 'from', 'like', 'It']
```

100 749 mots n'apparaissent qu'une fois :

```
themeparks
Disney-MGM
artistically-inclined
conscience-less
monsieur
non-lonely
upsetting.
finger-sewing
boondoggling
Bathian
moneygrubbing
smarmy.
Olivier/Garson
cold-fish
highlife
Marchionesse
Udolpho
frazzled.
bunt
Lorelay
obsesion
adverterous
Giraurd
Schlater
MissCastaway.com
UNBELIEVABLE
Coober
Pedy
Docudrama
'Cobra
'Renegade
Farsape
mega-makeup
puppet/digital
Hynerian
Sebaceans
irreversiby
Crichton.
starburst
Hillarious
unfortuatley
dissapeared
```

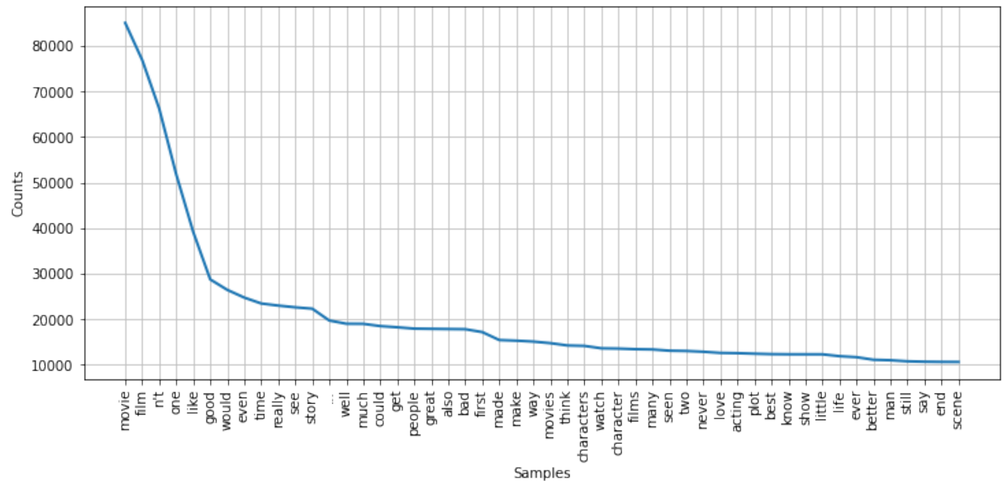
```python
hapax = frequency_dist.hapaxes()
```

'the' apparait 573 397 fois

```
{'the': 573397, ',': 544031, '.': 467886, 'and': 309118, 'a': 309103, 'of': 285087, 'to': 263658, 'is': 214740
```

les tokens les plus fréquents après suppressions des mots outils :

```python
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
tokens = [w.lower() for w in tokens if not w.lower() in stop_words and len(w)>2]
```

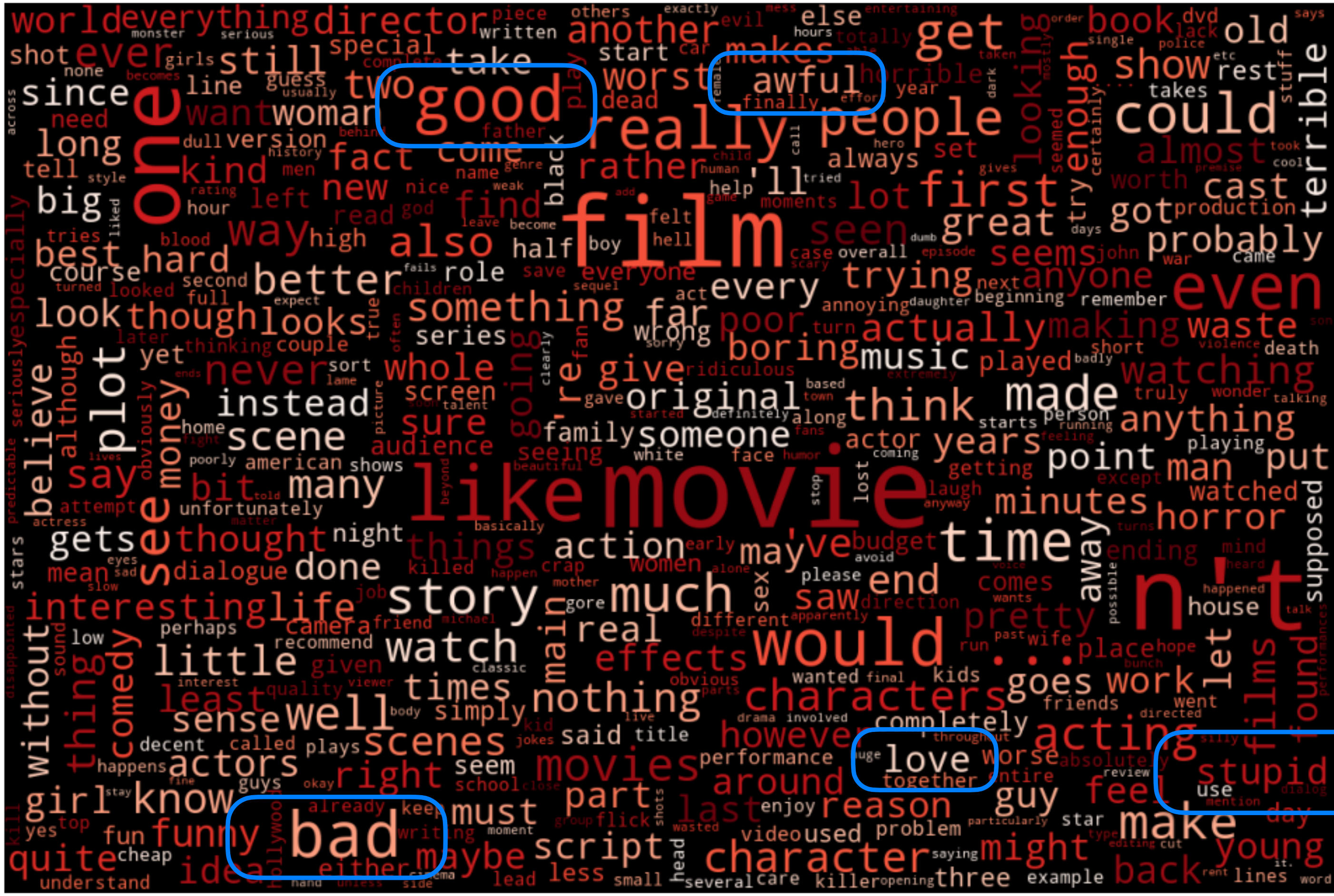


```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt
wordcloud = WordCloud(width=1200, height=800,
                      max_words=500,
                      max_font_size=100,
                      relative_scaling=0.5,
                      colormap='Blues',
                      normalize_plurals=True).generate_from_frequencies(frequency_dist)
plt.figure(figsize=(17,14))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

Toutes les critiques réunies

Les critiques positives

Les critiques négatives



Avec un classifieur bayésien naïf (NB)

```
#%%
X_train = df.loc[:24999, 'review'].to_numpy()
# Return a Numpy representation of the DataFrame
# Only the values in the DataFrame will be returned, the axes labels will be removed
y_train = df.loc[:24999, 'polarity'].to_numpy()
X_test = df.loc[25000:, 'review'].to_numpy()
y_test = df.loc[25000:, 'polarity'].to_numpy()
```

Division des exemples :
50 % entraînement (*train*)
50 % test

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(lowercase=False, max_features=10000)
train_vectors = vectorizer.fit_transform(X_train)
test_vectors = vectorizer.transform(X_test)
print(train_vectors.shape, test_vectors.shape)

#%%
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(train_vectors, y_train)

#%%
from  sklearn.metrics  import accuracy_score
from  sklearn.metrics  import classification_report
predicted = clf.predict(test_vectors)
print("Global Accuracy :", accuracy_score(y_test,predicted))
print(classification_report(y_test, predicted))
```

Evaluation (classes 0 et 1)

données de test

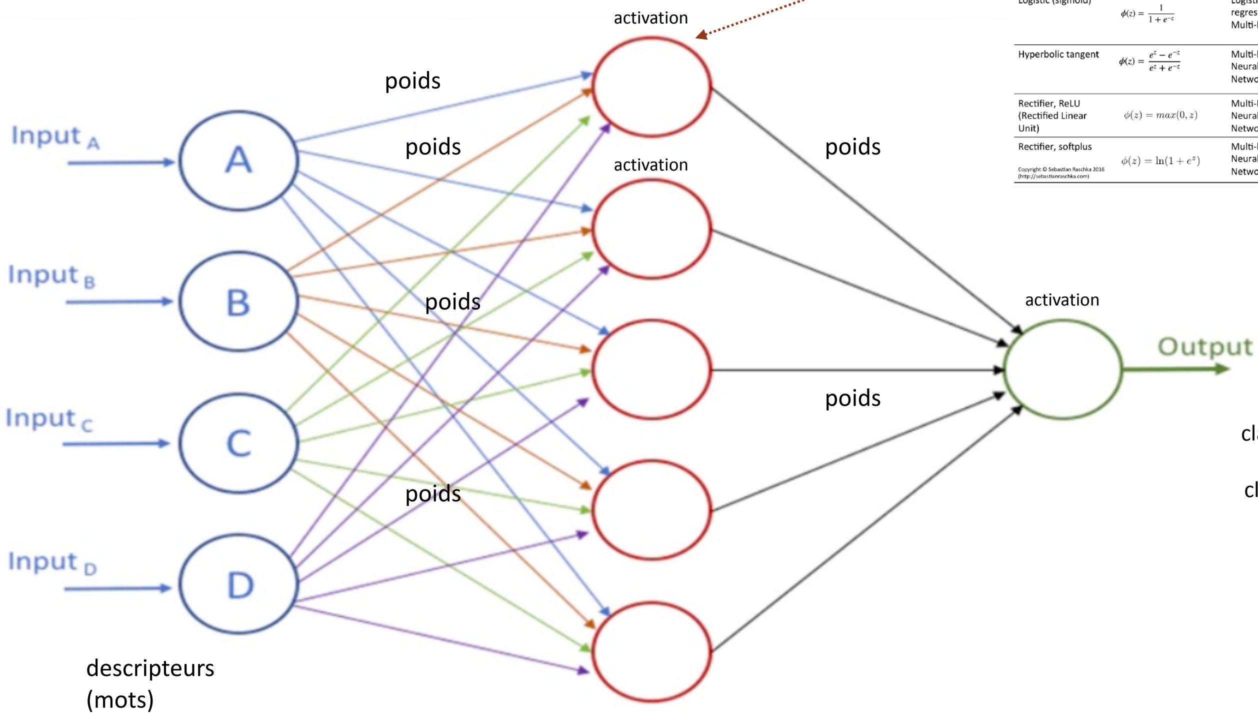
```
Global Accuracy : 0.8454
              precision    recall  f1-score   support

           0       0.84      0.86      0.85     12500
           1       0.85      0.83      0.84     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000
```

$$accuracy(Y, \widehat{Y}) = \frac{1}{n_{exemples}} \sum_i 1(\widehat{y_i} = y_i)$$

classes réelles — classes prédites

Réseau de neurones

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

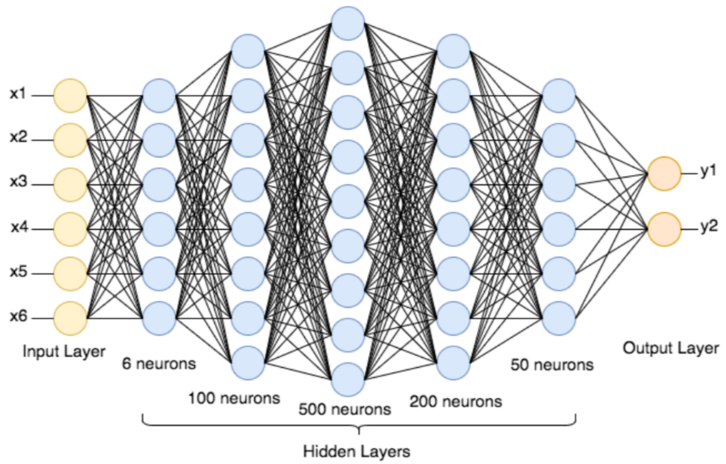


0 ou 1

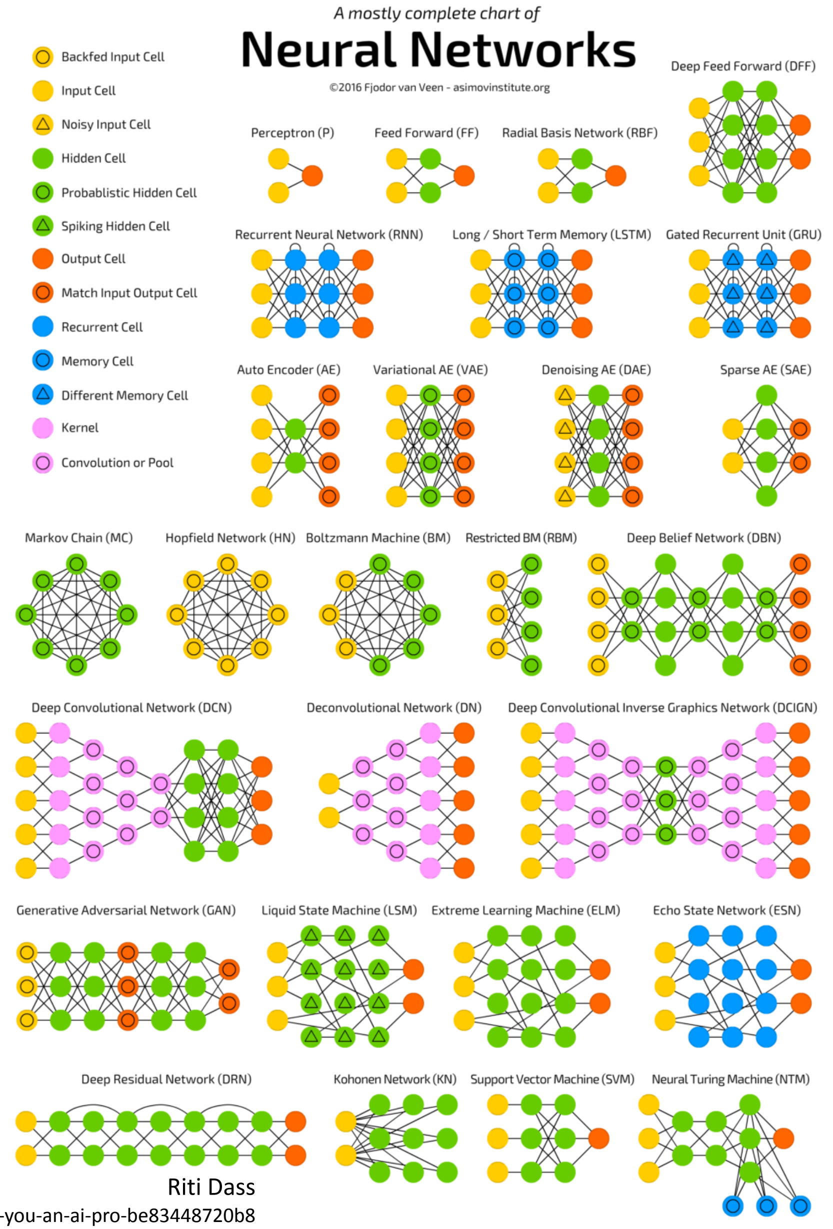
classe « négative »
vs
classe « positive »

Apprentissage de relations (non linéaires) entre les entrées et la sortie

Architectures neuronales



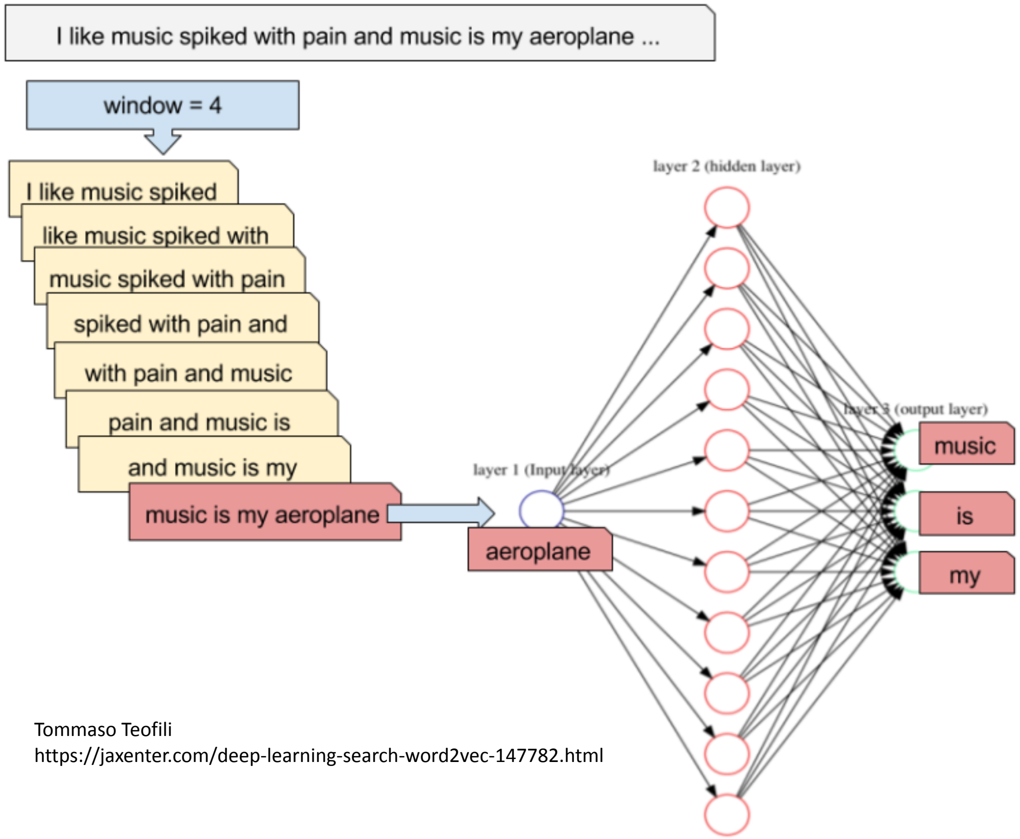
Deep Learning for Ligand-Based Virtual Screening in Drug Discovery
October 2018
DOI: 10.1109/PAIS.2018.8598488
Conference: 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)
Meriem BahiMohamed Batouche



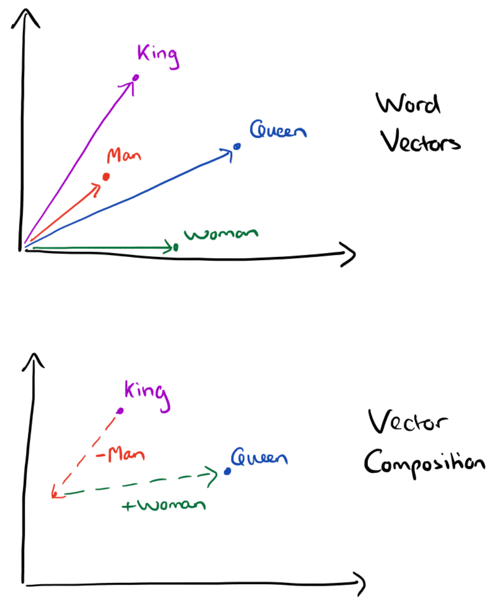
Riti Dass

https://medium.com/predict/the-complete-list-to-make-you-an-ai-pro-be83448720b8

Réduction de la dimension (projection)
Les plongements de mots (word embeddings)



Tommaso Teofili
https://jaxenter.com/deep-learning-search-word2vec-147782.html



Adrian Colyer
https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Distributed Representations of Words and Phrases and their Compositionality – Mikolov et al. 2013

Efficient Estimation of Word Representations in Vector Space – Mikolov et al. 2013

Apprentissage de représentation Word2Vec

```python
#L'espace de réprésentation est appris sur l'ensemble du corpus
for line in df['review']:
    tokens = word_tokenize(line)
    stop_words = set(stopwords.words('english'))
    tokens = [w.lower() for w in tokens if w.isalpha() and len(w)>1 and not w.lower() in stop_words]
    review_lines.append(tokens)
```

```python
import gensim
model = gensim.models.Word2Vec(sentences=review_lines, size=200, window=5, workers=4, min_count=1)
motsComplet = list(model.wv.vocab)
```

```
movie -1.3216566 0.36635584 -0.28186616 -1.0511837 -1.0501945 -1.7482823 -0.42692444 0.16830114 -1.073119 -1.5651205 -
96654 -0.54516864 1.2929311 0.49605948 1.1482662 0.38361785 -0.30000296 0.78807664 -0.62371856 -1.5082116 -0.13787036 -
74925745 0.41954425 0.35796735 0.3195898 -0.20374134 -0.25748256 -0.90302813 -0.44684523 -0.46419883 0.43331063 0.3801
-0.23262957 -0.57022005 -0.6890808 0.29229978 -0.06665888 -0.045591816 -0.31439704 -0.44238204 -1.19862 0.12611166 0.9
1796 0.17370766 0.20563798 0.8580158 0.8143437 -0.026487244 -0.12953776 1.6001002 0.2723402 0.053601284 0.440381 0.058
9 -0.14719109 -0.3533582 -1.16035 1.0383319 0.3641711 -0.29797938 -0.041548226 0.35354558 -0.7025537 0.17
1.3149184 0.21495351 -0.7291604 0.18647747 -1.2000268 -0.51228637 0.36612657 -0.25129464 -0.746 -0.183673
6096396 0.34609687 0.40593633 0.7030198 0.023112642 -0.9067271 0.43155307 0.4280309 -0.049969178 -0.67905
.6962609 0.16017178 0.66016424 -0.5926901 -0.013376136 -0.22369754 -1.0953285 -0.56589377 -0.42723322 0.7
673262 0.8491248 -0.484025 -0.31997883 0.18664318 -0.5761222 0.33220634 -1.0463667 -0.009183551 0.547165
037895 -1.0772457 -0.646116 1.1264194 -0.9413773 0.08854891 -0.122176886 -0.056594223 0.5072317 1.13529 -
88807 0.37230954 -0.61006385 -1.1492089 -1.5274029 -0.037806857 -0.19853547 0.2762417 -0.9356259 -0.37737
film -1.1342325 0.5424572 0.0140855415 -0.54681146 -1.0229077 -2.3149817 -0.3617721 0.08117554 -0.69557166 -1.0018283 -
25 0.1879876 1.2258501 0.53333026 0.71119124 -1.3764403 -0.69352823 0.67989963 0.049601056 -1.0814724 -0.17875 -0.2995
0.46457902 0.110982075 0.07333746 -1.321096 -0.19277126 0.023522813 -0.31523454 -0.23818257 0.4992599 0.20365019 0.210
204 0.43208042 0.03197141 0.19413853 -0.32528928 -0.14852582 0.12936993 0.068569176 -0.36599588 0.116247706 0.68026376
314871 -0.30278912 0.69517577 0.4294458 -0.3990693 -0.76446646 1.5112543 0.3708154 0.11746891 0.701029 -0.7823005 1.62
8 -0.26889926 1.0239882 -1.2052739 -0.047914516 0.9869529 -0.46331605 -0.07111113 0.079658456 0.37919065 -0.006453751 -
45773625 -0.58498067 0.45197055 -0.49910277 0.317274 -0.90511173 0.42767948 0.22158863 -0.068598926 0.58532935 -0.0108
21 2.0317261 0.7017311 0.12857646 1.0322477 0.30594614 0.5822884 -1.2792618 -0.27707702 0.5073626 0.5156112 -0.7731857
63963 -0.25596482 0.66147095 -0.007577596 -1.0135919 -0.37657994 0.21909198 -1.2694278 -0.758413 -0.9453872 -0.235683
5475771 0.36981234 0.29823944 -0.37622204 0.22047852 0.2637362 -1.1235323 0.12577608 -0.56808615 -0.49570698 0.290590
0.37622717 0.11014897 -1.2906862 0.10878839 0.9532716 -0.9014037 -0.41337353 0.57484233 -0.76305604 0.26593128 0.2917
```

pour chaque mot, un vecteur en 200 dimensions

Le modèle Word2Vec appris sur les critiques

```
#### Enregistrement du modèle appris

nomEmbeddings = 'imdb_embeddings_word2vec_200_5_100'
model.wv.save_word2vec_format(nomEmbeddings, binary=False)
```

```
print("Taille du vocabulaire : ", len(motsComplet))
print("Les mots les plus proches de horrible sont :")
model.wv.most_similar('horrible')
#%%
print("Les mots les plus proches de superb sont :")
model.wv.most_similar('superb')
```

```
Taille du vocabulaire :  96855
Les mots les plus proches de horrible sont :
[('terrible', 0.9239196181297302),
 ('awful', 0.8446447849273682),
 ('horrendous', 0.7841349840164185),
 ('pathetic', 0.7593107223510742),
 ('sucks', 0.7501437664031982),
 ('atrocious', 0.744674563407898),
 ('dreadful', 0.7377941012382507),
 ('horrid', 0.7361111640930176),
 ('lousy', 0.7139973640441895),
 ('ridiculous', 0.7078706622123718)]
```

```
Les mots les plus proches de superb sont :
[('outstanding', 0.8770316243171692),
 ('exceptional', 0.8604843616485596),
 ('excellent', 0.8578838109970093),
 ('terrific', 0.8461805582046509),
 ('fabulous', 0.8225735425949097),
 ('fantastic', 0.817896842956543),
 ('splendid', 0.8140406608581543),
 ('phenomenal', 0.8114627599716187),
 ('marvelous', 0.8058702945709229),
 ('impeccable', 0.788905680179596)]
```

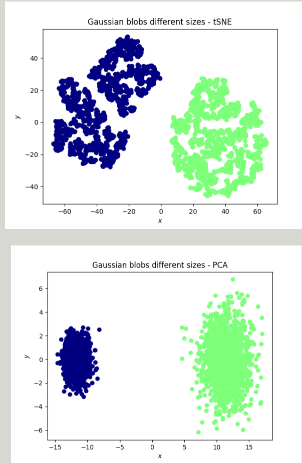
Visualisation des plongements (2 dimensions)

allure générale conservée (variance)

voisinage conservé (distance)

Table of Difference between PCA and t-SNE

S.NO.	PCA	t-SNE
1.	It is a linear Dimensionality reduction technique.	It is a non-linear Dimensionality reduction technique.
2.	It tries to preserve the global structure of the data.	It tries to preserve the local structure(cluster) of data.
3.	It does not work well as compared to t-SNE.	It is one of the best dimensionality reduction technique.
4.	It does not involve Hyperparameters.	It involves Hyperparameters such as perplexity, learning rate and number of steps.
5.	It gets highly affected by outliers.	It can handle outliers.
6.	PCA is a deterministic algorithm.	It is a non-deterministic or randomised algorithm.
7.	It works by rotating the vectors for preserving variance.	It works by minimising the distance between the point in a guassian.
8.	We can find decide on how much variance to preserve using eigen values.	We cannot preserve variance instead we can preserve distance using hyperparameters.



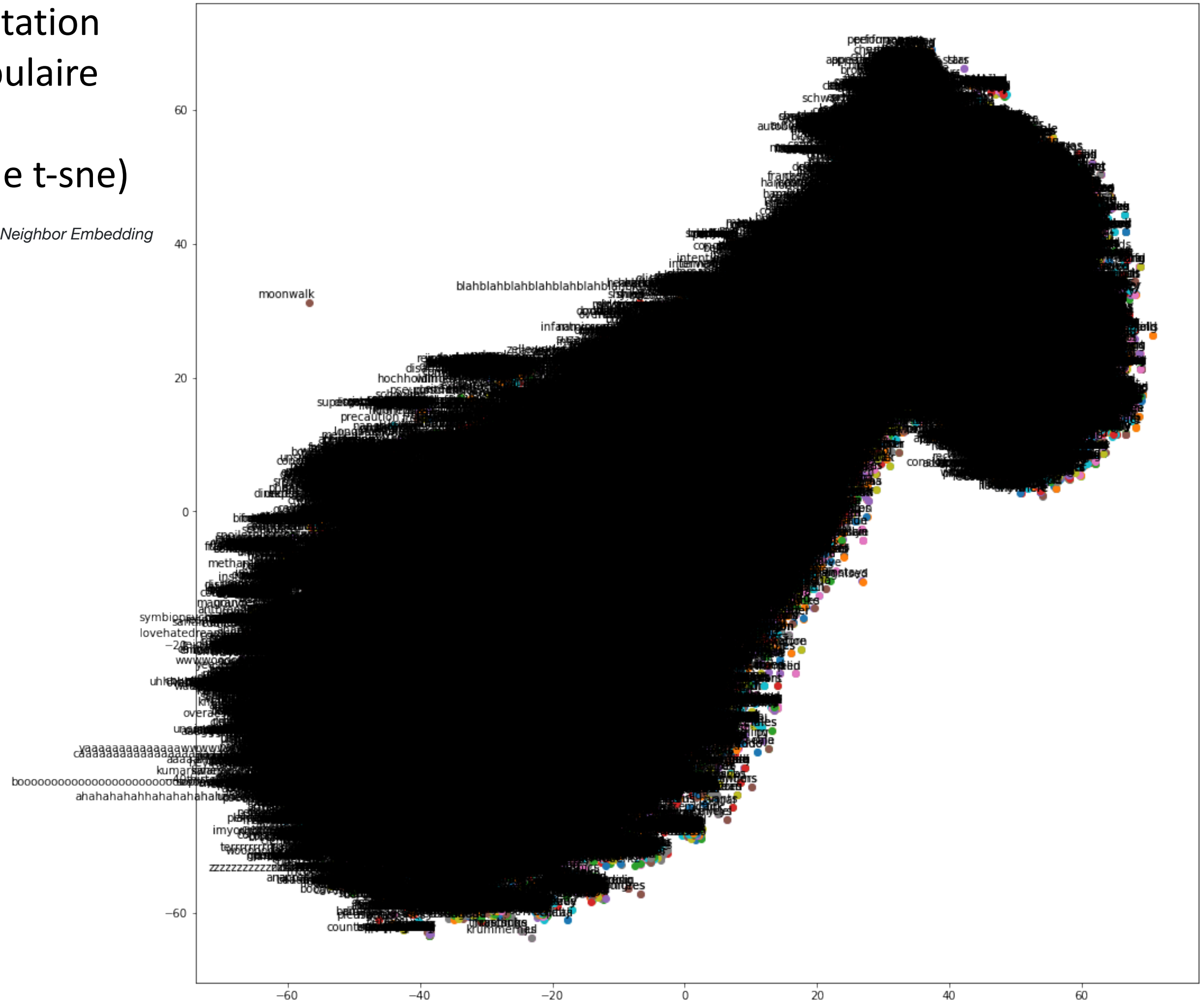
https://stats.stackexchange.com/questions/238538/are-there-cases-where-pca-is-more-suitable-than-t-sne

https://www.geeksforgeeks.org/difference-between-pca-vs-t-sne/

https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b

représentation
du vocabulaire
complet
(approche t-sne)

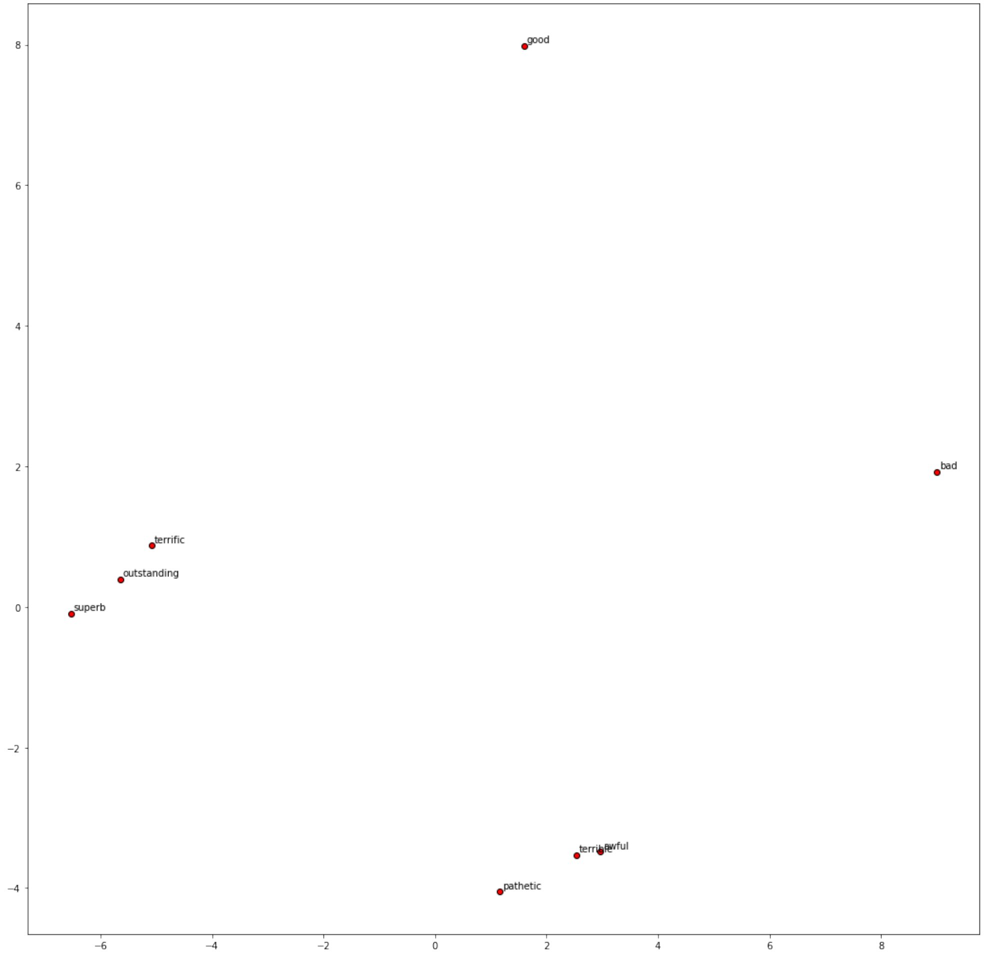
t-distributed Stochastic Neighbor Embedding



env. 30 s / itération
(1 seul cœur)

```
tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500, random_state=23)
```

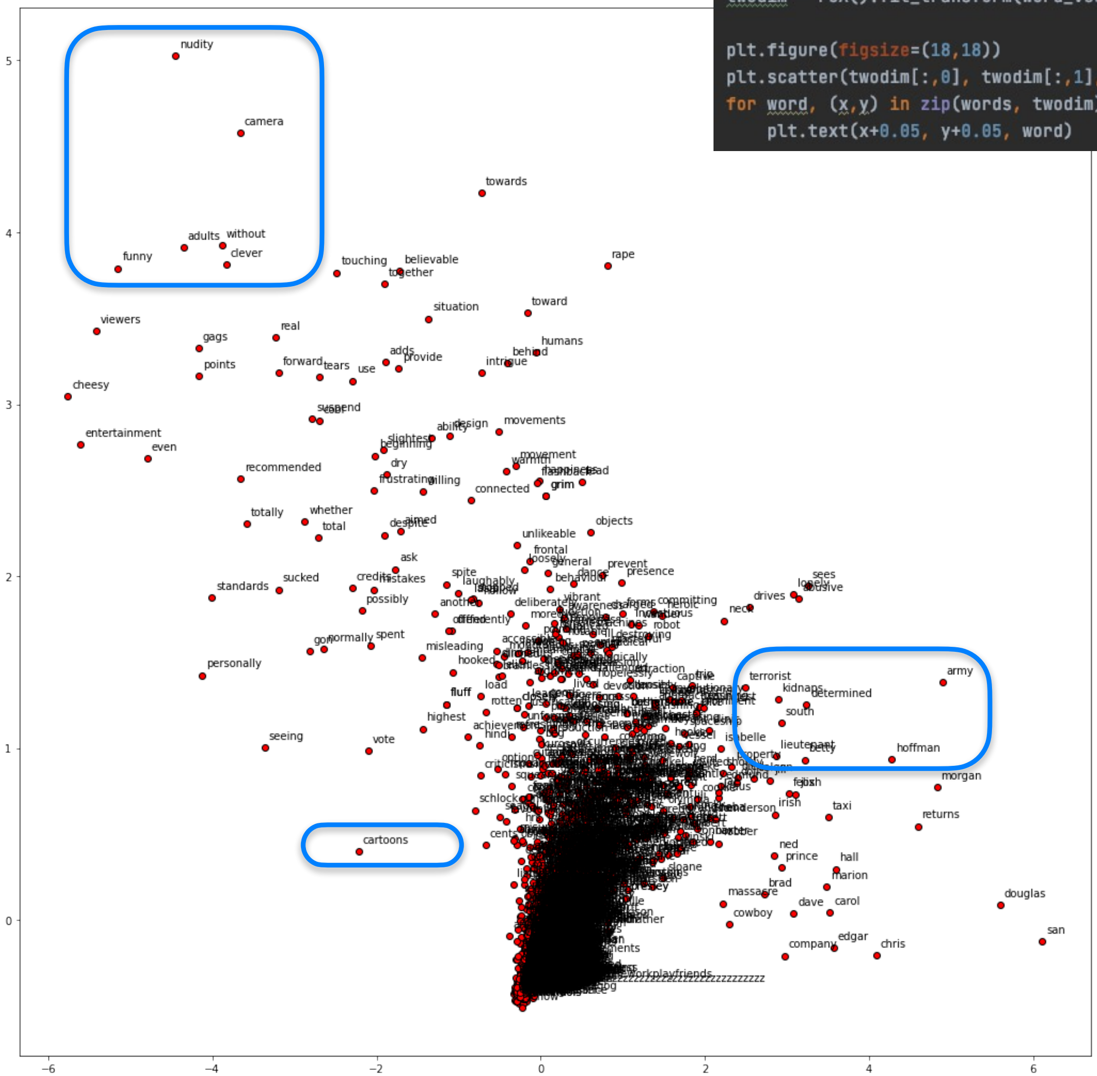
```
display_pca_scatterplot(model, ['superb', 'good', 'terrible', 'awful', 'pathetic', 'outstanding', 'terrific', 'bad'])
```



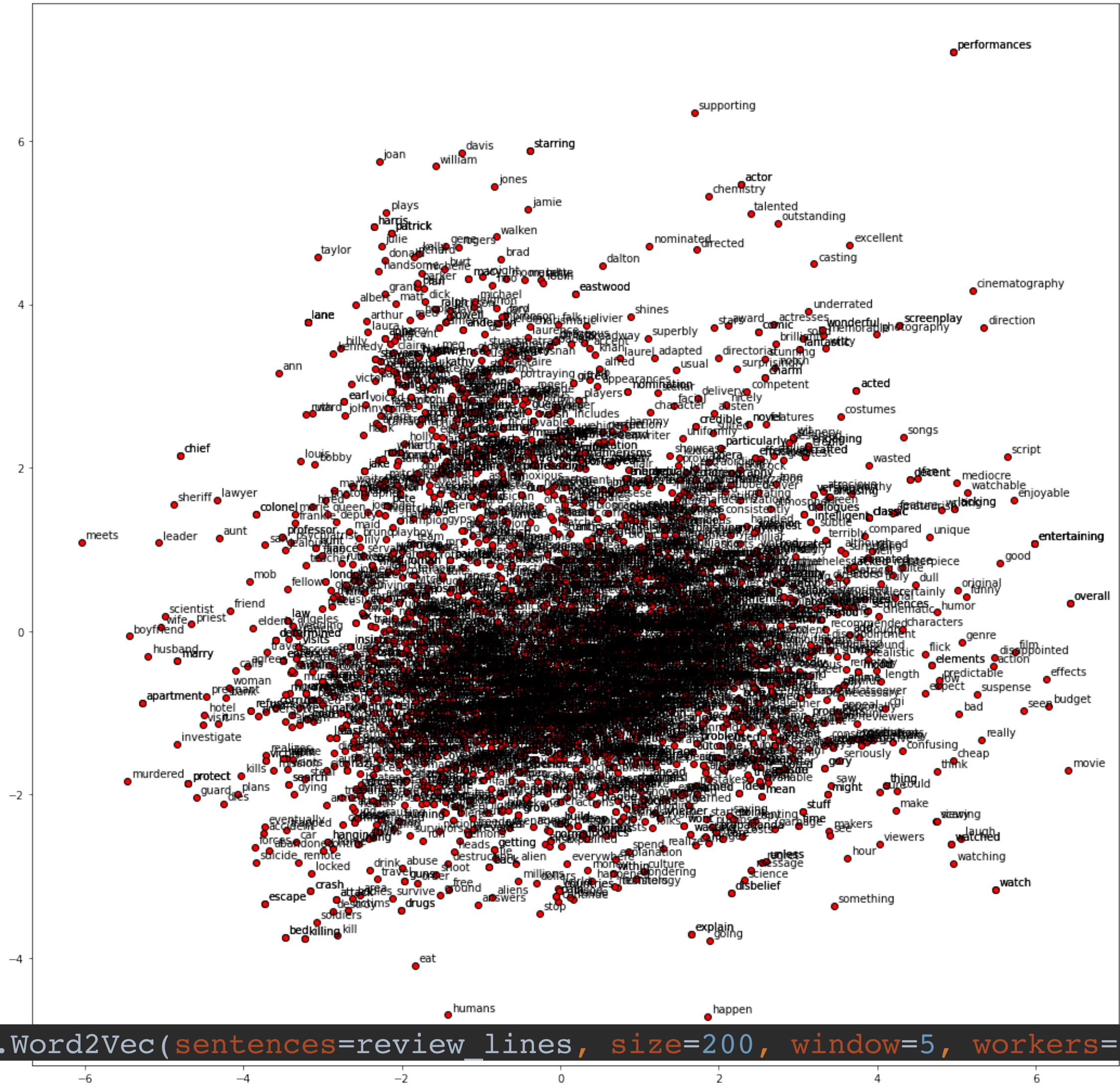
ACP
de 3000
mots pris
au hasard

https://web.stanford.edu/class/cs224n/materials/Gensim%20word%20vector%20visualization.html

```
twodim = PCA().fit_transform(word_vectors)[:,:2]

plt.figure(figsize=(18,18))
plt.scatter(twodim[:,0], twodim[:,1], edgecolors='k', c='r')
for word, (x,y) in zip(words, twodim):
    plt.text(x+0.05, y+0.05, word)
```

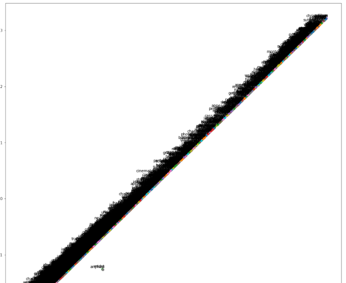


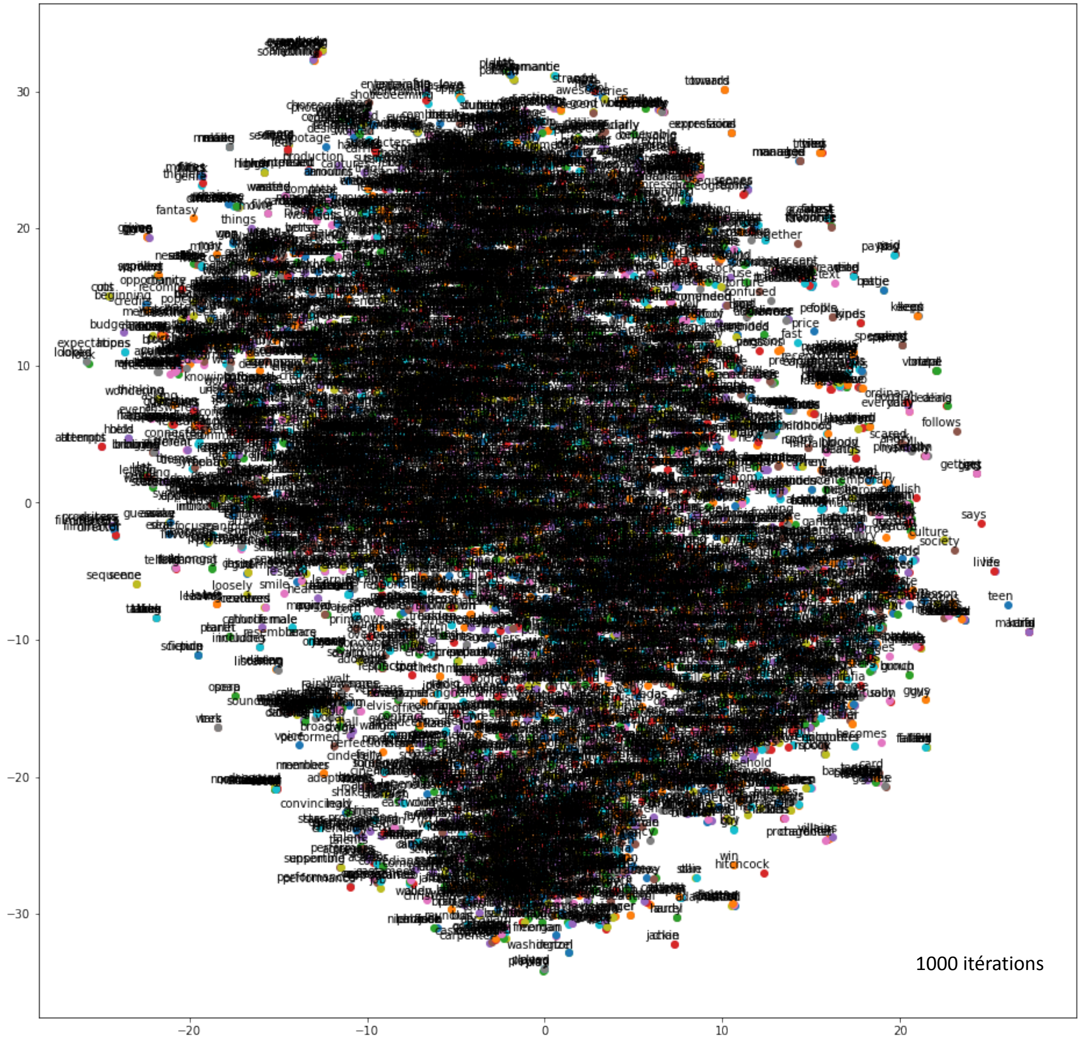
ACP
de 3000
mots pris
au hasard
<u>en limitant</u>
<u>Word2Vec</u>
<u>aux mots ayant</u>
<u>au moins</u>
<u>100 occurrences</u>
(soit 6561 mots)




```
model = gensim.models.Word2Vec(sentences=review_lines, size=200, window=5, workers=4, min_count=100)
```

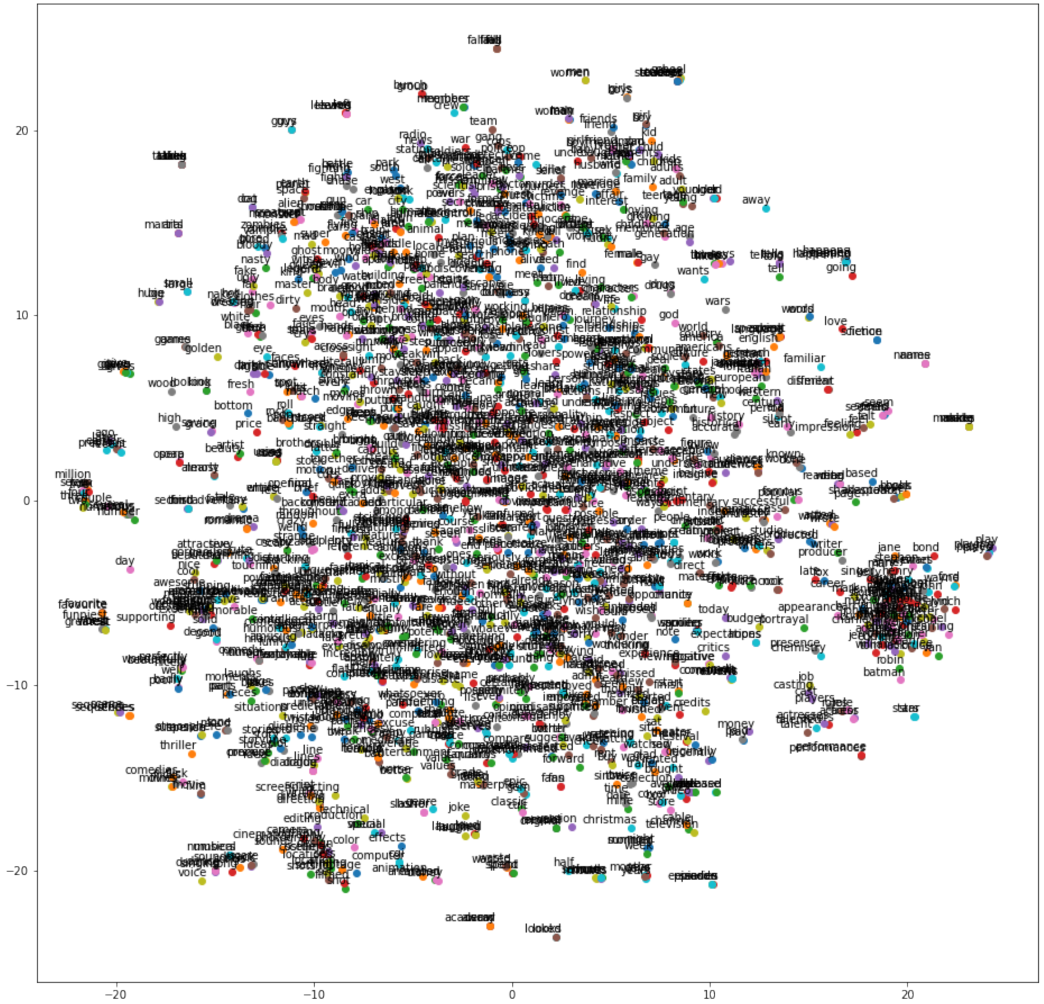
t-SNE
<u>en limitant</u>
<u>Word2Vec</u>
<u>aux mots ayant</u>
<u>au moins</u>
<u>100 occurrences</u>
(soit 6561 mots)


250 itérations


1000 itérations

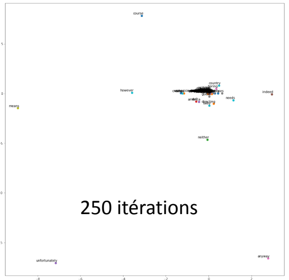
```
tsne_model = TSNE(perplexity=40, n_components=2, init='random', n_iter=1000, random_state=23, verbose=True, n_jobs=12)
```

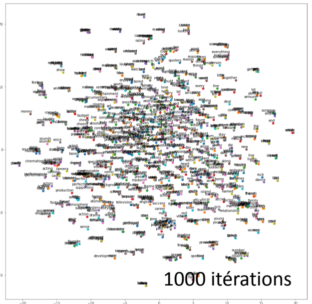
t-SNE des mots dont occurrences > 500 en limitant Word2Vec aux mots ayant au moins 100 occurrences

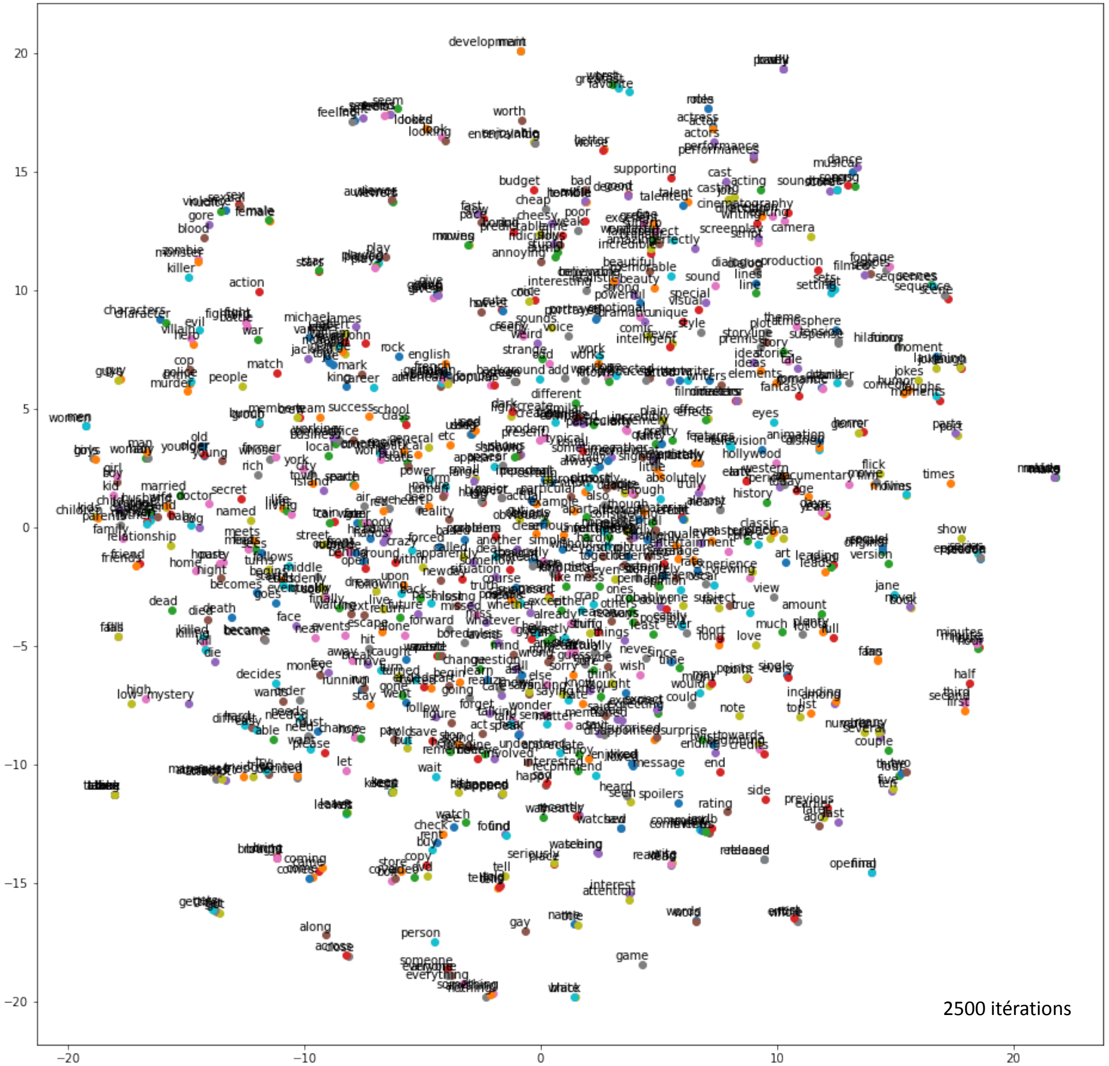


```
tsne_model = TSNE(perplexity=40, n_components=2, init='random', n_iter=1000, random_state=23, verbose=True, n_jobs=12)
```

t-SNE des mots dont occurrences > 1000 en limitant Word2Vec aux mots ayant au moins 100 occurrences

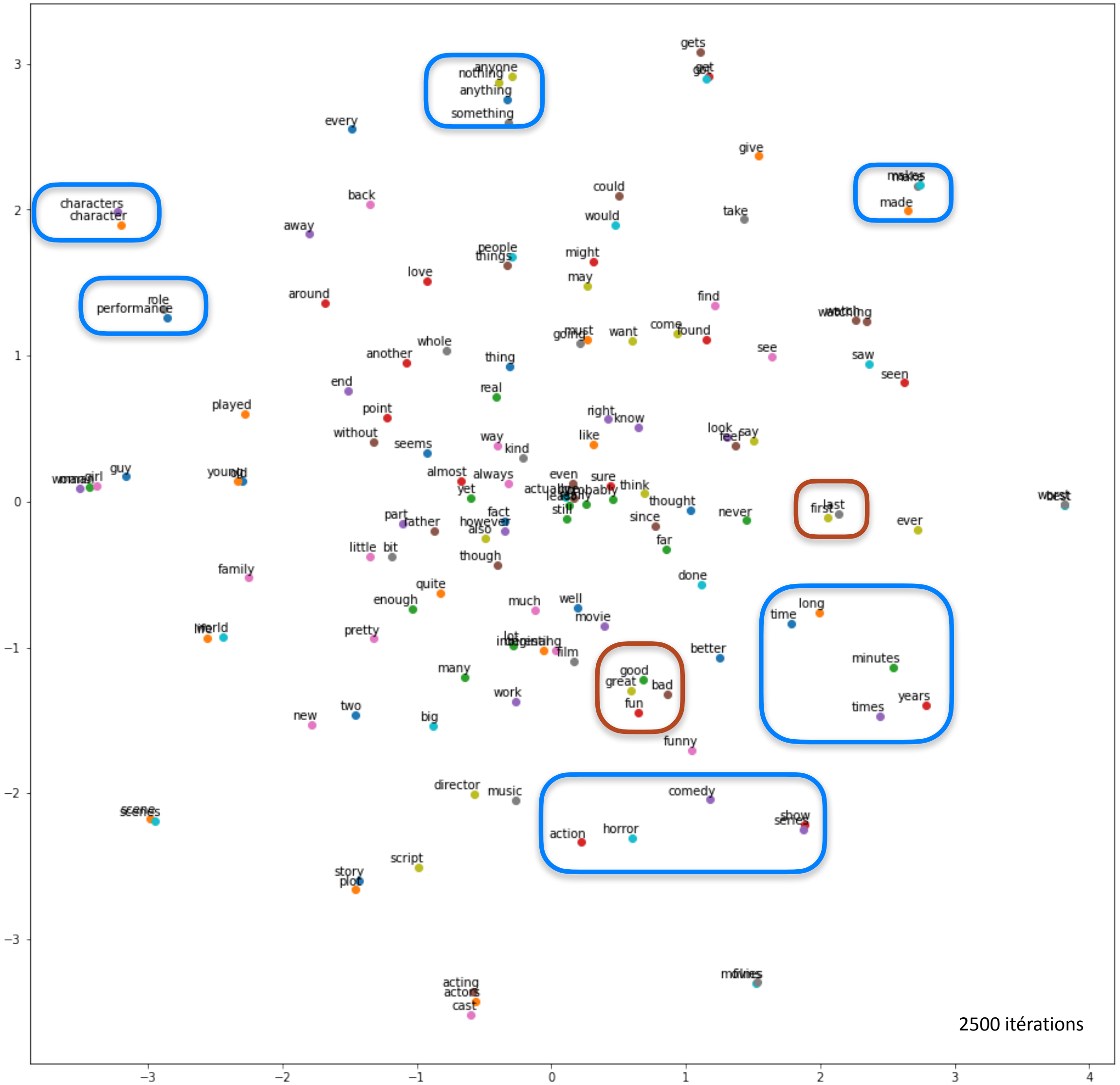






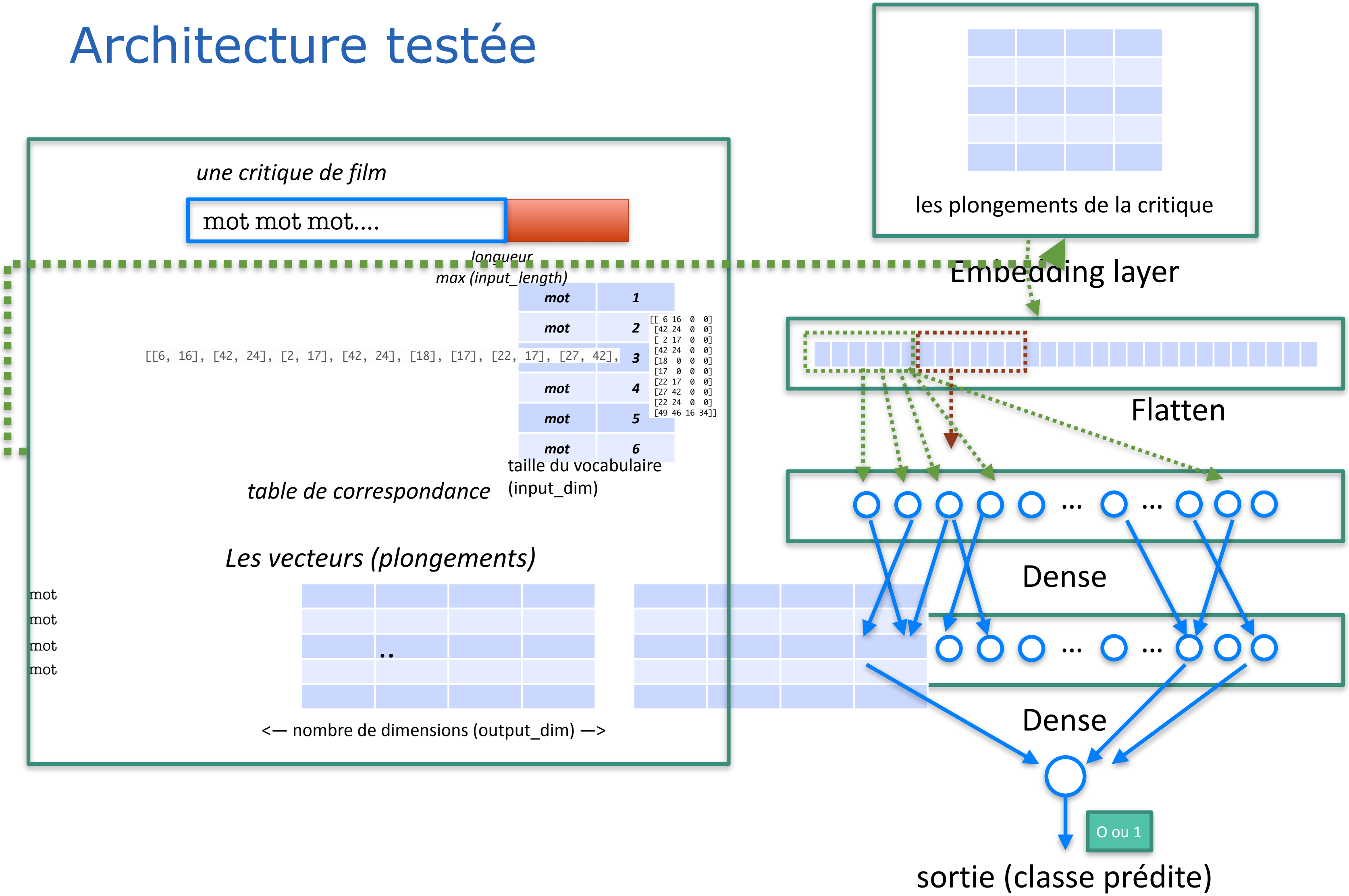

```
tsne_model = TSNE(perplexity=40, n_components=2, init='random', n_iter=1000, random_state=23, verbose=True, n_jobs=12)
```

t-SNE des mots dont occurrences > 5000 en limitant Word2Vec aux mots ayant au moins 100 occurrences




```
tsne_model = TSNE(perplexity=40, n_components=2, init='random', n_iter=2500, random_state=23, verbose=True, n_jobs=12)
```

Architecture testée

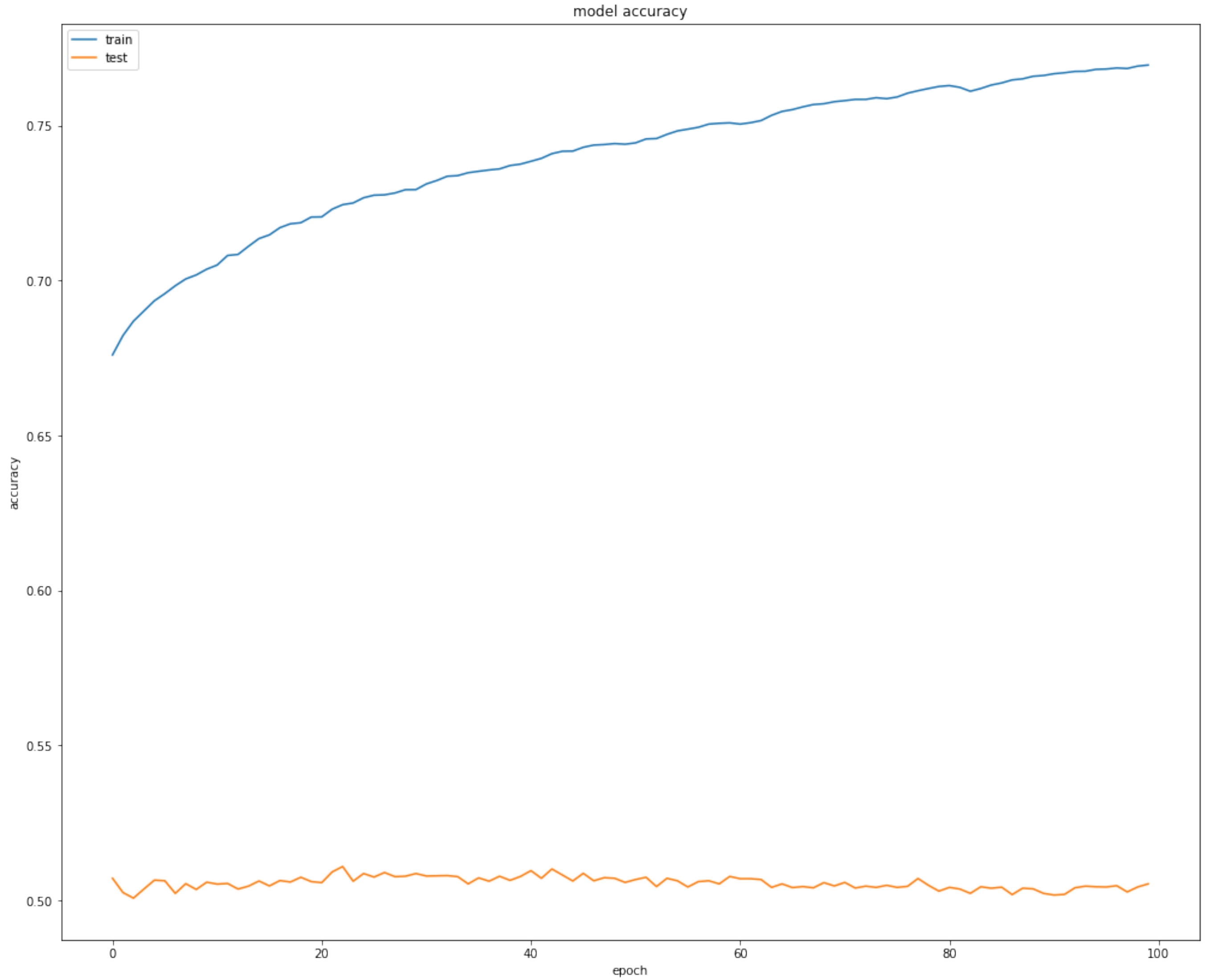


```
DIMENSION_EMBEDDINGS = 200
modelEmbeddings = gensim.models.Word2Vec(sentences=review_lines, size=DIMENSION_EMBEDDINGS, window=5, workers=12, min_count=100)
```

```
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

1er essai : on ne garde
que le mots qui apparaissent
au moins 100 fois
Réseau : une seule couche cachée

```
history = model.fit(X_train_pad, y_train, batch_size=128, epochs=100, validation_data=(X_test_pad, y_test), verbose=1)

Train on 35000 samples, validate on 15000 samples
Epoch 1/100
35000/35000 [==============================] - 11s 317us/step - loss: 0.4944 - accuracy: 0.6760 - val_loss: 1.0262 - val_accuracy: 0.5071
Epoch 2/100
35000/35000 [==============================] - 11s 312us/step - loss: 0.4831 - accuracy: 0.6823 - val_loss: 1.0537 - val_accuracy: 0.5025
Epoch 3/100
35000/35000 [==============================] - 11s 312us/step - loss: 0.4750 - accuracy: 0.6869 - val_loss: 1.1092 - val_accuracy: 0.5007
Epoch 4/100
35000/35000 [==============================] - 11s 315us/step - loss: 0.4708 - accuracy: 0.6902 - val_loss: 1.1248 - val_accuracy: 0.5037
Epoch 5/100
35000/35000 [==============================] - 11s 318us/step - loss: 0.4667 - accuracy: 0.6935 - val_loss: 1.2038 - val_accuracy: 0.5065
Epoch 6/100
35000/35000 [==============================] - 11s 313us/step - loss: 0.4619 - accuracy: 0.6958 - val_loss: 1.1686 - val_accuracy: 0.5063
Epoch 7/100
35000/35000 [==============================] - 11s 313us/step - loss: 0.4562 - accuracy: 0.6983 - val_loss: 1.2811 - val_accuracy: 0.5023
Epoch 8/100
35000/35000 [==============================] - 11s 313us/step - loss: 0.4543 - accuracy: 0.7005 - val_loss: 1.2396 - val_accuracy: 0.5054
Epoch 9/100
35000/35000 [==============================] - 12s 330us/step - loss: 0.4502 - accuracy: 0.7018 - val_loss: 1.3060 - val_accuracy: 0.5035
Epoch 10/100
35000/35000 [==============================] - 11s 315us/step - loss: 0.4459 - accuracy: 0.7037 - val_loss: 1.3753 - val_accuracy: 0.5059
Epoch 11/100
35000/35000 [==============================] - 11s 312us/step - loss: 0.4448 - accuracy: 0.7050 - val_loss: 1.3746 - val_accuracy: 0.5053
Epoch 12/100
35000/35000 [==============================] - 11s 312us/step - loss: 0.4424 - accuracy: 0.7081 - val_loss: 1.3833 - val_accuracy: 0.5055
Epoch 13/100
35000/35000 [==============================] - 11s 312us/step - loss: 0.4404 - accuracy: 0.7085 - val_loss: 1.4658 - val_accuracy: 0.5037
Epoch 14/100
35000/35000 [==============================] - 11s 309us/step - loss: 0.4375 - accuracy: 0.7111 - val_loss: 1.3758 - val_accuracy: 0.5046
Epoch 15/100
35000/35000 [==============================] - 11s 311us/step - loss: 0.4329 - accuracy: 0.7136 - val_loss: 1.5291 - val_accuracy: 0.5063
Epoch 16/100
35000/35000 [==============================] - 11s 310us/step - loss: 0.4289 - accuracy: 0.7148 - val_loss: 1.4570 - val_accuracy: 0.5047
```

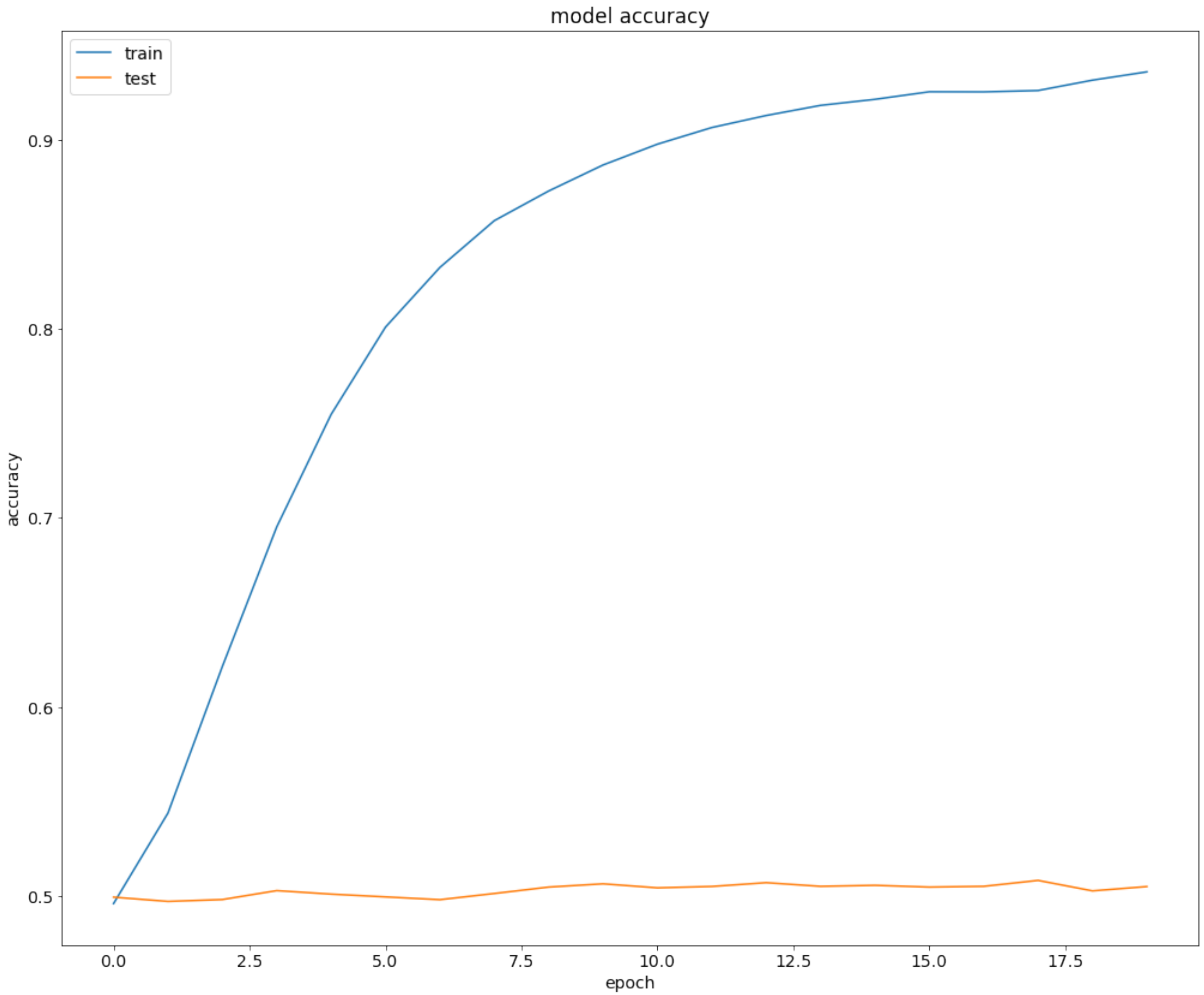


model accuracy
train
test
accuracy
epoch
0.50
0.55
0.60
0.65
0.70
0.75
0
20
40
60
80
100

```
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

2è essai :
on ajoute une 2è couche

```
history = model.fit(X_train_pad, y_train, batch_size=128, epochs=20, validation_data=(X_test_pad, y_test), verbose=1)

Train on 35000 samples, validate on 15000 samples
Epoch 1/20
35000/35000 [==============================] - 12s 343us/step - loss: 0.6959 - accuracy: 0.4959 - val_loss: 0.6937 - val_accuracy: 0.4993
Epoch 2/20
35000/35000 [==============================] - 11s 303us/step - loss: 0.6754 - accuracy: 0.5438 - val_loss: 0.7063 - val_accuracy: 0.4970
Epoch 3/20
35000/35000 [==============================] - 11s 302us/step - loss: 0.6101 - accuracy: 0.6214 - val_loss: 0.7265 - val_accuracy: 0.4980
Epoch 4/20
35000/35000 [==============================] - 11s 303us/step - loss: 0.5159 - accuracy: 0.6953 - val_loss: 0.7975 - val_accuracy: 0.5027
Epoch 5/20
35000/35000 [==============================] - 11s 302us/step - loss: 0.4270 - accuracy: 0.7548 - val_loss: 0.9331 - val_accuracy: 0.5009
Epoch 6/20
35000/35000 [==============================] - 11s 303us/step - loss: 0.3545 - accuracy: 0.8011 - val_loss: 1.0973 - val_accuracy: 0.4994
Epoch 7/20
35000/35000 [==============================] - 11s 303us/step - loss: 0.2949 - accuracy: 0.8327 - val_loss: 1.2164 - val_accuracy: 0.4979
Epoch 8/20
35000/35000 [==============================] - 11s 305us/step - loss: 0.2526 - accuracy: 0.8574 - val_loss: 1.4961 - val_accuracy: 0.5012
Epoch 9/20
35000/35000 [==============================] - 11s 304us/step - loss: 0.2285 - accuracy: 0.8731 - val_loss: 1.6137 - val_accuracy: 0.5046
Epoch 10/20
35000/35000 [==============================] - 11s 303us/step - loss: 0.2023 - accuracy: 0.8869 - val_loss: 1.7251 - val_accuracy: 0.5063
Epoch 11/20
35000/35000 [==============================] - 11s 305us/step - loss: 0.1832 - accuracy: 0.8979 - val_loss: 1.9321 - val_accuracy: 0.5042
Epoch 12/20
35000/35000 [==============================] - 11s 300us/step - loss: 0.1657 - accuracy: 0.9067 - val_loss: 1.9291 - val_accuracy: 0.5049
Epoch 13/20
35000/35000 [==============================] - 11s 302us/step - loss: 0.1539 - accuracy: 0.9131 - val_loss: 2.0730 - val_accuracy: 0.5069
```

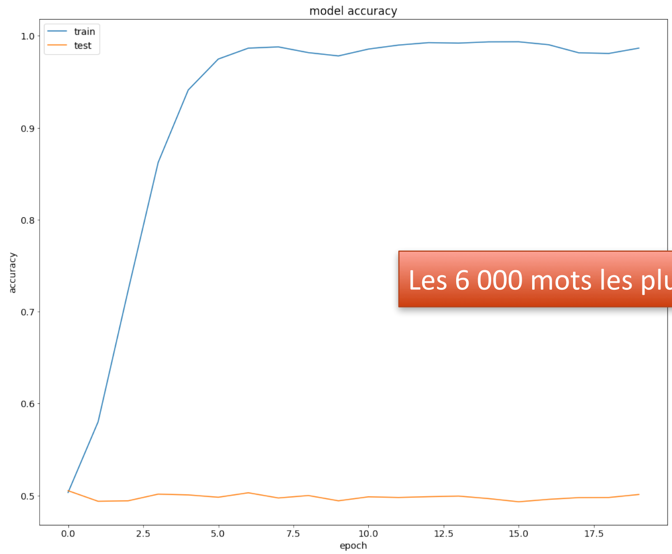

model accuracy
train
test
accuracy
epoch
0.9
0.8
0.7
0.6
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

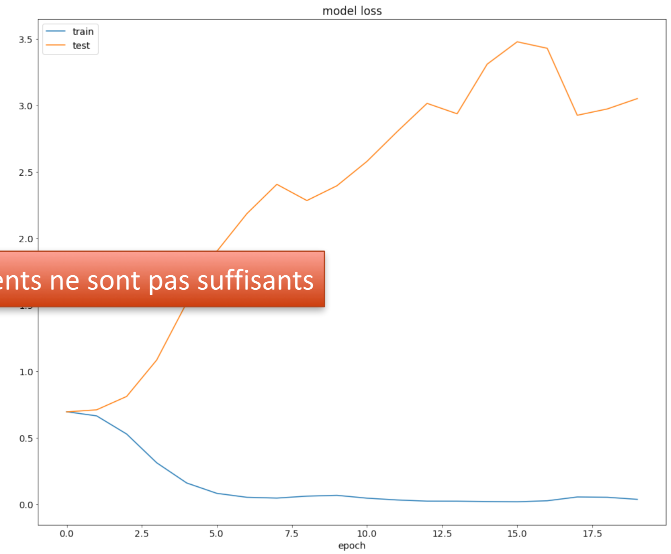
3è essai : on rajoute une 3è couche…

```
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_12 (Embedding)     (None, 256, 200)          19352000
_________________________________________________________________
flatten_7 (Flatten)          (None, 51200)             0
_________________________________________________________________
dense_18 (Dense)             (None, 32)                1638432
_________________________________________________________________
dense_19 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_20 (Dense)             (None, 8)                 264
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 9
=================================================================
Total params: 20,991,761
Trainable params: 1,639,761
Non-trainable params: 19,352,000
```

```
Epoch 16/20
35000/35000 [==============================] - 12s 330us/step - loss: 0.0199 - accuracy: 0.9935 - val_loss: 3.4786 - val_accuracy: 0.4931
```



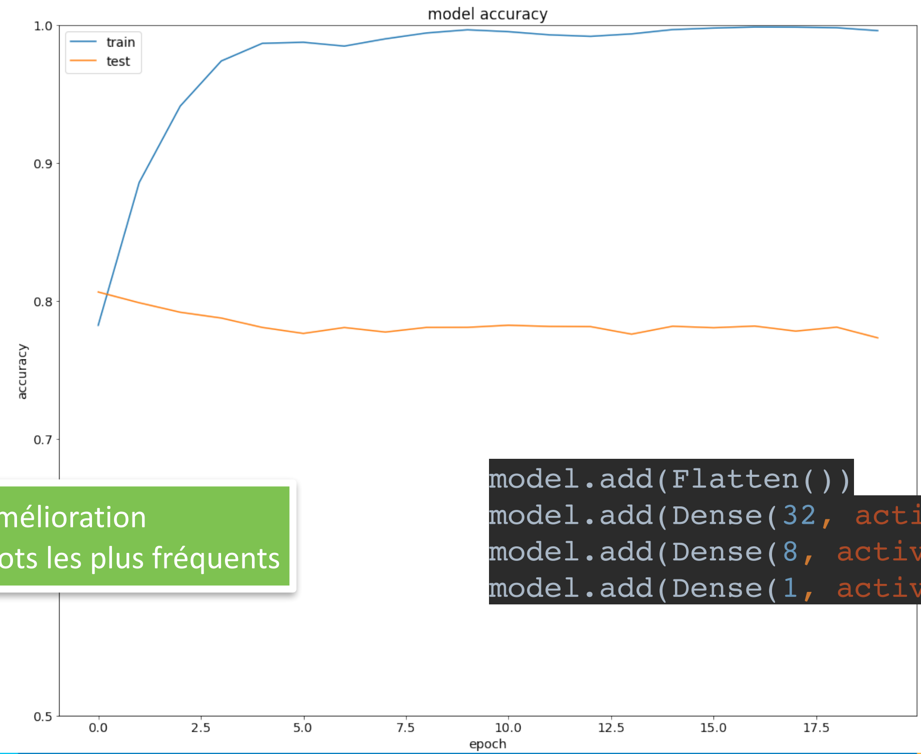


Les 6 000 mots les plus fréquents ne sont pas suffisants

4è essai : on prend en compte plus de mots du vocabulaire

```
DIMENSION_EMBEDDINGS = 200
model = gensim.models.Word2Vec(sentences=review_lines, size=DIMENSION_EMBEDDINGS, window=5, workers=12, min_count=10)
```

```
history = model.fit(X_train_pad, y_train, batch_size=128, epochs=20, validation_data=(X_test_pad, y_test), verbose=1)
Epoch 1/20
35000/35000 [==============================] - 12s 340us/step - loss: 0.4560 - accuracy: 0.7825 - val_loss: 0.4157 - val_accuracy: 0.8066
Epoch 2/20
35000/35000 [==============================] - 11s 304us/step - loss: 0.2631 - accuracy: 0.8859 - val_loss: 0.4545 - val_accuracy: 0.7988
```

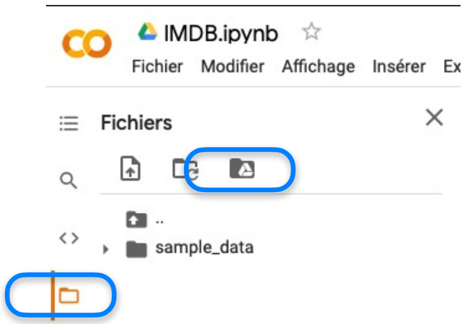


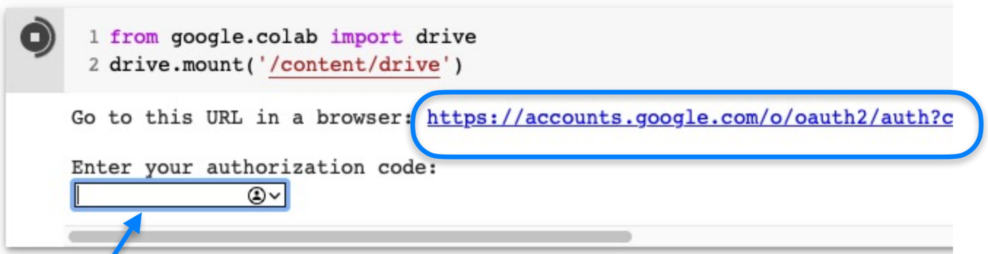
Nette amélioration
avec les 27 000 mots les plus fréquents

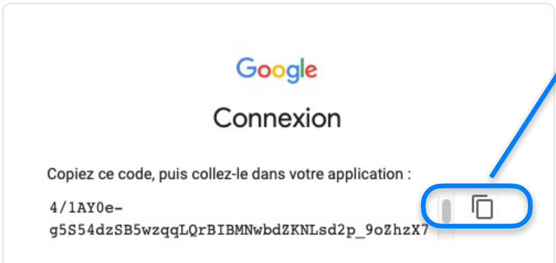
```
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

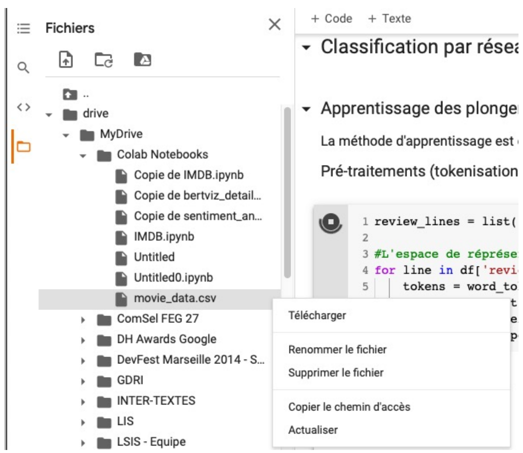
DANS GOOGLE COLAB

https://colab.research.google.com









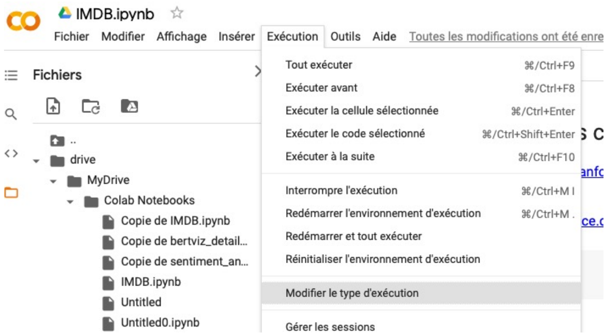
!! Penser à changer le nom du répertoire/dossier de départ

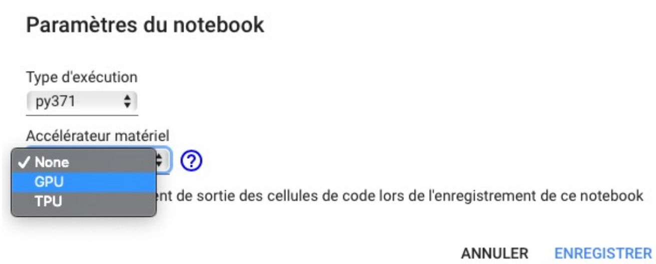
```
[2]  repertoire_depart = '/content/drive/MyDrive/Colab Notebooks/'
     nomCSV = repertoire_depart+'/movie_data.csv'
```

Appel de la méthode entraînant le réseau et test au fur et à mesure des époques :

```
#selon la configuration de votre machine, des conflits entre bibliothèques peuvent survenir.
#Si Python quitte brutalement la ligne suivante peut permettre de contourner le problème
#sinon la mettre en commentaires
os.environ['KMP_DUPLICATE_LIB_OK']='True'

history = model.fit(X_train_pad, y_train, batch_size=32, epochs=3, validation_data=(X_test_pad, y_test), verbose=1)
```

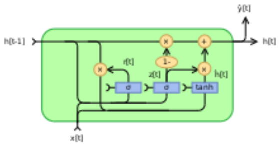
```
Epoch 1/3
1094/1094 [==============================] - 10s 9ms/step - loss: 0.5306 - accuracy: 0.7269 - val_loss: 0.4157 - val_accuracy: 0.8067
Epoch 2/3
1094/1094 [==============================] - 8s 8ms/step - loss: 0.3056 - accuracy: 0.8690 - val_loss: 0.4371 - val_accuracy: 0.8022
Epoch 3/3
1094/1094 [==============================] - 8s 8ms/step - loss: 0.2212 - accuracy: 0.9106 - val_loss: 0.4864 - val_accuracy: 0.7949
```






```
#selon la configuration de votre machine, des conflits entre bibliothèq
#Si Python quitte brutalement la ligne suivante peut permettre de conto
#sinon la mettre en commentaires
os.environ['KMP_DUPLICATE_LIB_OK']='True'

history = model.fit(X_train_pad, y_train, batch_size=32, epochs=3, vali
```

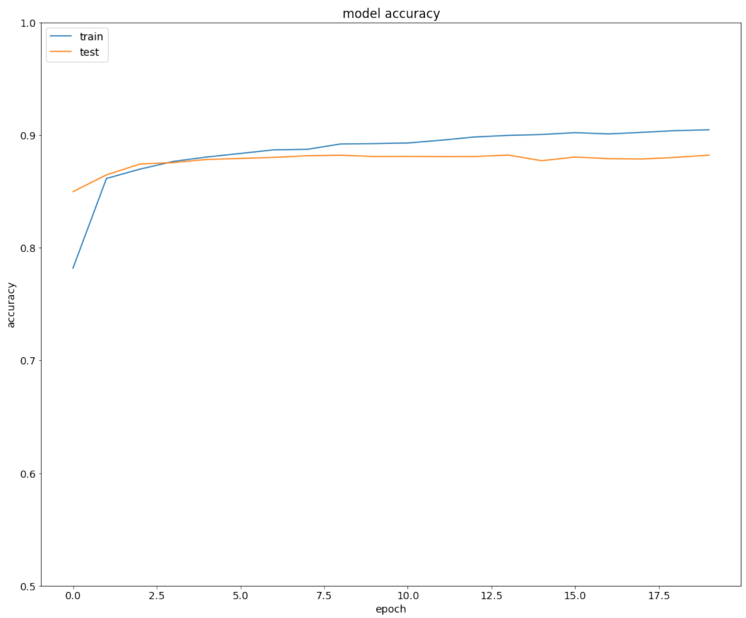
```
Epoch 1/3
1094/1094 [==============================] - 8s 6ms/step - loss: 0.4966 -
Epoch 2/3
1094/1094 [==============================] - 6s 5ms/step - loss: 0.2925 -
Epoch 3/3
1094/1094 [==============================] - 6s 5ms/step - loss: 0.2143 -
```

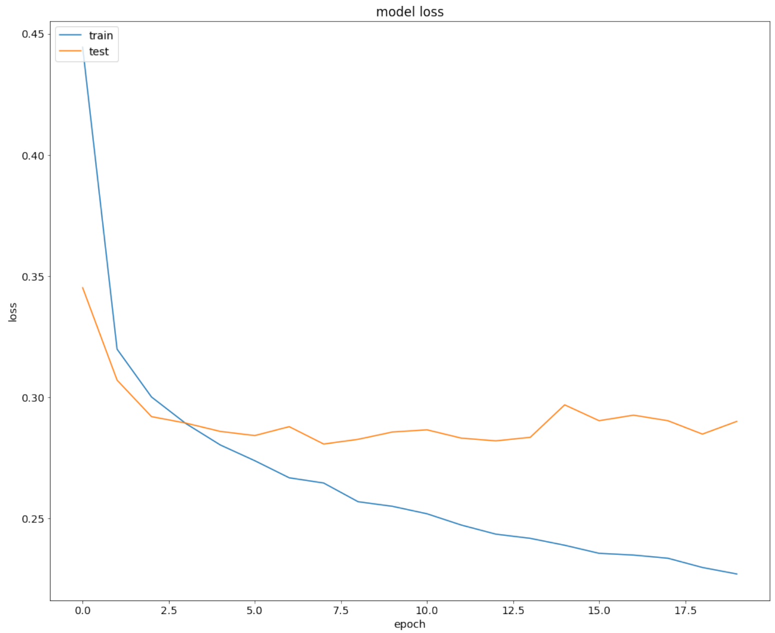


```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 128, 200)          19345800
_________________________________________________________________
gru (GRU)                    (None, 32)                22464
_________________________________________________________________
dense_7 (Dense)              (None, 8)                 264
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 9
=================================================================
Total params: 19,368,537
Trainable params: 22,737
Non-trainable params: 19,345,800
```

```
Epoch 1/20
1094/1094 [==============================] - 187s 168ms/step - loss: 0.5532 - accuracy: 0.6923 - val_loss: 0.3451 - val_accuracy: 0.8497
Epoch 2/20
1094/1094 [==============================] - 188s 172ms/step - loss: 0.3212 - accuracy: 0.8617 - val_loss: 0.3069 - val_accuracy: 0.8646
Epoch 3/20
1094/1094 [==============================] - 188s 171ms/step - loss: 0.2948 - accuracy: 0.8736 - val_loss: 0.2919 - val_accuracy: 0.8743
Epoch 4/20
1094/1094 [==============================] - 191s 175ms/step - loss: 0.2875 - accuracy: 0.8748 - val_loss: 0.2892 - val_accuracy: 0.8755
Epoch 5/20
1094/1094 [==============================] - 185s 169ms/step - loss: 0.2820 - accuracy: 0.8804 - val_loss: 0.2858 - val_accuracy: 0.8783
Epoch 6/20
1094/1094 [==============================] - 182s 166ms/step - loss: 0.2721 - accuracy: 0.8833 - val_loss: 0.2841 - val_accuracy: 0.8792
Epoch 7/20
1094/1094 [==============================] - 185s 169ms/step - loss: 0.2629 - accuracy: 0.8881 - val_loss: 0.2877 - val_accuracy: 0.8801
Epoch 8/20
1094/1094 [==============================] - 178s 162ms/step - loss: 0.2607 - accuracy: 0.8869 - val_loss: 0.2806 - val_accuracy: 0.8816
Epoch 9/20
1094/1094 [==============================] - 182s 166ms/step - loss: 0.2553 - accuracy: 0.8929 - val_loss: 0.2825 - val_accuracy: 0.8821
```

```
1094/1094 [==============================] - 24s 22ms/step - loss: 0.1825 - accuracy: 0.9275
469/469 [==============================] - 10s 21ms/step - loss: 0.2899 - accuracy: 0.8821
```


model accuracy
train
test
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


model loss
train
test
loss
epoch
0.45
0.40
0.35
0.30
0.25
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

En conclusion

— Approche bayésienne « naïve » : 0,85
- durée d'apprentissage : quelques secondes

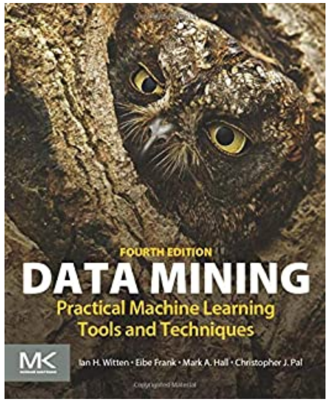
— Approche neuronale « plongements + couches denses » :
- mal configurée : 0,50 (soit l'équivalent d'un tirage aléatoire…)
- **après quelques réglages et essais : 0,80**
- durée d'apprentissage
  — avec CPU seul 12 cœurs : environ 10s / epoch, soit 3 mn
  — avec GPU (Google Colab) : environ 8s. / epoch

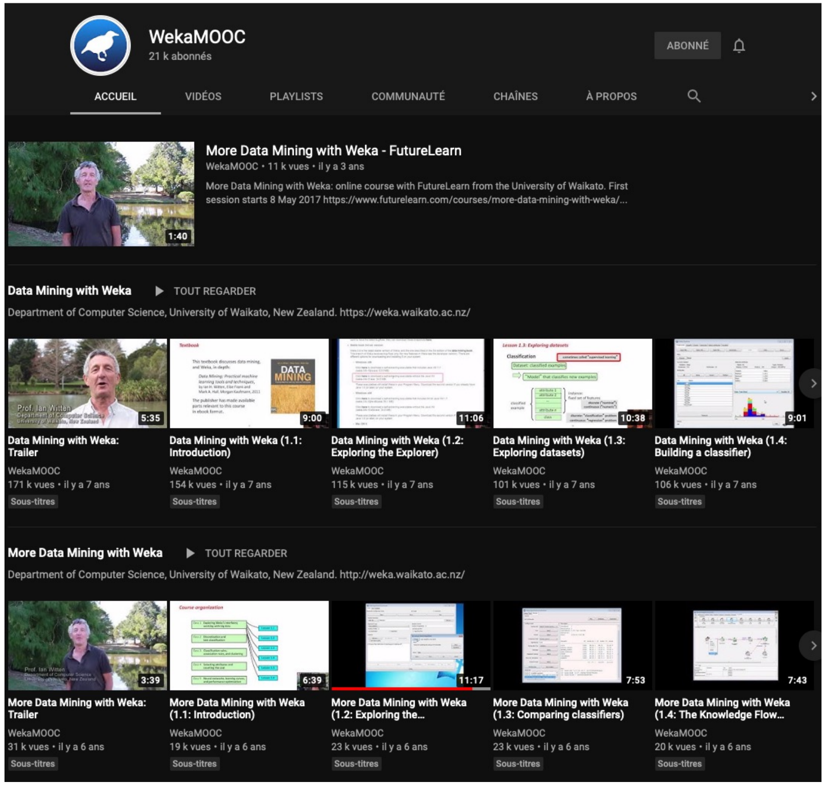
— Approche neuronale « plongements + réseaux récurrents »
- **meilleur score : 0,88 (soit 3% de gain) — 0,9**
- durée d'apprentissage :
  — avec CPU seul : environ 3000 s. / epoch, soit > 24 h.
  — avec TPU (Google Colab) : environ 200 s. / epoch

POUR ALLER PLUS LOIN

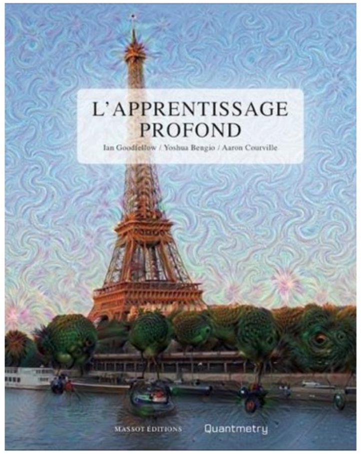
Références et liens pour WEKA

Data Mining with Weka





Data Mining: Practical Machine Learning Tools and Techniques
de Ian H. Witten , Eibe Frank, et al.



L'apprentissage profond

Yoshua Bengio, Ian Goodfellow, Aaron Courville

Massot Editions - 18 Octobre 2018

Sciences & Techniques

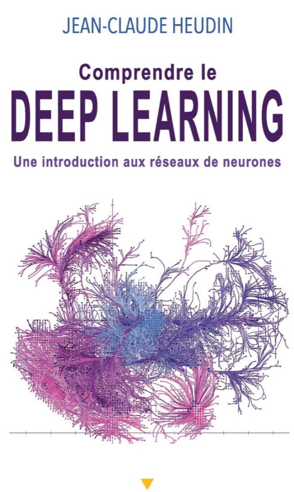
Voir les détails produits

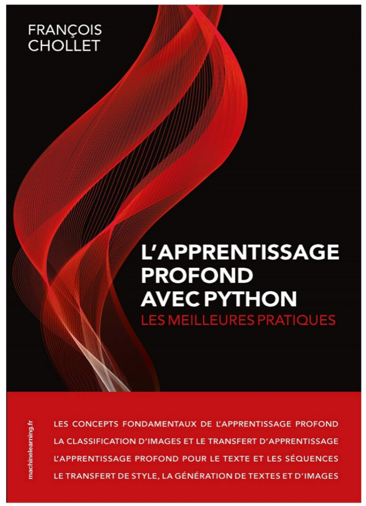
(Aucun avis)

À propos

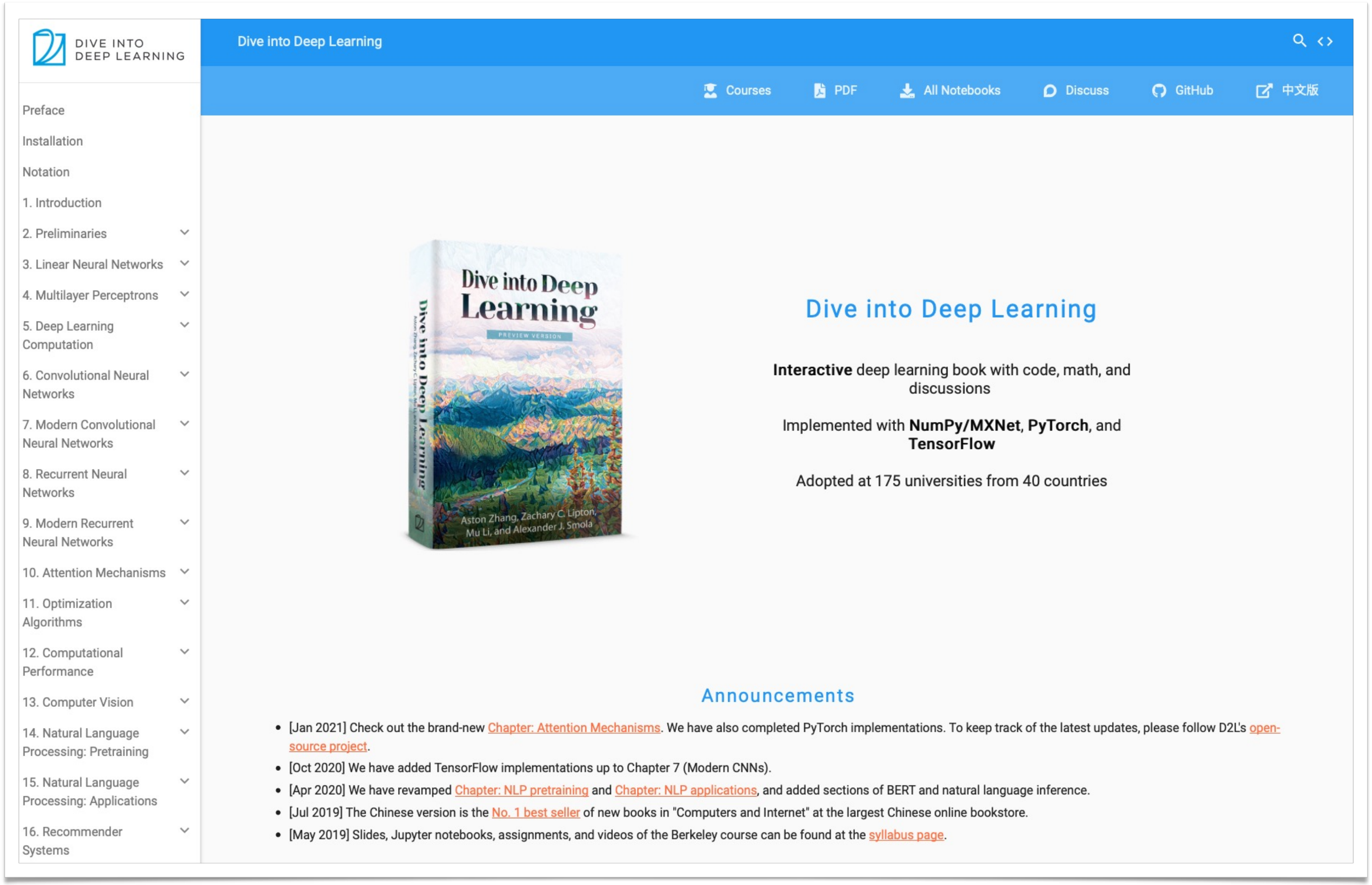
Écrit par trois experts dans le domaine, Deep Learning est le seul livre complet sur le sujet. Il fournit une perspective générale et des préliminaires mathématiques indispensables aux ingénieurs en logiciel et aux étudiants qui entrent sur le terrain, et sert de référence aux autorités. Elon Musk, cofondateur et PDG de Tesla et SpaceXstudents L'apprentissage profond (ou deep learning) est un apprentissage automatique qui permet à l'ordinateur d'apprendre par l'expérience et de comprendre le monde en termes de hiérarchie de concepts. Parce que l'ordinateur recueille des connaissances à partir de l'expérience, il n'est pas nécessaire qu'un opérateur humain spécifie formellement toutes les connaissances dont l'ordinateur a besoin. Cet ouvrage présente un large éventail de sujets d'apprentissage profond.

Le **Lire la suite**

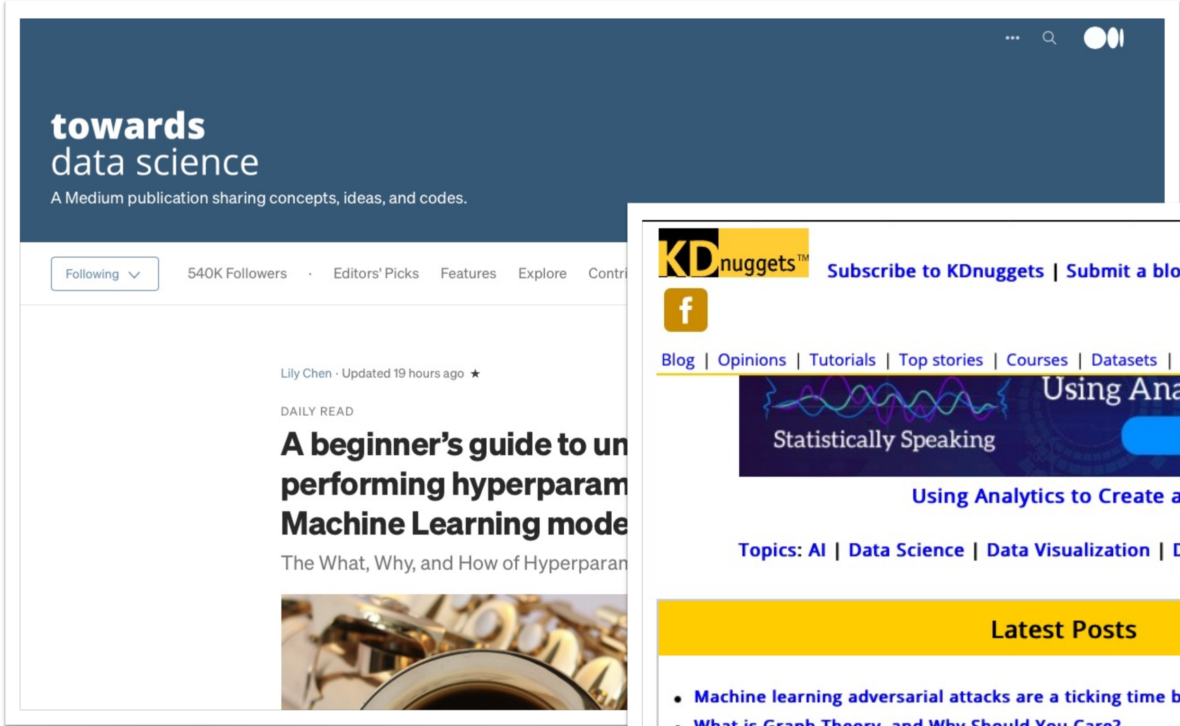


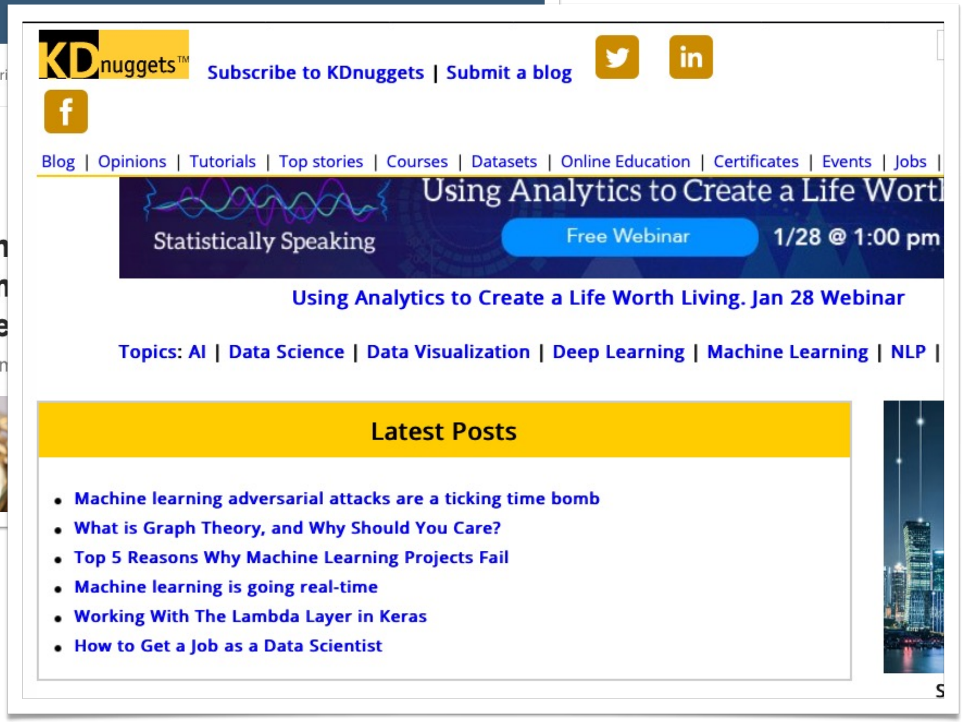


https://d2l.ai/index.html



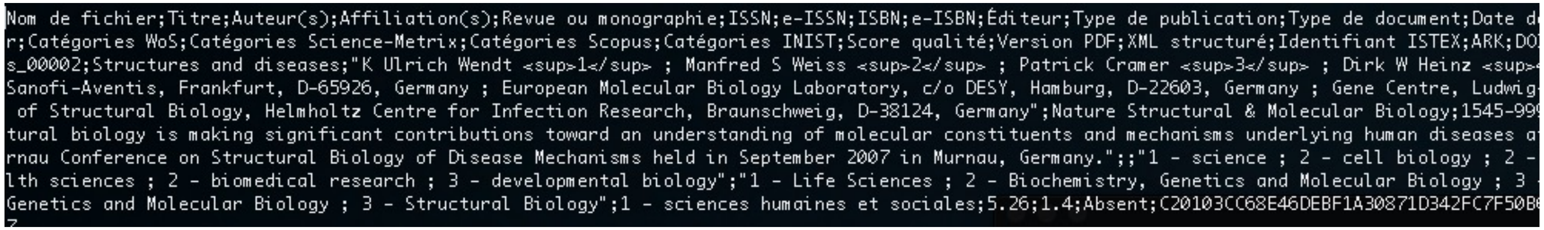
https://towardsdatascience.com

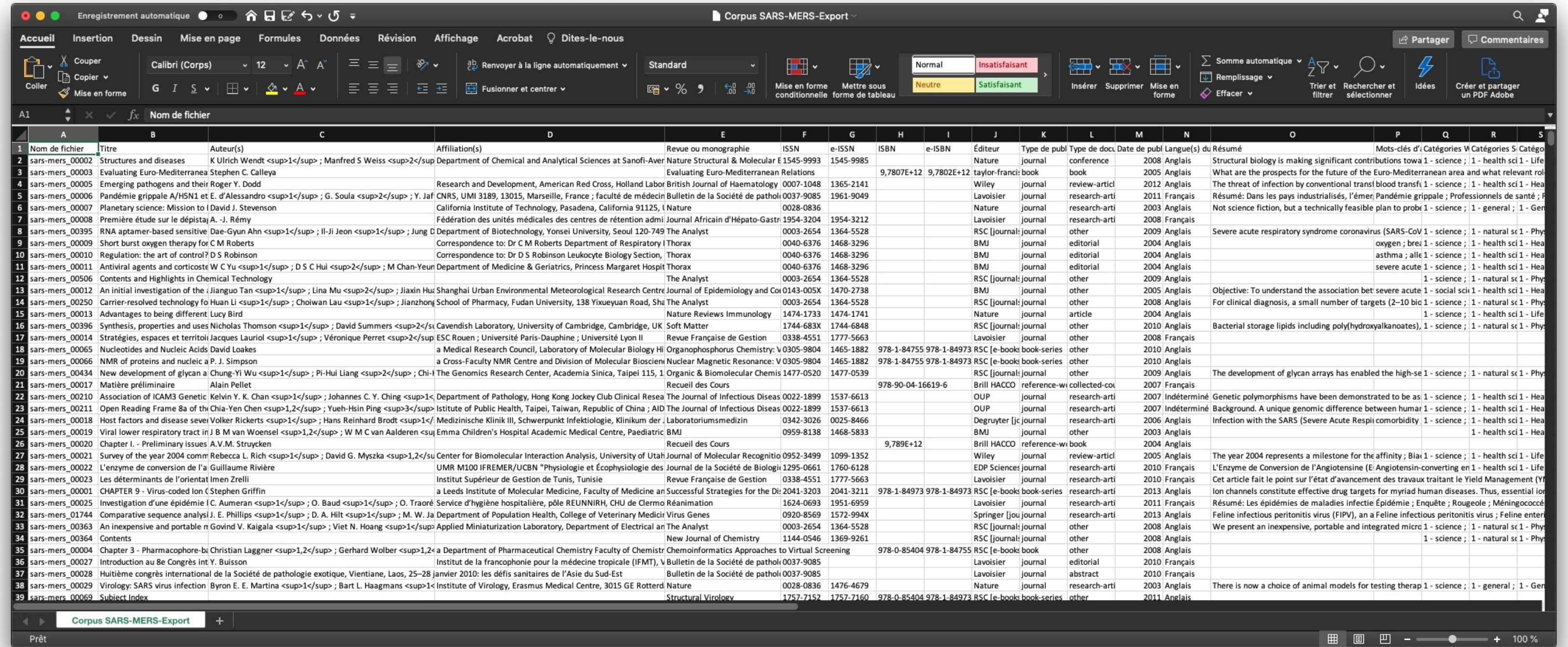




https://www.kdnuggets.com

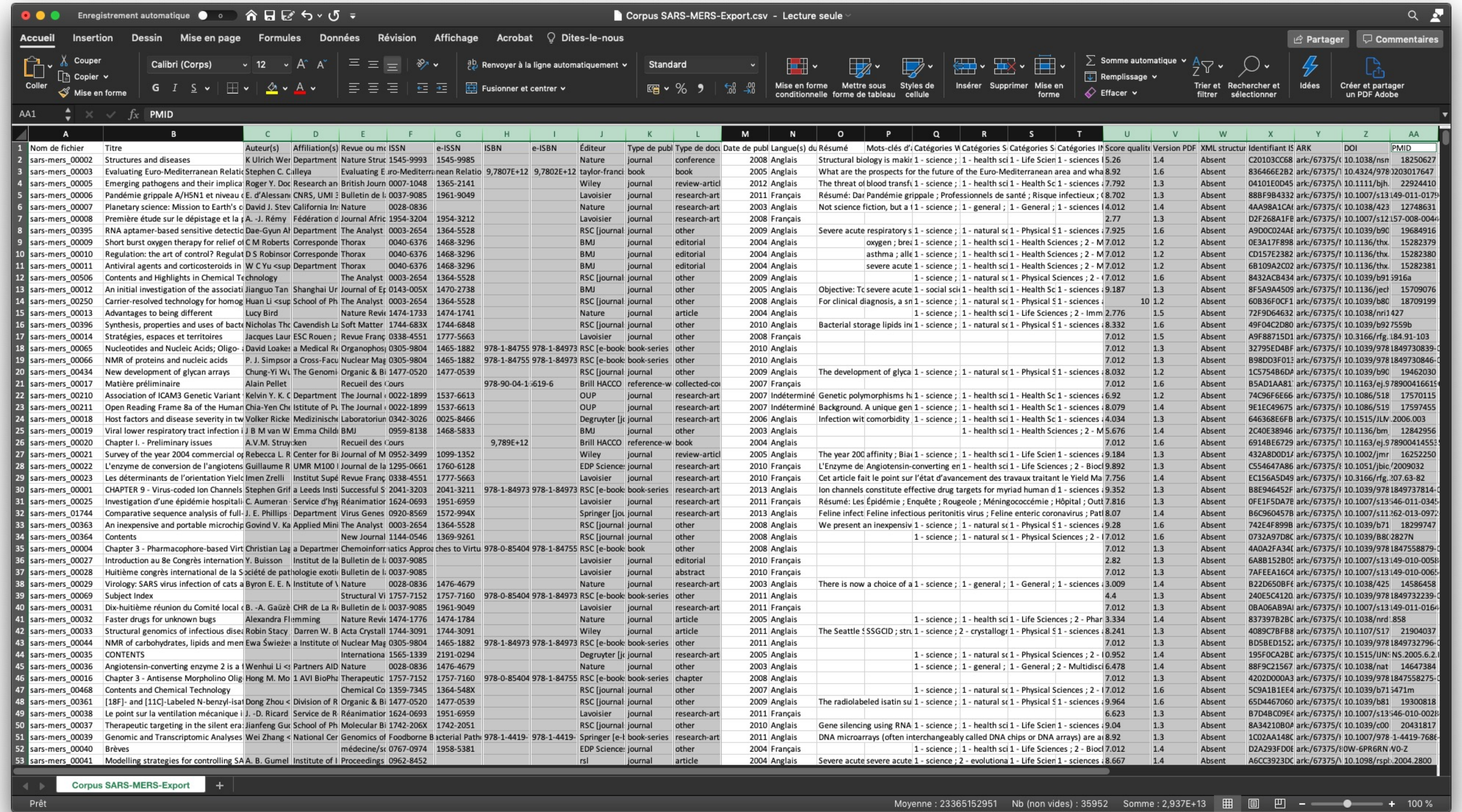
CLASSIFICATION THEMATIQUE ET PARTITIONNEMENT





fichier .CSV sur 27 colonnes
(méta-données ISTEX)

Identifier les colonnes (in)utiles



Lire le fichier .csv en Python avec le module Pandas

```
import pandas as pd

fichierCSV = "//Users/Patrice/PycharmProjects/ANF2021/ANF/test2.csv"
# load train data
data = pd.read_csv(fichierCSV, sep=";", header=0, error_bad_lines=False, encoding="utf_8", usecols=[0,1,13,14,15,16])
data.info()
data.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 6 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Nom de fichier        9 non-null      object
 1   Titre                 9 non-null      object
 2   Langue(s) du document  9 non-null     object
 3   Résumé                6 non-null      object
 4   Mots-clés d'auteur    4 non-null      object
 5   Catégories WoS        6 non-null      object
dtypes: object(6)
memory usage: 560.0+ bytes
```

	Nom de fichier	Titre	Langue(s) du document	Résumé	Mots-clés d'auteur	Catégories WoS
0	sars-mers_00002	Structures and diseases	Anglais	Structural biology is making significant contr...	NaN	1 - science ; 2 - cell biology ; 2 - biophysic...
1	sars-mers_00003	Evaluating Euro-Mediterranean Relations	Anglais	What are the prospects for the future of the E...	NaN	NaN
2	sars-mers_00005	Emerging pathogens and their implications for ...	Anglais	The threat of infection by conventional transf...	blood transfusion ; safety ; emerging infections	1 - science ; 2 - hematology
3	sars-mers_00006	Pandémie grippale A/H5N1 et niveau de préparat...	Français	Résumé: Dans les pays industrialisés, l'émerge...	Pandémie grippale ; Professionnels de santé ; ...	NaN
4	sars-mers_00007	Planetary science: Mission to Earth's core — a...	Anglais	Not science fiction, but a technically feasibl...	NaN	1 - science ; 2 - multidisciplinary sciences
5	sars-mers_00008	Première étude sur le dépistage et la prise en...	Français	NaN	NaN	NaN
6	sars-mers_00395	RNA aptamer-based sensitive detection of SARS ...	Anglais	Severe acute respiratory syndrome coronavirus ...	NaN	1 - science ; 2 - chemistry, analytical
7	sars-mers_00009	Short burst oxygen therapy for relief of breat...	Anglais	NaN	oxygen ; breathlessness ; chronic obstructive ...	1 - science ; 2 - respiratory system
8	sars-mers_00010	Regulation: the art of control? Regulatory T c...	Anglais	NaN	asthma ; allergy ; immunotherapy ; T cells	1 - science ; 2 - respiratory system

Explorer les données

Données Corpus SARS-MERS-Export.csv

Mise en forme en .csv pour Weka

Lecture du fichier de départ Corpus SARS-MERS-Export.csv

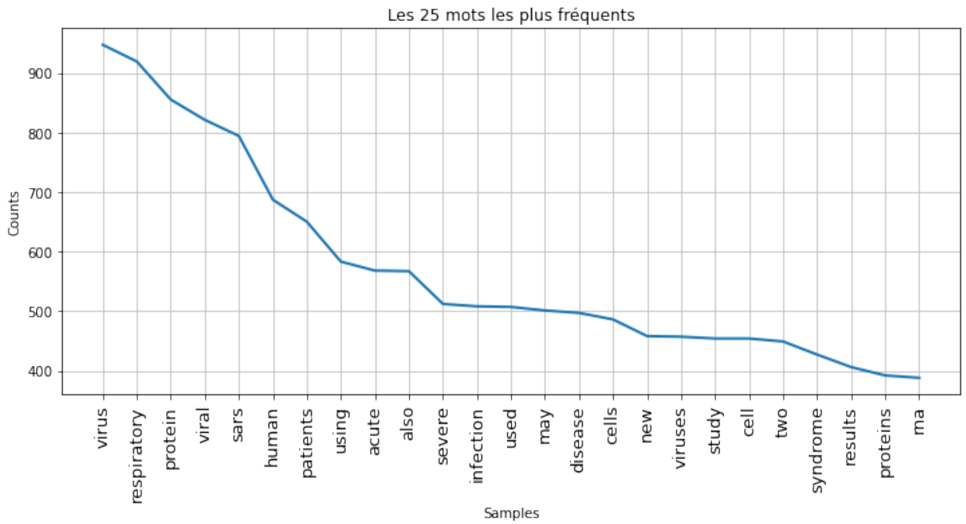
Entrée [ ]:

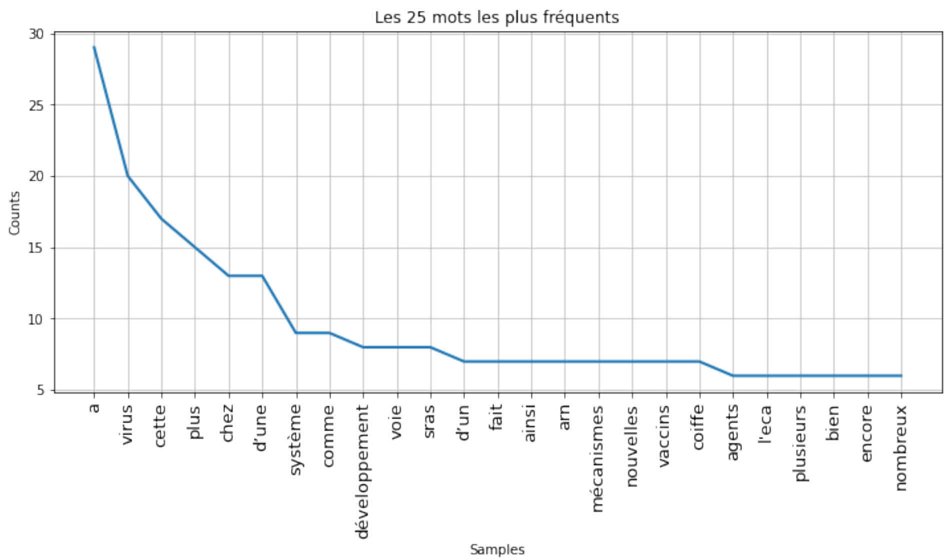
```python
import pandas as pd
import csv
from collections import Counter
import matplotlib.pyplot as plt
import nltk
from nltk.corpus import stopwords

fichierCSVEntree = "/Users/Patrice/PycharmProjects/ANF2021/ANF/CorpusCovid.csv"
fichierSortie = "/Users/Patrice/PycharmProjects/ANF2021/ANF/CorpusWeka.csv"
```

```
Nombre total de documents :   2532 (2532, 6)
Nombre de documents en français :  67
Documents en  Anglais  :  2197
Documents en  Français  :  67
Documents en  Indéterminé  :  230
Documents en  Allemand  :  38
Les mots les plus fréquents par langue dans les titres :
pour langue  Anglais  :  [('of', 1543), ('and', 973), ('in', 852), ('the', 774), ('a', 451), ('for', 338), ('respiratory', 281), ('acute', 229), ('to', 217), ('with', 217), ('sars', 204), ('severe', 204), ('coronavirus', 202), ('virus', 198), ('syndrome', 179), ('on', 164), ('by', 162), ('from', 150), ('human', 138), ('protein', 131), ('an', 114), ('viral', 101), ('infection', 84), ('analysis', 82), ('infectious', 82)]
pour langue  Français  :  [('de', 41), ('des', 29), ('la', 29), ('et', 18), ('les', 14), ('le', 14), ('en', 11), (':', 10), ('à', 10), ('du', 9), ('virus', 9), ('un', 8), ('au', 6), ('sur', 5), ('brèves', 5), ('dans', 4), ('santé', 4), ('une', 3), ('prise', 3), ('charge', 3), ('international', 3), ('développement', 3), ('entre', 3), ('?', 3), ('nouvelles', 3)]
pour langue  Indéterminé  :  [('of', 170), ('and', 112), ('in', 112), ('the', 90), ('respiratory', 85), ('acute', 74), ('severe', 65), ('a', 57), ('with', 52), ('syndrome', 49), ('coronavirus', 49), ('for', 39), ('human', 37), ('by', 30), ('infection', 25), ('to', 23), ('viral', 20), ('virus', 17), ('patients', 17), ('influenza', 16), ('from', 14), ('clinical', 13), ('on', 12), ('disease', 12), ('is', 12)]
```

```python
print('Nombre total de documents :  ',len(data), data.shape)  data: {DataFrame:
#Nombre de document par langue
print("Nombre de documents en français : ", len(data[data.Langue=='Français']))
for langue in data['Langue'].unique():  data: {DataFrame: (2532, 6)}
    print("Documents en ", langue, " : ", len(data[data.Langue==langue]))  langu
data.groupby('Langue', dropna=False).describe()  data: {DataFrame: (2532, 6)}
```

```python
for langue in data['Langue'].unique():  data: {DataFrame: (2532, 6)}
    mots_des_titres = []
    for titre in list(data['Titre'][data.Langue==langue]):  data: {DataFrame: (2532
        mots = titre.split()  titre: Trends in der Impfstoffentwicklung. DNA- und z
        for mot in mots:  mots: ['Für', 'eine', 'Reihe', 'von', 'Infektionskrankhei
            mots_des_titres.append(mot.lower())  mots_des_titres: ['wirksamkeit', '
    print("pour langue ", langue, " : ", Counter(mots_des_titres).most_common(25))
```


Les 25 mots les plus fréquents
Counts
Samples
virus
respiratory
protein
viral
sars
human
patients
using
acute
also
severe
infection
used
may
disease
cells
new
viruses
study
cell
two
syndrome
results
proteins
ma


Les 25 mots les plus fréquents
Counts
Samples
a
virus
cette
plus
chez
d'une
système
comme
développement
voie
sras
d'un
fait
ainsi
arn
mécanismes
nouvelles
vaccins
coiffe
agents
l'eca
plusieurs
bien
encore
nombreux

Mise en forme du .csv pour Weka

— Weka utilise normalement le format .ARFF mais peut importer les .CSV

— Il est plus facile de respecter les paramètres CSV de Weka avant l'importation…

```
resumesClasses = data[[‘Resume','Categories']][data['Resume'].notnull()]
resumesClasses = resumesClasses[resumesClasses['Categories'].notnull()]
```

On extrait les colonnes *Résumés* et *Catégories*

```
for index, row in resumesClasses.iterrows():
    cat = str(row['Categories'])
    catRetenue = re.search(" 2 - ([a-z]+)", cat)
    if catRetenue:
        row['Categories'] = catRetenue[1]
    else:
        catRetenue = re.search("1 - ([a-z]+)", cat)
        if catRetenue:
            row['Categories'] = catRetenue[1]
```

On ne conserve qu'une seule catégorie

1 - science ; 2 - immunology

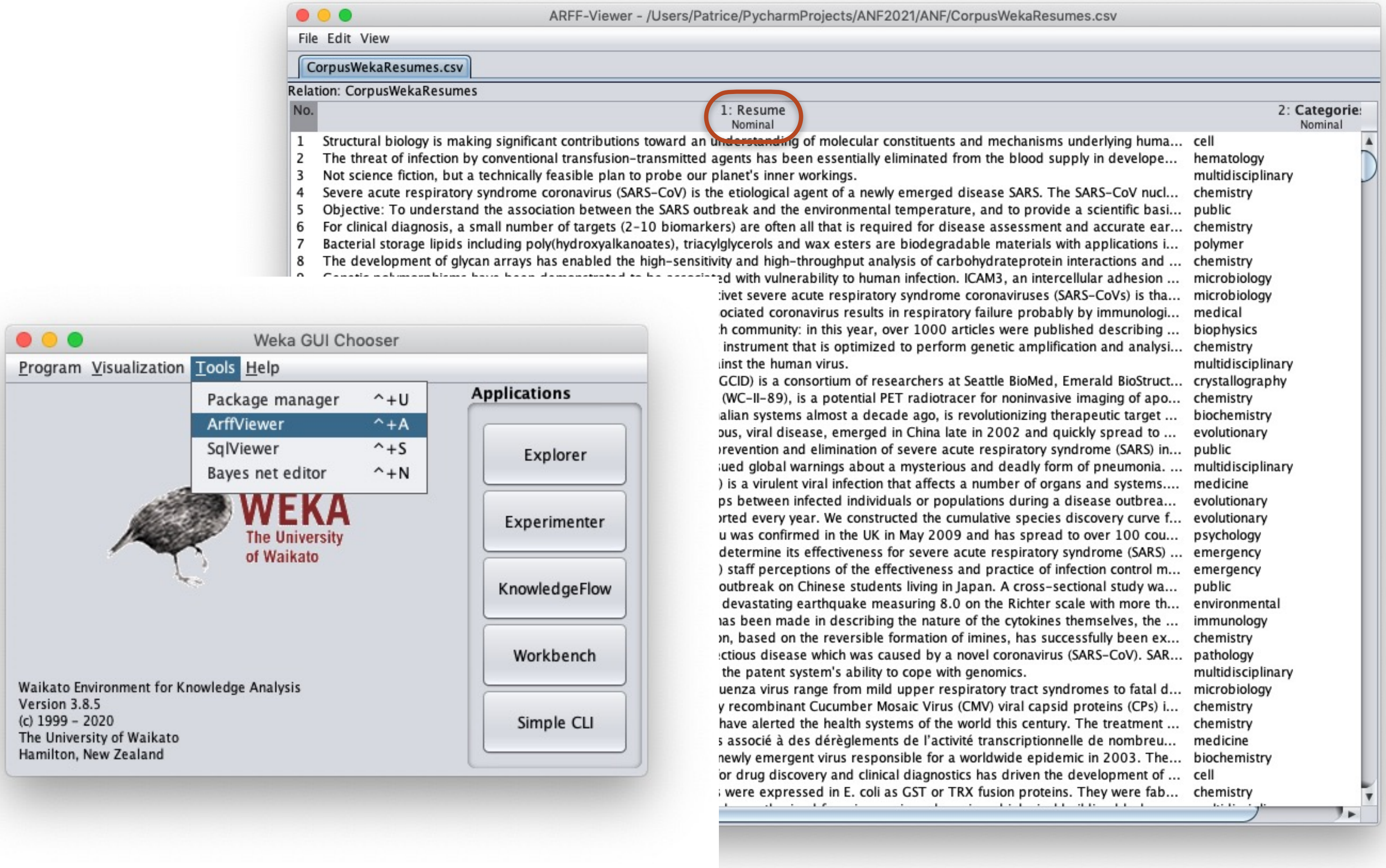
On enregistre en .csv « Weka »

```
resumesClasses.to_csv(fichierSortie,sep=',',na_rep='?', index = False, header=True, quoting=csv.QUOTE_NO
quotechar='"', doublequote=False, escapechar='\\')
```

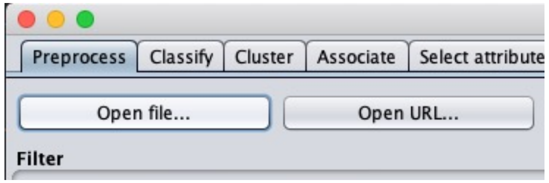
ça ne suffit pas… les types ne sont pas spécifiés

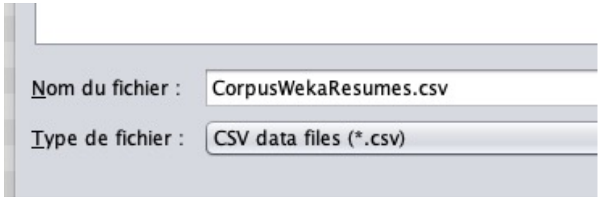
Three attribute types are supported:

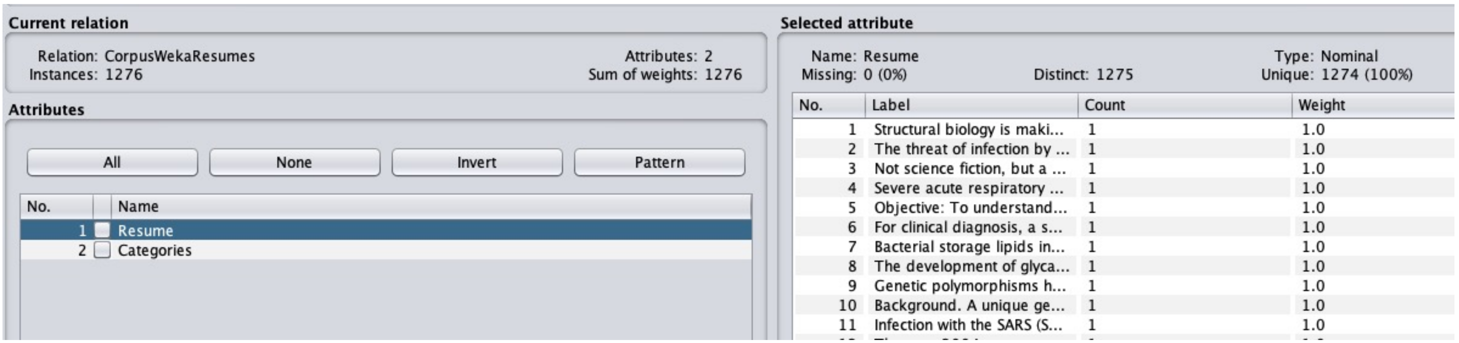
- numeric:
  This type of attribute represents a floating-point number.
- nominal:
  This type of attribute represents a fixed set of nominal values.
- string:
  This type of attribute represents a dynamically expanding set of nominal values.



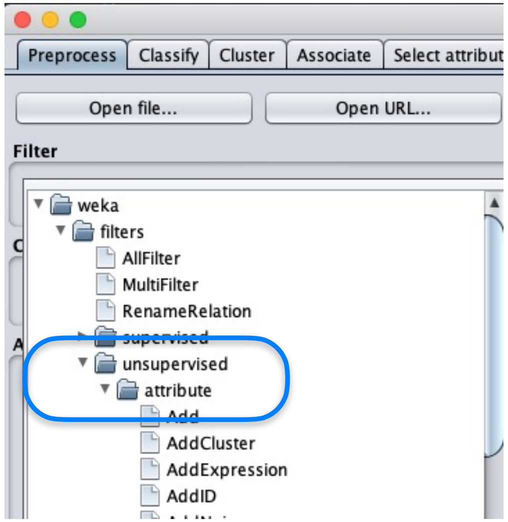
Ouverture du .csv dans Weka « Explorer »

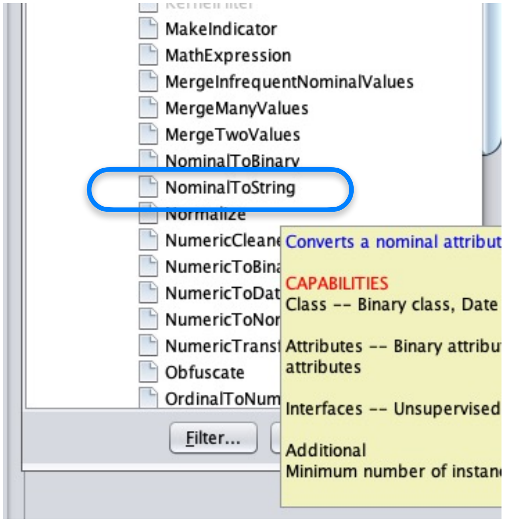


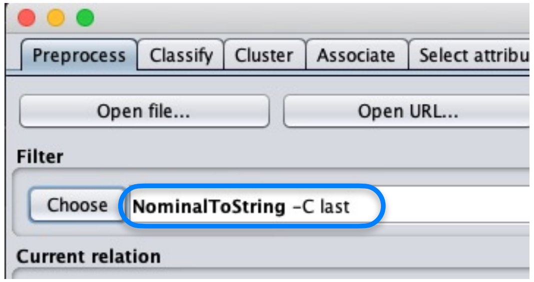


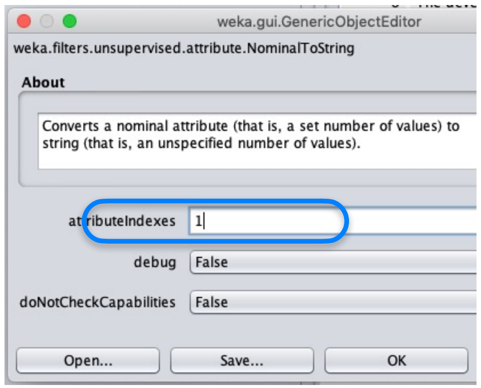


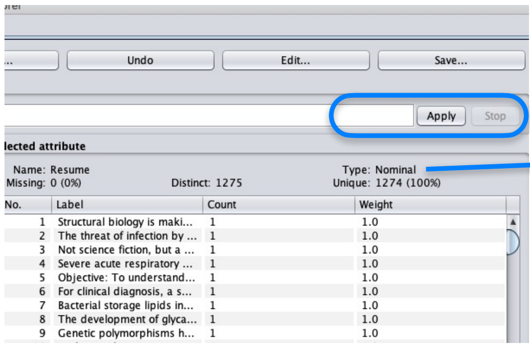
Conversion de l’attribut Résumé en « string »

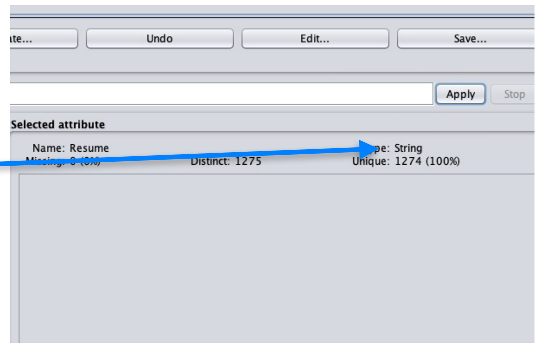




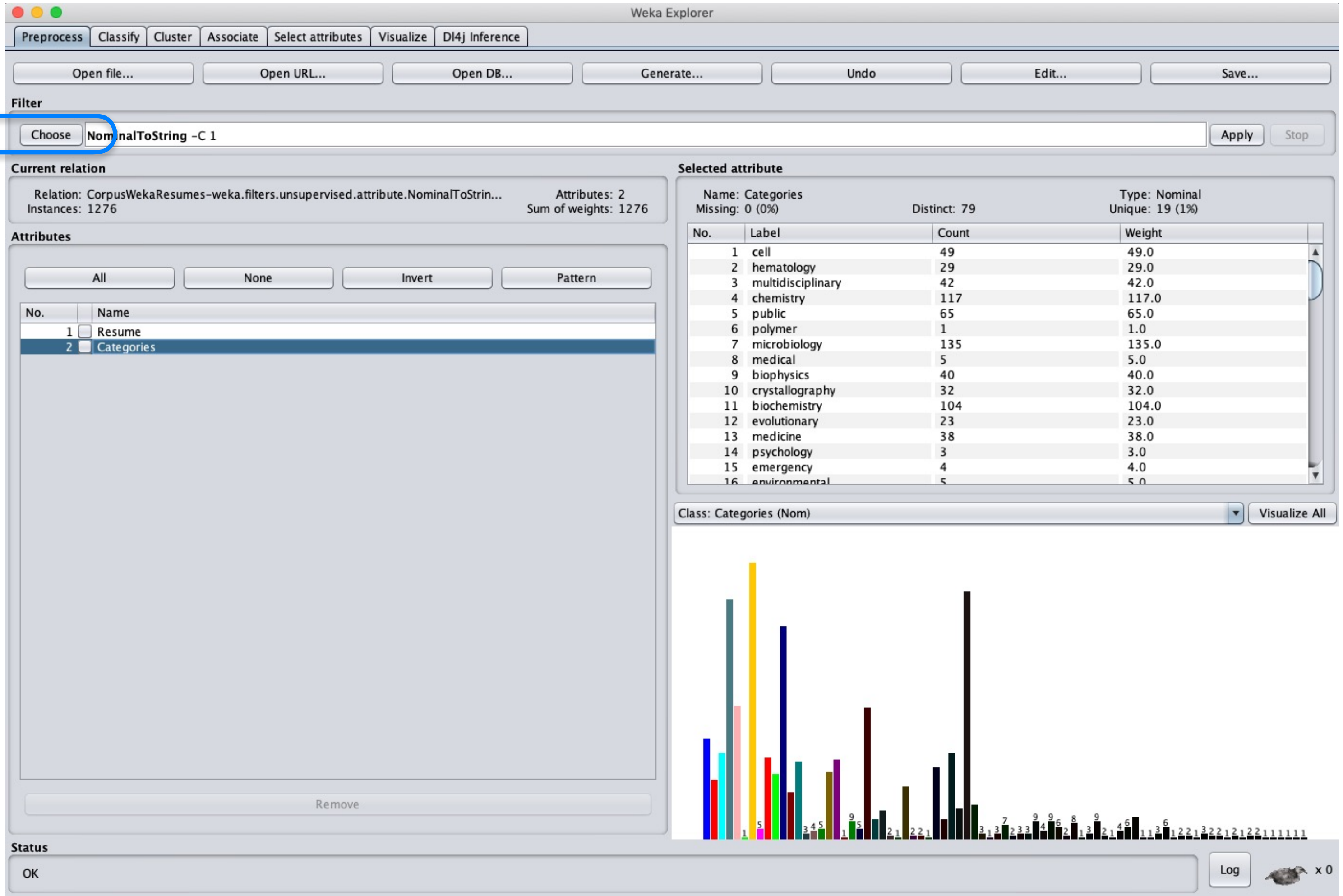




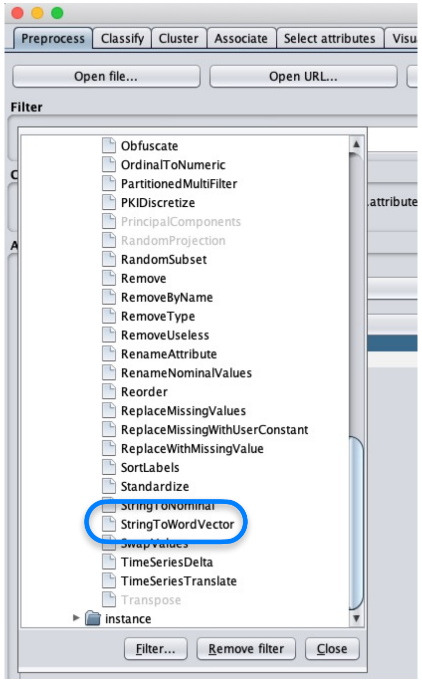




Reste à vectoriser les résumés avec un filtre adapté

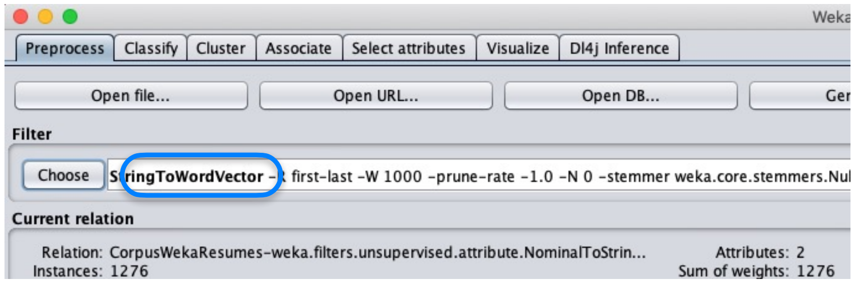


1



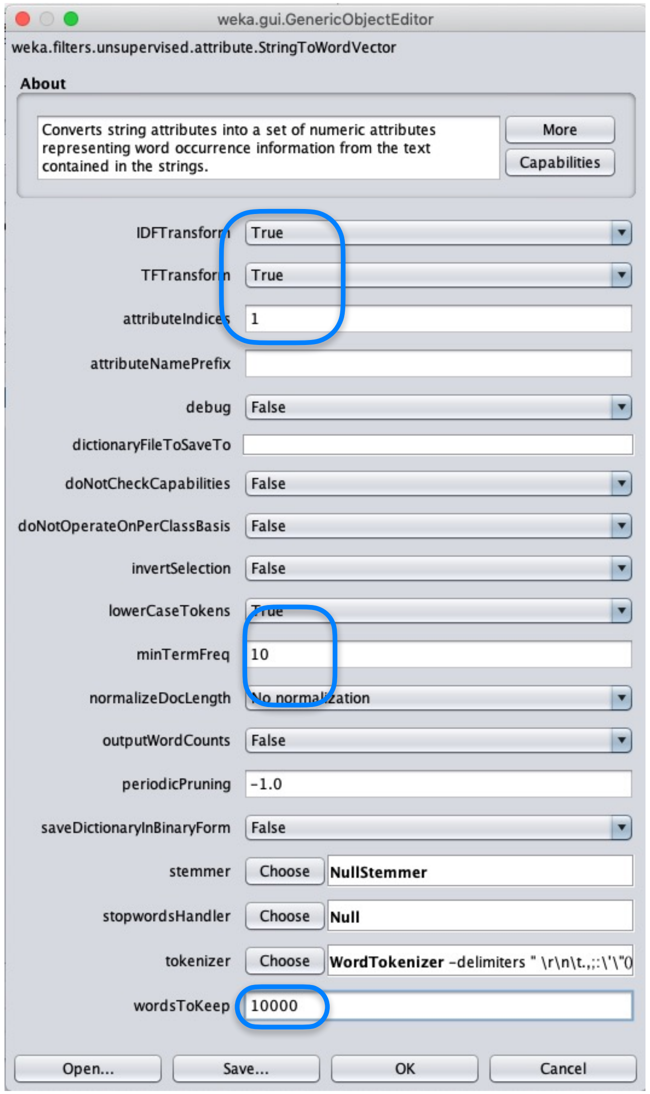
Preprocess
Classify
Cluster
Associate
Select attributes
Open file...
Open URL...
Filter
Obfuscate
OrdinalToNumeric
PartitionedMultiFilter
PKIDiscretize
PrincipalComponents
RandomProjection
RandomSubset
Remove
RemoveByName
RemoveType
RemoveUseless
RenameAttribute
RenameNominalValues
Reorder
ReplaceMissingValues
ReplaceMissingWithUserConstant
ReplaceWithMissingValue
SortLabels
Standardize
StringToWordVector
TimeSeriesDelta
TimeSeriesTranslate
Transpose
instance
Filter...
Remove filter
Close

2



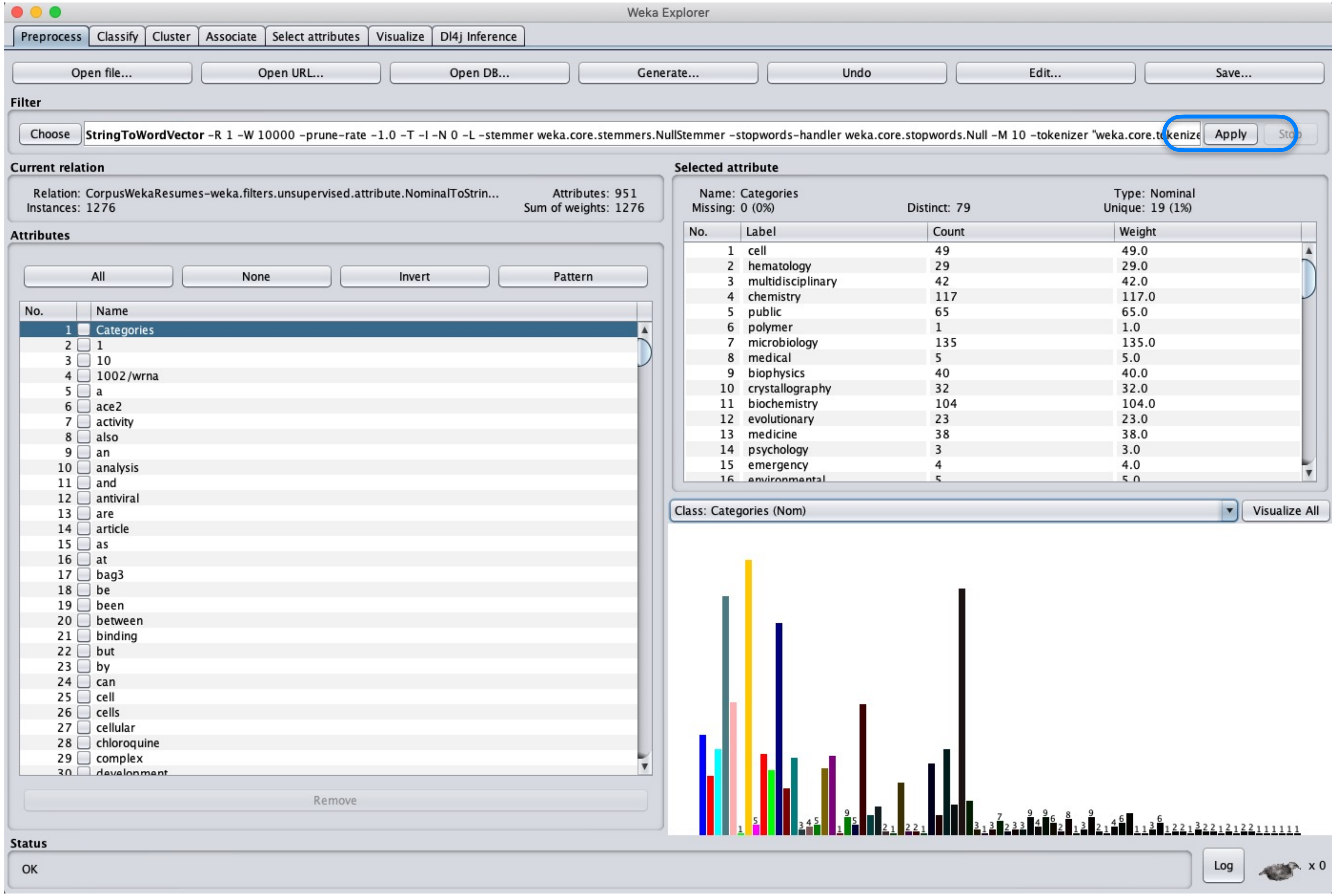
Weka
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Dl4j Inference
Open file...
Open URL...
Open DB...
Filter
Choose
first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.Nul
Current relation
Relation: CorpusWekaResumes-weka.filters.unsupervised.attribute.NominalToStrin...
Attributes: 2
Instances: 1276
Sum of weights: 1276

3

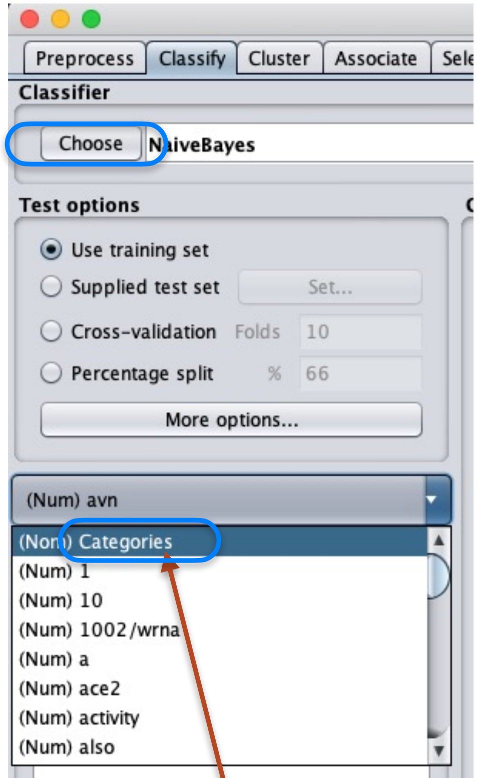


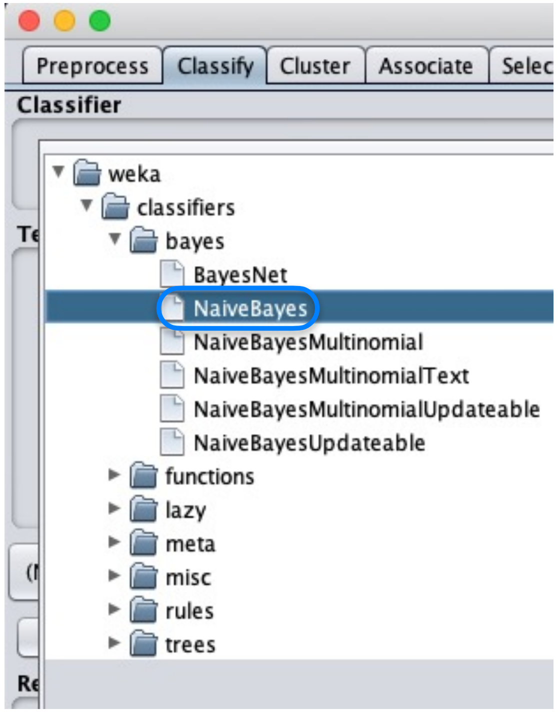
weka.gui.GenericObjectEditor
weka.filters.unsupervised.attribute.StringToWordVector
About
Converts string attributes into a set of numeric attributes representing word occurrence information from the text contained in the strings.
More
Capabilities
IDFTransform True
TFTransform True
attributeIndices 1
attributeNamePrefix
debug False
dictionaryFileToSaveTo
doNotCheckCapabilities False
doNotOperateOnPerClassBasis False
invertSelection False
lowerCaseTokens True
minTermFreq 10
normalizeDocLength No normalization
outputWordCounts False
periodicPruning -1.0
saveDictionaryInBinaryForm False
stemmer Choose NullStemmer
stopwordsHandler Choose Null
tokenizer Choose WordTokenizer -delimiters " \r\n\t.,;:\'\"(
wordsToKeep 10000
Open...
Save...
OK
Cancel

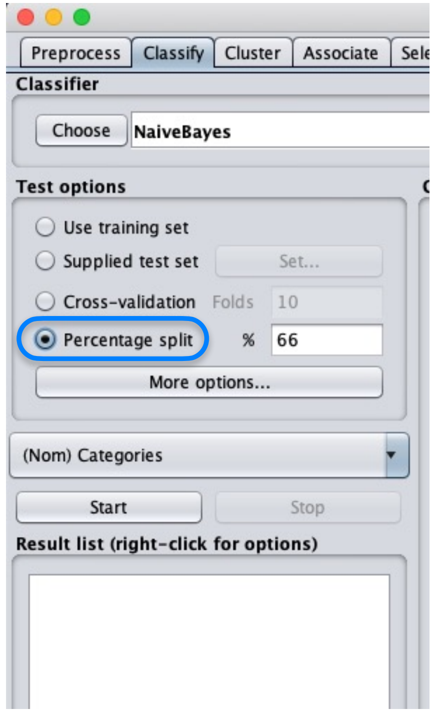
On applique StringToWordVector : il y a autant d'attributs que de mots



Utilisation d’un classifieur bayésien pour apprendre à prédire les catégories







Attention à
bien choisir l’attribut
à prédire

$$X = (x_1, x_2, \ldots, x_n) \longrightarrow classe$$

$$X = (x_1, x_2, \ldots, x_n) \longrightarrow classe$$
$$X = (x_1, x_2, \ldots, x_n) \longrightarrow classe$$
$$X = (x_1, x_2, \ldots, x_n) \longrightarrow classe$$

Données d'apprentissage

connaissance a priori

Max ? $$P(classe|X) = \frac{\boxed{P(X|classe)}\,P(classe)}{P(X)}$$

La probabilité de chaque classe <u>candidate</u>
Autant de scores que de classes

<u>inutile</u> pour comparer les P(classe)

Classifieur bayésien (règle de Bayes)

avec :

les x sont les descripteurs (*features*) de l'individu à classer

$$P(X|classe) = \prod_i P(x_1|classe) \times \ldots \times P(x_i|classe) \times \ldots \times P(x_n|classe)$$

le modèle appris

{ la fréquence avec laquelle on observe x_1 dans la classe parmi les exemples (données d'apprentissage) .... la fréquence avec laquelle on observe x_n dans la classe parmi les exemples (données d'apprentissage)

Classifier output

```
Time taken to build model: 0.36 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 6.46 seconds

=== Summary ===

Correctly Classified Instances         209               48.1567 %
Incorrectly Classified Instances       225               51.8433 %
Kappa statistic                          0.4467
Mean absolute error                      0.013
Root mean squared error                  0.111
Relative absolute error                 54.1205 %
Root relative squared error            101.4383 %
Total Number of Instances              434

=== Detailed Accuracy By Class ===
```

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,357	0,031	0,278	0,357	0,313	0,289	0,883	0,307	cell
0,667	0,002	0,857	0,667	0,750	0,751	0,923	0,746	hematology
0,300	0,042	0,143	0,300	0,194	0,180	0,815	0,172	multidisciplinar
0,476	0,043	0,541	0,476	0,506	0,458	0,886	0,620	chemistry
0,633	0,037	0,559	0,633	0,594	0,563	0,912	0,578	public
0,000	0,007	0,000	0,000	0,000	-0,004	0,894	0,021	polymer
0,585	0,055	0,596	0,585	0,590	0,534	0,889	0,625	microbiology
0,000	0,000	?	0,000	?	?	0,365	0,005	medical
0,706	0,010	0,750	0,706	0,727	0,717	0,935	0,780	biophysics
0,500	0,000	1,000	0,500	0,667	0,702	0,985	0,892	crystallography
0,750	0,068	0,500	0,750	0,600	0,570	0,950	0,753	biochemistry
0,600	0,021	0,250	0,600	0,353	0,377	0,983	0,684	evolutionary
0,263	0,000	1,000	0,263	0,417	0,505	0,688	0,327	medicine
0,000	0,000	?	0,000	?	?	0,866	0,092	psychology
0,000	0,000	?	0,000	?	?	0,891	0,031	emergency
0,000	0,000	?	0,000	?	?	0,995	0,333	environmental

Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize | Dl4j Inference

Classifier

Choose NaiveBayes

Test options

- (•) Use training set
- ( ) Supplied test set — Set...
- ( ) Cross-validation — Folds 10
- ( ) Percentage split — % 66

More options...

(Nom) Categories

Start | Stop

Result list (right-click for options)

- 19:12:44 - bayes.NaiveBayes
- 19:15:35 - bayes.NaiveBayes

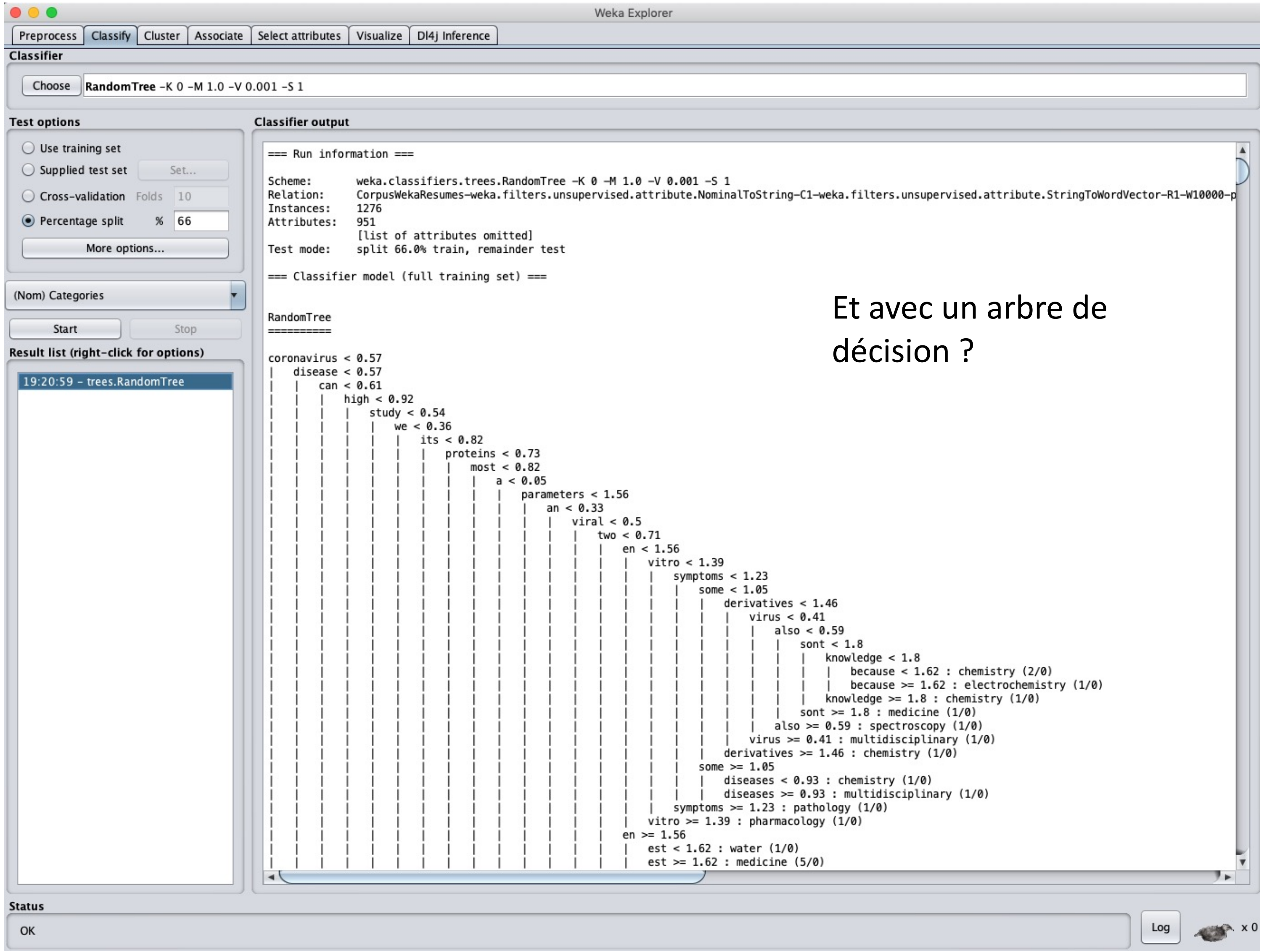
Classifier output

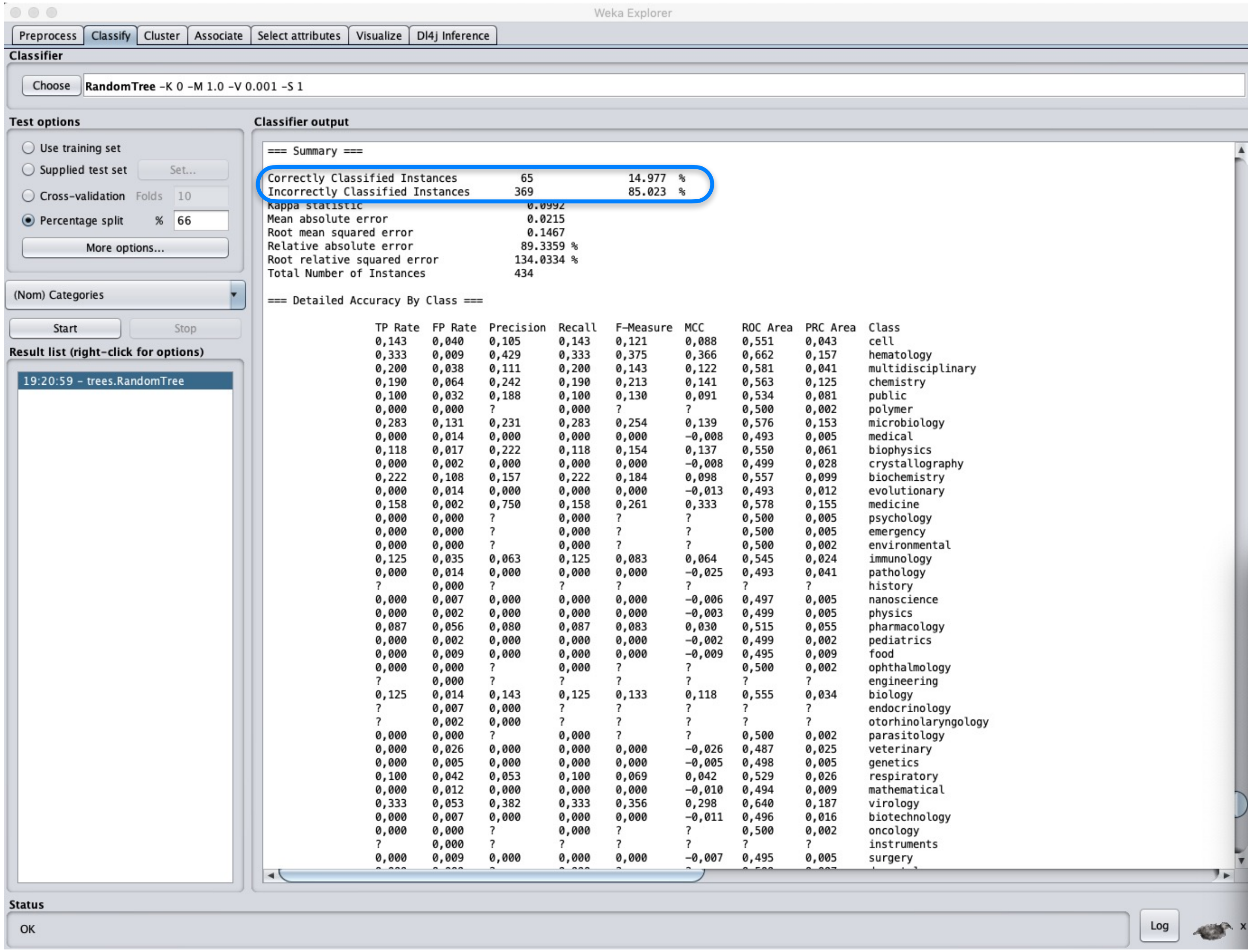
```
Time taken to build model: 0.38 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 19.13 seconds

=== Summary ===

Correctly Classified Instances        1079               84.5611 %
Incorrectly Classified Instances       197               15.4389 %
Kappa statistic                          0.8374
Mean absolute error                      0.0039
Root mean squared error                  0.0597
Relative absolute error                 16.1586 %
Root relative squared error             54.4993 %
Total Number of Instances             1276

=== Detailed Accuracy By Class ===
```

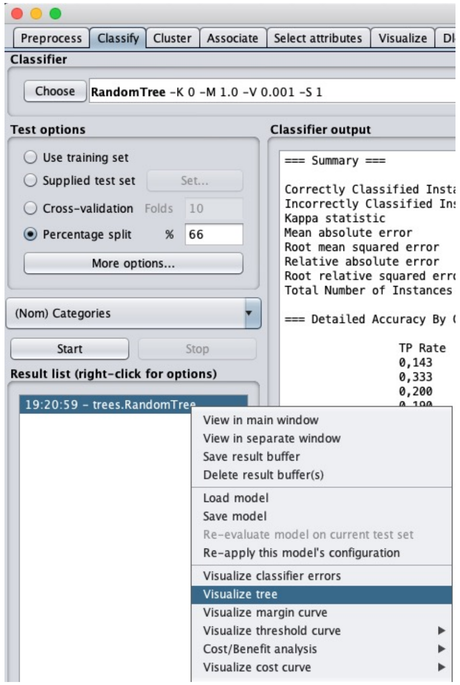
TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,735	0,012	0,706	0,735	0,720	0,709	0,982	0,803	cell
0,966	0,000	1,000	0,966	0,982	0,982	1,000	0,989	hematology
0,714	0,024	0,508	0,714	0,594	0,587	0,979	0,702	multidisciplinary
0,735	0,010	0,878	0,735	0,800	0,785	0,978	0,881	chemistry
0,892	0,010	0,829	0,892	0,859	0,852	0,989	0,920	public
1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	polymer
0,807	0,027	0,779	0,807	0,793	0,768	0,974	0,866	microbiology
0,800	0,000	1,000	0,800	0,889	0,894	1,000	1,000	medical
0,900	0,004	0,878	0,900	0,889	0,885	0,998	0,969	biophysics
0,906	0,000	1,000	0,906	0,951	0,951	1,000	0,998	crystallography
0,865	0,023	0,769	0,865	0,814	0,799	0,991	0,895	biochemistry
1,000	0,002	0,920	1,000	0,958	0,958	1,000	0,994	evolutionary
0,605	0,001	0,958	0,605	0,742	0,756	0,976	0,830	medicine

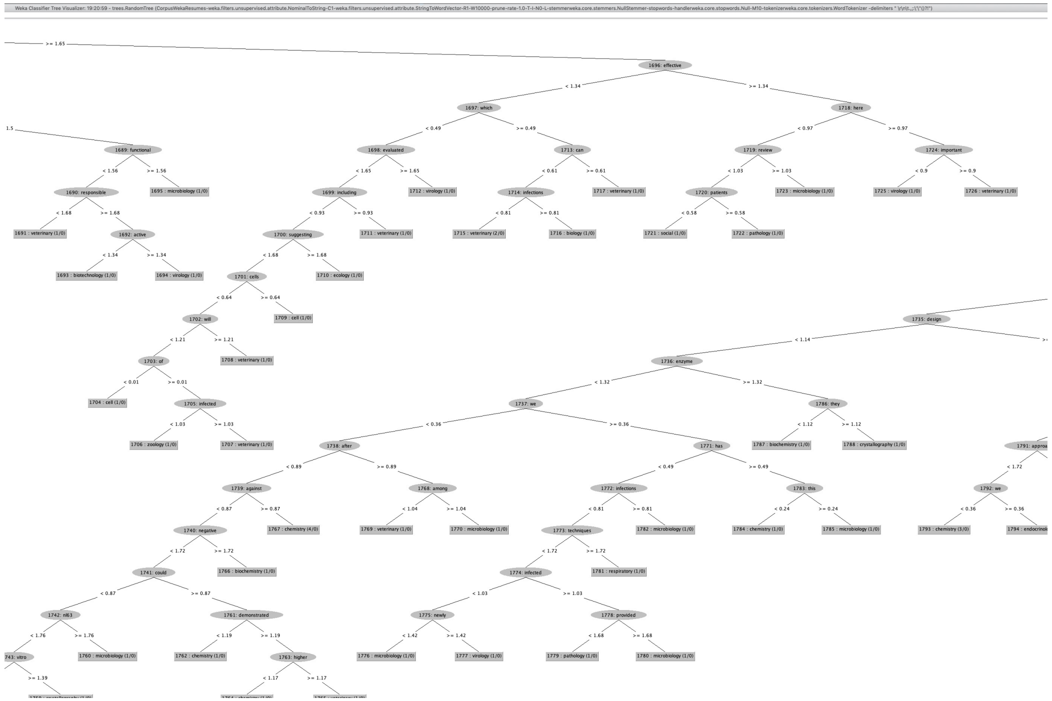
à comparer avec 48% avec 2/3 — 1/3 (test)

Et avec un arbre de décision ?




Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Dl4j Inference
Classifier
Choose
RandomTree -K 0 -M 1.0 -V 0.001 -S 1
Test options
Use training set
Supplied test set
Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) Categories
Start
Stop
Result list (right-click for options)
19:20:59 - trees.RandomTree
Classifier output
=== Summary ===
Correctly Classified Instances 65 14.977 %
Incorrectly Classified Instances 369 85.023 %
Kappa statistic 0.0992
Mean absolute error 0.0215
Root mean squared error 0.1467
Relative absolute error 89.3359 %
Root relative squared error 134.0334 %
Total Number of Instances 434
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0,143 0,040 0,105 0,143 0,121 0,088 0,551 0,043 cell
0,333 0,009 0,429 0,333 0,375 0,366 0,662 0,157 hematology
0,200 0,038 0,111 0,200 0,143 0,122 0,581 0,041 multidisciplinary
0,190 0,064 0,242 0,190 0,213 0,141 0,563 0,125 chemistry
0,100 0,032 0,188 0,100 0,130 0,091 0,534 0,081 public
0,000 0,000 ? 0,000 ? ? 0,500 0,002 polymer
0,283 0,131 0,231 0,283 0,254 0,139 0,576 0,153 microbiology
0,000 0,014 0,000 0,000 0,000 -0,008 0,493 0,005 medical
0,118 0,017 0,222 0,118 0,154 0,137 0,550 0,061 biophysics
0,000 0,002 0,000 0,000 0,000 -0,008 0,499 0,028 crystallography
0,222 0,108 0,157 0,222 0,184 0,098 0,557 0,099 biochemistry
0,000 0,014 0,000 0,000 0,000 -0,013 0,493 0,012 evolutionary
0,158 0,002 0,750 0,158 0,261 0,333 0,578 0,155 medicine
0,000 0,000 ? 0,000 ? ? 0,500 0,005 psychology
0,000 0,000 ? 0,000 ? ? 0,500 0,005 emergency
0,000 0,000 ? 0,000 ? ? 0,500 0,002 environmental
0,125 0,035 0,063 0,125 0,083 0,064 0,545 0,024 immunology
0,000 0,014 0,000 0,000 0,000 -0,025 0,493 0,041 pathology
? 0,000 ? ? ? ? ? ? history
0,000 0,007 0,000 0,000 0,000 -0,006 0,497 0,005 nanoscience
0,000 0,002 0,000 0,000 0,000 -0,003 0,499 0,005 physics
0,087 0,056 0,080 0,087 0,083 0,030 0,515 0,055 pharmacology
0,000 0,002 0,000 0,000 0,000 -0,002 0,499 0,002 pediatrics
0,000 0,009 0,000 0,000 0,000 -0,009 0,495 0,009 food
0,000 0,000 ? 0,000 ? ? 0,500 0,002 ophthalmology
? 0,000 ? ? ? ? ? ? engineering
0,125 0,014 0,143 0,125 0,133 0,118 0,555 0,034 biology
? 0,007 0,000 ? ? ? ? ? endocrinology
? 0,002 0,000 ? ? ? ? ? otorhinolaryngology
0,000 0,000 ? 0,000 ? ? 0,500 0,002 parasitology
0,000 0,026 0,000 0,000 0,000 -0,026 0,487 0,025 veterinary
0,000 0,005 0,000 0,000 0,000 -0,005 0,498 0,005 genetics
0,100 0,042 0,053 0,100 0,069 0,042 0,529 0,026 respiratory
0,000 0,012 0,000 0,000 0,000 -0,010 0,494 0,009 mathematical
0,333 0,053 0,382 0,333 0,356 0,298 0,640 0,187 virology
0,000 0,007 0,000 0,000 0,000 -0,011 0,496 0,016 biotechnology
0,000 0,000 ? 0,000 ? ? 0,500 0,002 oncology
? 0,000 ? ? ? ? ? ? instruments
0,000 0,009 0,000 0,000 0,000 -0,007 0,495 0,005 surgery
Status
OK
Log


Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
RandomTree -K 0 -M 1.0 -V 0.001 -S 1
Test options
Use training set
Supplied test set
Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) Categories
Start
Stop
Result list (right-click for options)
19:20:59 - trees.RandomTree
Classifier output
=== Summary ===
Correctly Classified Insta
Incorrectly Classified In
Kappa statistic
Mean absolute error
Root mean squared error
Relative absolute error
Root relative squared err
Total Number of Instances
=== Detailed Accuracy By
TP Rate
0,143
0,333
0,200
View in main window
View in separate window
Save result buffer
Delete result buffer(s)
Load model
Save model
Re-evaluate model on current test set
Re-apply this model's configuration
Visualize classifier errors
Visualize tree
Visualize margin curve
Visualize threshold curve
Cost/Benefit analysis
Visualize cost curve



Classification non supervisée (avec Python)

— **Objectif** : réunir les documents en fonction de leurs similarités et visualiser les classes obtenues

— **Moyens** :

  — algorithmes de partitionnement tels que les k-Moyennes ou les cartes auto-organisées

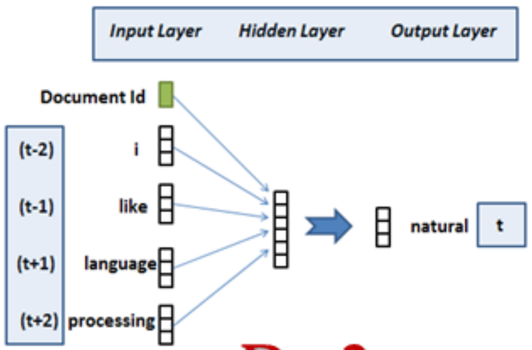
  — visualisation par ACP

— Espace initial en très grande dimension (la taille du vocabulaire) :

  — réunir les mots similaires = projeter les documents sur un espace réduit

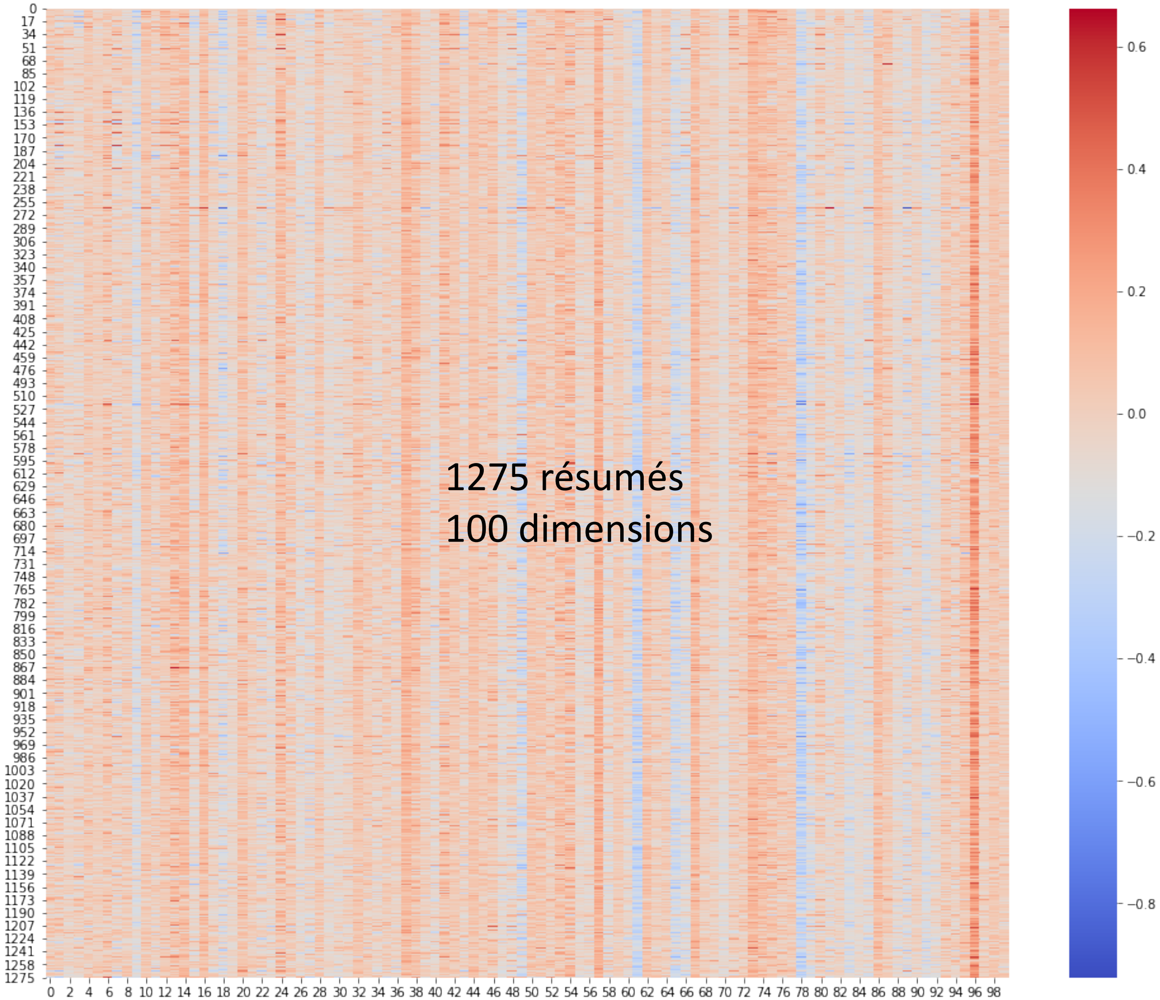
GENSIM

python



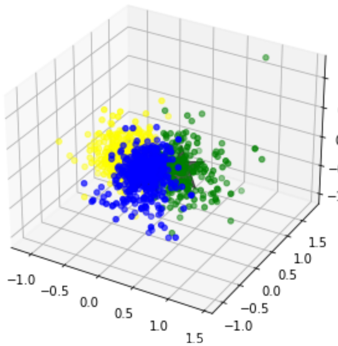
Doc2vec

```python
def doc2vect():
    document_tagged  = []
    tagged_count = 0
    for _ in resumesClasses['Resume'].values:
        document_tagged.append(gensim.models.doc2vec.TaggedDocument(_,[tagged_count]))
        tagged_count += 1
    d2v = Doc2Vec(document_tagged)
    return d2v.docvecs.vectors_docs
```

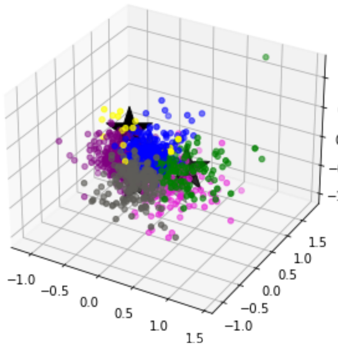

1275 résumés
100 dimensions
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

```python
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA  PCA: <cla

kmean_model = KMeans(n_clusters=15, n_jobs=-1)
%time km = kmean_model.fit_predict(doc2vec)
```

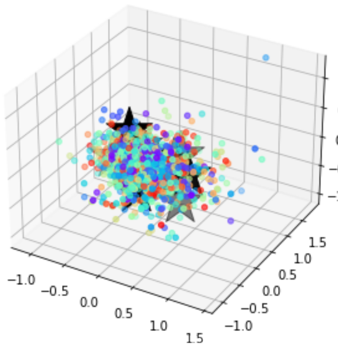
3 classes

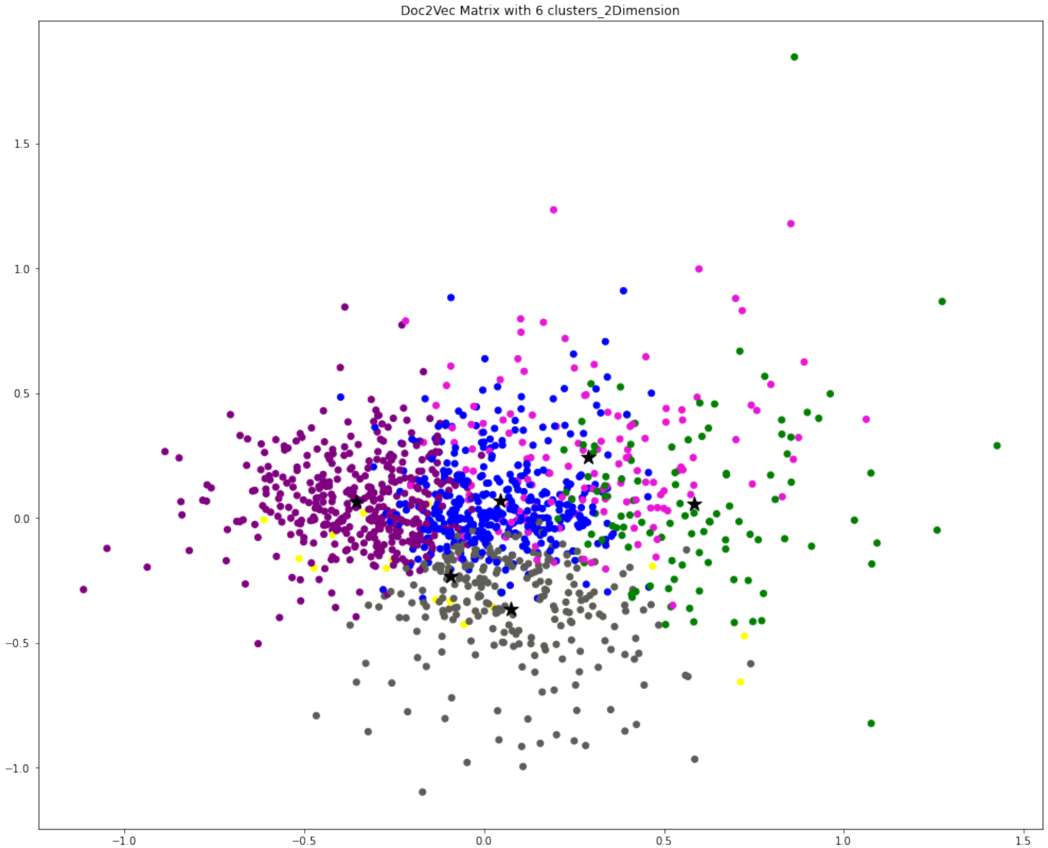


6 classes



15 classes

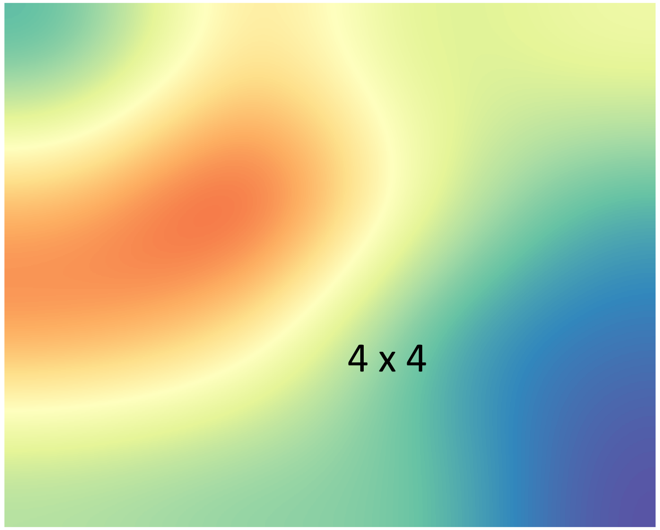


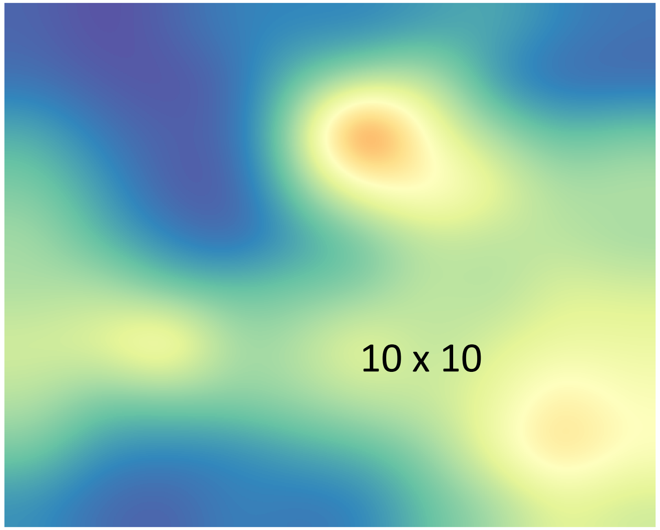


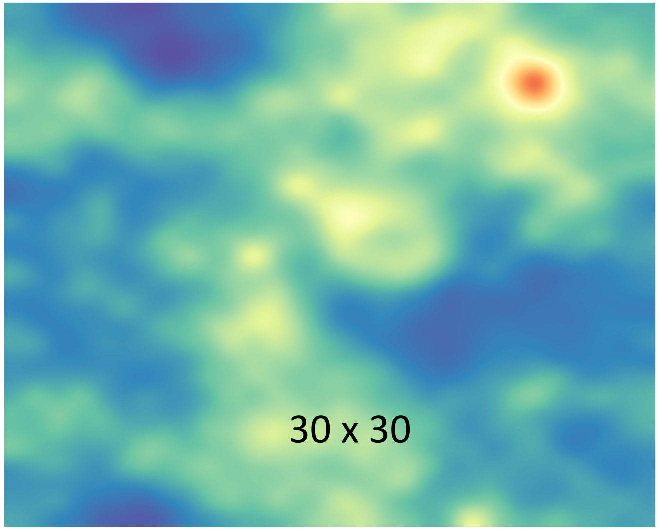
k-Means puis ACP pour visualiser les classes

```
som = somoclu.Somoclu(4, 4, maptype="toroid")
```

```
som.train(doc2vec)
```







Cartes auto-organisées
(SOM)