1 Introduction

Instanciation de l'algorithme ''Word2Vec'' dans un problème de text mining. Représentation des termes et des documents. Utilisation de la librairie H2O pour Python. Utilisation également de la librairie NLTK.

Ce tutoriel fait suite au support de cours consacré au prolongement lexical (word embedding) où nous avions étudié l'algorithme ''Word2Vec'' dans le cadre de la fouille de textes (text mining ; on parle aussi de NLP, natural language processing). Nous mettons en œuvre la technique sur un jeu de données jouet tiré de l'ouvrage de Coelho et Richert (2015). Le premier objectif est de représenter les termes du corpus dans un espace de dimension réduite en les contextualisant c.-à-d. en tenant compte de leur voisinage. Le second consiste à calculer les coordonnées des documents pour apprécier leurs proximités dans ce nouvel espace de représentation ainsi défini.

Nous nous appuyons sur la librairie H2O pour Python. Nous l'avions déjà exploré à plusieurs reprises précédemment (par ex. ''Machine Learning avec H2O'', janvier 2019). L'enjeu dans notre contexte est de savoir préparer correctement le corpus pour que l'on puisse faire appel aux fonctions dédiées. Cette tâche est quand-même assez particulière sous H20. Nous y porterons toute notre attention – de la manière la plus didactique possible, voire scolaire – pour ne pas perdre le lecteur en route. Il faut dire (je trouve) que les tutoriels que l'on peut trouver ici ou là sur le web sont plus qu'elliptiques sur le sujet, rendant très difficile la reproductibilité de la démarche sur nos propres données.

2 Données – Corpus

2.1 Description du corpus

Le corpus est issu de l'ouvrage de Coelho et Richert (2015). Il est composé de 5 phrases (documents) relativement simples (Remarque : les numéros de documents ne figurent pas dans le fichier ''**toy_posts.txt**'' à traiter)

(0) This is a toy post about machine learning. Actually, it contains not much interesting stuff.
(1) Imaging databases provide storage capabilities.
(2) Most imaging databases save images permanently.
(3) Imaging databases store data.
(4) Imaging databases store data. Imaging databases store data. Imaging databases store data.

Quelques commentaires :

- Le 1er document (n°0) n'a absolument rien à voir avec les autres.
- La dernière (n°4) est identique à la précédente (n°3), mais dupliquée 3 fois.

Voyons comment l'algorithme ''word2vec'' va réagir face à ces données.

2.2 Importation du corpus

Nous chargeons les corpus en utilisant les outils standards de Python (**3.7.4**).

```
#changement de répertoire
import os
os.chdir("... votre dossier ...")

#chargement des posts dans une liste
f = open("toy_posts.txt","r")
liste = f.readlines()
f.close()

#nombre de posts
print(len(liste))
```

```
5
```

Nous les affichons.

```
#contenu des posts
print(liste)
```

```
['This is a toy post about machine learning. Actually, it contains not much interesting stuff.\n', 'Imaging databases provide storage capabilities.\n', 'Most imaging databases save images permanently.\n', 'Imaging databases store data.\n', 'Imaging databases store data. Imaging databases store data. Imaging databases store data.\n']
```

Nous avons une structure de liste. Nous noterons la présence du caractère saut de ligne ''**\n**'' au bout de chaque document.

2.3 Prétraitement – Nettoyage du corpus

Le nettoyage des données est crucial dans le traitement du langage naturel. Nous réalisons une série de transformations sur notre corpus. Je vais détailler chaque étape pour que l'on comprenne bien la finalité et le résultat de chaque opération. Remarque : C'est peut-être le reproche que je ferais aux tutoriels que l'on peut trouver ici ou là sur le web. La majorité effectuent les manipulations d'une traite, sans trop de commentaires, il est très facile de s'y perdre et, surtout, dès que l'on change de corpus, plus rien ne marche. Dans ces conditions, le découragement nous envahit très vite.

Retrait des ''\n''. Nous retirons le caractère saut de ligne tout d'abord.

```
#enlever les "\n"
posts = [s.replace("\n","") for s in liste]
print(posts)
```

```
['This is a toy post about machine learning. Actually, it contains not much interesting stuff.', 'Imaging databases provide storage capabilities.', 'Most imaging databases save images permanently.', 'Imaging databases store data.', 'Imaging databases store data. Imaging databases store data. Imaging databases store data.']
```

Retrait des ponctuations. Pour identifier les ponctuations, nous en récupérons la liste via le module ''string'', puis nous retirons des documents les caractères qui pourraient leur correspondre. Voici la liste des ponctuations :

```
#liste des ponctuations - récupérée de la classe ''string''
import string
ponctuations = list(string.punctuation)
print(ponctuations)
```

```
['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`', '{', '|', '}', '~']
```

Nous souhaitons les retirer de notre corpus. Les informaticiens aiment bien les instructions de la mort qui font tout en une seule ligne de code. Essayons de détailler les opérations clés de la commande ci-dessous : chaque document (msg) est transformé en liste de caractères ; chaque caractère (w) n'est conservé que s'il n'apparaît pas parmi les ponctuations ; la liste résultante est refusionnée (.join) pour former la chaîne de caractères correspondant au document.

```
#retirer les ponctuations
posts = ["".join([w for w in list(msg) if not w in ponctuations]) for msg in posts]
print(posts)
```

```
['This is a toy post about machine learning Actually it contains not much interesting stuff', 'Imaging databases provide storage capabilities', 'Most imaging databases save images permanently', 'Imaging databases store data', 'Imaging databases store data Imaging databases store data Imaging databases store data']
```

Dans notre cas, seuls ''.'' et '','' ont été retirés. Mais l'instruction est générique et peut traiter les différentes ponctuations énumérées dans la variable ''**ponctuations**''.

Harmonisation de la casse. Nous passons tous les caractères en minuscules.

```
#mettre tout en minuscule
posts = [msg.lower() for msg in posts]
print(posts)
```

```
['this is a toy post about machine learning actually it contains not much interesting stuff', 'imaging databases provide storage capabilities', 'most imaging databases save images permanently', 'imaging databases store data', 'imaging databases store data imaging databases store data imaging databases store data']
```

Retrait des stopwords. Les stopwords (mots-vides) sont les termes communs qui ne sont pas porteurs de sens (articles, pronoms, etc.). Il n'est pas nécessaire de les conserver dans le cadre de notre analyse. Nous travaillons en deux temps : nous chargeons une liste de stopwords pour la langue anglaise, puis nous les retirons de nos documents.

La librairie NLTK nous fournit la liste des stopwords. Attention : il faut tout d'abord l'installer si elle n'est pas disponible d'emblée

```
#installation -- si la liste des stopwords n'est pas disponible
import nltk
nltk.download() #cf. CORPORA / STOPWORDS

#puis chargement des stopwords
from nltk.corpus import stopwords
mots_vides = stopwords.words('english')
print(mots_vides)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's",
'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which',
'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are',
'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from',
'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',
'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will',
'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're',
've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't",
'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't",
'won', "won't", 'wouldn', "wouldn't"]
```

Il nous est possible d'inspecter la liste et de rajouter d'autres termes si cela paraît nécessaire par rapport aux spécificités du corpus que nous traitons.

Il faut transformer les documents en liste de termes par tokenisation avant de pouvoir retirer les mots-vides. La librairie NLTK fait encore des merveilles ici.

```
#transformer les messages en listes par tokenisation
from nltk.tokenize import word_tokenize
posts = [word_tokenize(msg)  for msg in posts]
print(posts)
```

Le corpus se présente comme une liste (corpus) de listes (5 documents) à présent :

```
[['this', 'is', 'a', 'toy', 'post', 'about', 'machine', 'learning', 'actually',
'it', 'contains', 'not', 'much', 'interesting', 'stuff'], ['imaging', 'databases',
'provide', 'storage', 'capabilities'], ['most', 'imaging', 'databases', 'save',
'images', 'permanently'], ['imaging', 'databases', 'store', 'data'], ['imaging',
'databases', 'store', 'data', 'imaging', 'databases', 'store', 'data', 'imaging',
'databases', 'store', 'data']]
```

Nous éliminons les mots-vides des documents.

```
#retirer les stopwords et reformer les messages
posts = [[w for w in msg if not w in mots_vides] for msg in posts]
print(posts)
```

```
[['toy', 'post', 'machine', 'learning', 'actually', 'contains', 'much',
'interesting', 'stuff'], ['imaging', 'databases', 'provide', 'storage',
'capabilities'], ['imaging', 'databases', 'save', 'images', 'permanently'],
['imaging', 'databases', 'store', 'data'], ['imaging', 'databases', 'store',
'data', 'imaging', 'databases', 'store', 'data', 'imaging', 'databases', 'store',
'data']]
```

Pour le document n°0 par exemple, nous constatons que (''this'', ''is'', ''a'', etc.) ont été retirés.

Stemming. La racinisation (stemming) consiste à ramener les termes à leur radical. Dans nos documents, nous observons par exemple que les termes ''images'' et ''imaging'' traduisent le même concept. Les ramener au même ''token'' serait une bonne chose pour éviter la démultiplication de la dimensionnalité. L'algorithme de Porter fait référence dans le domaine, en particulier pour la langue anglaise. Elle est implémentée dans la librairie NLTK. Nous l'appliquons aux termes de notre corpus.

```
#outil pour stemming - racinisation des termes
from nltk.stem import PorterStemmer
ps = PorterStemmer()

#transformation des messages par racinisation des mots
posts = [[ps.stem(w) for w in msg] for msg in posts]
print(posts)
```

```
[['toy', 'post', 'machin', 'learn', 'actual', 'contain', 'much', 'interest',
'stuff'], ['imag', 'databas', 'provid', 'storag', 'capabl'], ['imag', 'databas',
'save', 'imag', 'perman'], ['imag', 'databas', 'store', 'data'], ['imag',
'databas', 'store', 'data', 'imag', 'databas', 'store', 'data', 'imag', 'databas',
'store', 'data']]
```

Nous remarquons ainsi que : ''machine'' a été ramené à ''machin'', ''learning'' à ''learn'', etc ; nous ne distinguons plus les différentes déclinaisons de ''imag''. La taille du dictionnaire – et donc la dimensionnalité pour les traitements à venir – sera réduite.

Nous reformons les documents à partir des listes de termes.

```
#reformer les messages à partir des listes
posts = [" ".join(msg) for msg in posts]
```

```
print(posts)
```

```
['toy post machin learn actual contain much interest stuff', 'imag databas provid storag capabl', 'imag databas save imag perman', 'imag databas store data', 'imag databas store data imag databas store data imag databas store data']
```

Nous sommes maintenant parés pour lancer l'algorithme Word2Vec de la librairie H2O.

3 Word2Vec avec H20

3.1 Phase préparatoire

3.1.1 Démarrage du serveur H2O

H2O est une plate-forme JAVA qui implémente nombre d'algorithmes de machine learning. Nous pouvons accéder à ces fonctionnalités via le mécanisme des API, notamment sous R (ex. ''Packages R pour le Deep Learning'', décembre 2018) ou sous Python (ex. ''Packages Python pour le Deep Learning'', décembre 2018). Après avoir installé la librairie sous Anaconda Python (je travaille avec Windows 10 - 64 bits - Version Education), nous l'importons et nous vérifions le numéro de version (précaution indispensable pour s'assurer de la reproductibilité des tutoriels).

```
#import h2o
import h2o
print(h2o.__version__)
```

```
3.26.0.10
```

Nous démarrons ensuite le serveur local.

```
#initialisation
h2o.init()
```

```
Checking whether there is an H2O instance running at http://localhost:54321 ..... not found.
Attempting to start a local H2O server...
; Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
  Starting server from D:\Logiciels\Anaconda3\lib\site-packages\h2o\backend\bin\h2o.jar
  Ice root: C:\Users\Zatovo\AppData\Local\Temp\tmp883khx8a
  JVM stdout: C:\Users\Zatovo\AppData\Local\Temp\tmp883khx8a\h2o_Zatovo_started_from_python.out
  JVM stderr: C:\Users\Zatovo\AppData\Local\Temp\tmp883khx8a\h2o_Zatovo_started_from_python.err
   Server is running at http://127.0.0.1:54321

Connecting to H2O server at http://127.0.0.1:54321 ... successful.
--------------------------  ------------------------------------------------------------
H2O cluster uptime:         03 secs
H2O cluster timezone:       Europe/Paris
H2O data parsing timezone:  UTC
H2O cluster version:        3.26.0.10
H2O cluster version age:    1 month
H2O cluster name:           H2O_from_python_Zatovo_yq10nf
H2O cluster total nodes:    1
H2O cluster free memory:    1.759 Gb
H2O cluster total cores:    8
H2O cluster allowed cores:  8
H2O cluster status:         accepting new members, healthy
H2O connection url:         http://127.0.0.1:54321
```

```
H2O connection proxy:       {'http': None, 'https': None}
H2O internal security:      False
H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
Python version:             3.7.4 final
--------------------------  ------------------------------------------------------------
```

Tout s'est très bien déroulé en ce qui me concerne, avec la version 3.26.0.10 de H2O.

3.1.2 Traduire le corpus au format H2O

Première tâche, il faut convertir notre corpus dans un format reconnu par H2O (H2OFrame).

```
#formation d'un Frame H2O - chaque ligne = un document
df = h2o.H2OFrame(posts)

#et transtyper en caractères -- !!! très important !!!
df = df.ascharacter()

#vérification dimension
df.describe()
```

```
Rows:5
Cols:1
```

Nous avons un data frame avec 5 lignes (5 documents) et une seule colonne, confirmé par l'affichage de contrôle (l'objet doit être transformé en data frame Pandas pour pouvoir être affiché).

```
#affichage de contrôle
print(df.as_data_frame())
```

```
                                                  C1
0  toy post machin learn actual contain much inte...
1                  imag databas provid storag capabl
2                      imag databas save imag perman
3                            imag databas store data
4  imag databas store data imag databas store dat...
```

3.1.3 Tokenisation au format H2O

Etape cruciale ensuite pour implémenter ''word2vec'' sous H2O, nous devons ''tokeniser'' les documents du corpus. H2O forme un vecteur très particulier…

```
#tokenisation - " " séparateur de mots
tokenized = df.tokenize(" ")
tokenized.describe()
```

… de 40 lignes (???) et 1 colonne.

Dans lequel…

```
#affichage - on a une seule colonne
#chaque document est séparé par un NaN
print(tokenized.as_data_frame())
```

```
          C1
0        toy
1       post
2     machin
3      learn
4     actual
5    contain
6       much
7   interest
8      stuff
9        NaN
10      imag
11   databas
12    provid
13    storag
14    capabl
15       NaN
16      imag
17   databas
18      save
19      imag
20    perman
21       NaN
22      imag
23   databas
24     store
25      data
26       NaN
27      imag
28   databas
29     store
30      data
31      imag
32   databas
33     store
34      data
35      imag
36   databas
37     store
38      data
39       NaN
```

… les documents sont vectorisés, avec pour séparateur le symbole NaN (not a number) pour les distinguer. Note : Cette transformation en vecteur est un passage obligé. Sinon, et j'ai tout testé je dois dire, il ne sera pas possible de lancer le calcul ''word2vec'' sous H2O.

3.2 Algorithme Word2Vec avec H20

A ce stade, nous pouvons importer et instancier la classe de calcul pour ''word2vec''.

```
#estimateur de word2vec - attention aux paramètres
from h2o.estimators.word2vec import H2OWord2vecEstimator
wv = H2OWord2vecEstimator(vec_size=2,min_word_freq=1,window_size=1,epochs=10000)
```

Un petit arrêt sur les paramètres :

- vec_size détermine le nombre de neurones (la dimensionalité de représentation) dans la couche cachée du réseau. Nous faisons le choix de (vec_size = 2) pour pouvoir représenter les termes dans le plan. En pratique, selon la complexité du problème à traiter, nous pouvons indiquer des dimensionalités largement plus importantes (de l'ordre de la centaine, voire plus…).
- min_word_freq indique la fréquence minimale d'un terme pour être inclus dans le calcul.
- window_size indique la taille du voisinage à prendre en compte. Spécifier la ''bonne'' valeur est tout sauf facile : trop faible, on risque de ne pas capter les influences croisées entre un terme et ses voisins ; trop élevée, nous risquons de diluer l'information.
- epochs enfin indique le nombre d'itérations sur la base de données. Vu la taille très faible de notre corpus, j'ai mis un epochs très élevé pour assurer la convergence du réseau.

Nous pouvons enfin lancer la modélisation sur notre corpus d'apprentissage.

```
#apprentissage
wv.train(training_frame=tokenized)
```

Etudions les résultats.

3.3 Coordonnées des termes et voisinage

Les coordonnées des termes dans le nouvel espace de représentation est le premier résultat attendu.

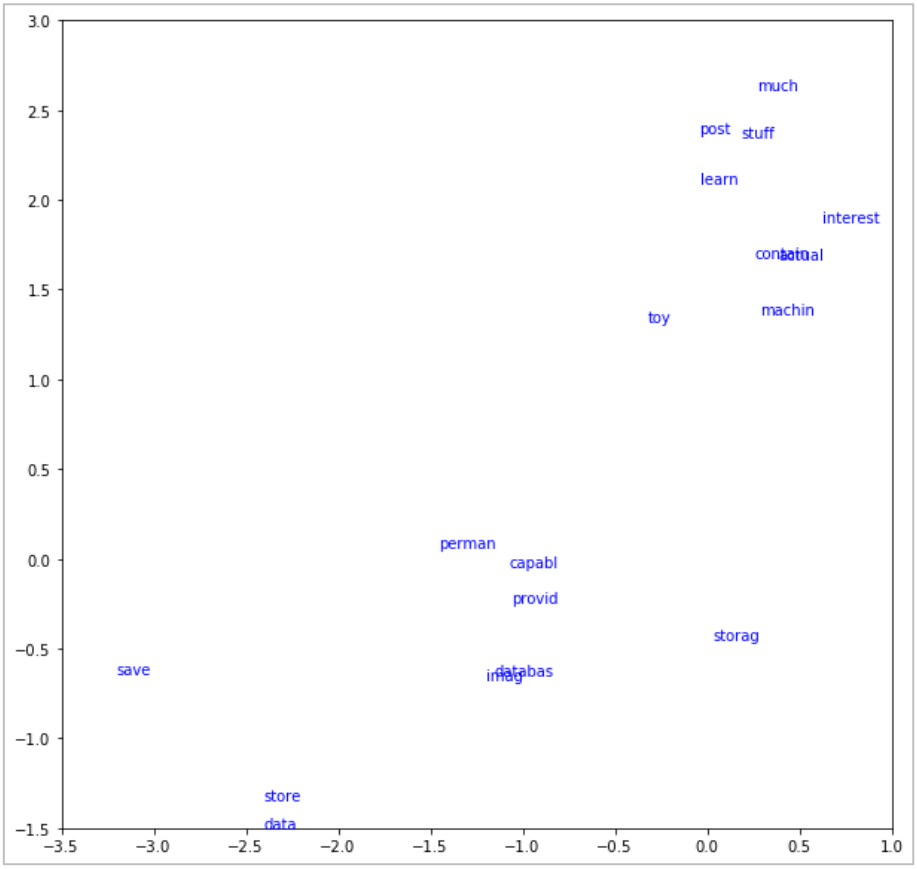
```
#description (vecteur) attribuée à chaque terme
terms = wv.to_frame().as_data_frame()
print(terms)
```

```
        Word        V1        V2
0       post -0.043074  2.370073
1       save -3.202884 -0.642050
2      learn -0.039290  2.093736
3     capabl -1.077052 -0.048554
4     provid -1.057952 -0.243790
5       much  0.273482  2.616317
6     storag  0.033232 -0.453390
7     machin  0.285956  1.359306
8        toy -0.318164  1.322068
9      stuff  0.187052  2.345181
10  interest  0.626550  1.872760
11   contain  0.254201  1.674144
12    actual  0.385817  1.669475
13    perman -1.453780  0.066561
14     store -2.403507 -1.346129
15      data -2.406931 -1.499143
16   databas -1.154334 -0.649261
17      imag -1.199246 -0.673638
```

Puisque que nous sommes à deux dimensions, une petite projection dans le plan fait l'affaire.

```
#représentation graphique des termes
import matplotlib.pyplot as plt

fig, axes = plt.subplots(figsize=(10,10))
axes.set_xlim(-3.5,+1.0)
axes.set_ylim(-1.5,+3.0)
for i in range(terms.shape[0]):
    plt.annotate(terms.Word[i],(terms.V1[i],terms.V2[i]),c='blue')
plt.show()
```



Des proximités semblent naturelles au regard du corpus que nous traitons. H2O propose un outil pour identifier le voisinage d'un point. Par exemple, pour ''store'', les 2 termes les plus proches seraient :

```
#chercher les voisins les plus proches
wv.find_synonyms("store",count=2)
```

```
OrderedDict([('databas', 0.9999983310699463), ('data', 0.9989187717437744)])
```

Pour ''data'', je comprends ; pour ''databas'', je suis moins convaincu par rapport à la projection dans le plan ci-dessus. En réalité, notre vision est faussée parce que le système utilise la mesure de similarité cosinus, très populaire en text mining, pour identifier le voisinage d'un terme.

Vérifions les calculs pour ''store'' et ''databas''. Nous copions les vecteurs de coordonnées dans une structure Numpy.

```
#coordonnées de store et databas
import numpy
coord1 = numpy.array(terms.loc[terms.Word=='store'].iloc[0,1:],dtype='float')
coord2 = numpy.array(terms.loc[terms.Word=='databas'].iloc[0,1:],dtype='float')
```

Puis nous calculons la similarité avec la formule du cosinus :

```
#similarité cosinus
sim = numpy.dot(coord1,coord2)/(numpy.linalg.norm(coord1)*numpy.linalg.norm(coord2))
print(sim)
```

```
0.9999983534177442
```

Nous retrouvons, aux erreurs de précision près, la similarité retournée par H20.

3.4 Coordonnées des documents

Pour les coordonnées des documents, H2O utilise le barycentre des termes qui les composent. Nous obtenons ainsi :

```
#description des documents sur la base des termes
description = wv.transform(tokenized,aggregate_method="AVERAGE")
desc_doc = description.as_data_frame()
print(desc_doc)
```

```
         C1        C2
0  0.179170  1.924784
1 -0.891070 -0.413727
2 -1.641898 -0.514405
3 -1.791004 -1.042043
4 -1.791004 -1.042043
```

Vérifions les calculs pour le document n°3, composé de 4 termes :

```
#vérification pour post n°3
doc = posts[3]
print(doc)
```

```
imag databas store data
```

Nous calculons à la main la coordonnée pour la 1ère dimension :

```
#à la main pour l'instant -- et c'est bien ça ! (1ere coordonnée)
v1          =          (1.0/4.0)*(terms.V1[terms.Word=="imag"].values[0]          +
terms.V1[terms.Word=="databas"].values[0]                                          +
terms.V1[terms.Word=="store"].values[0] + terms.V1[terms.Word=="data"].values[0])
print(v1)
```

```
-1.7910043895244598
```

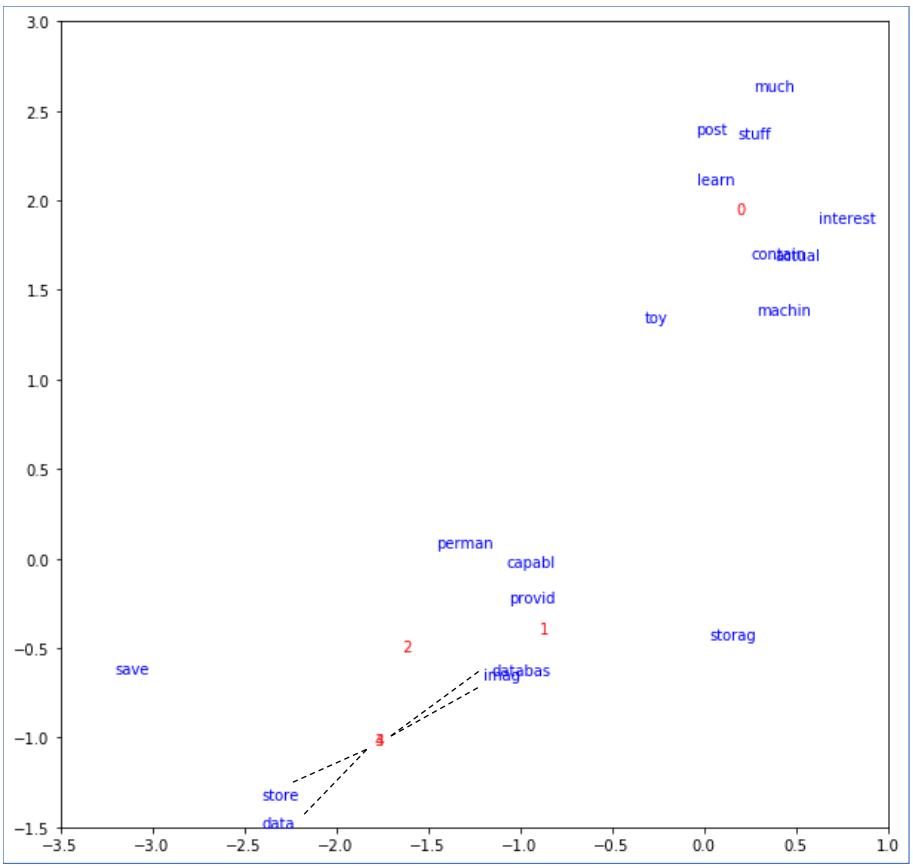
Et pour la seconde :

```
#seconde coordonnée
v2 = 0.0
for w in word_tokenize(doc):
    v2 = v2 + terms.V2[terms.Word==w].values[0]
v2 = v2 / 4.0
print(v2)
```

```
-1.0420429855585098
```

Cette relation barycentrique entre les termes et les documents permet de les représenter dans le même repère.

```
#représentation graphique simultanée avec les termes
fig, axes = plt.subplots(figsize=(10,10))
axes.set_xlim(-3.5,+1.0)
axes.set_ylim(-1.5,+3.0)
#les termes
for i in range(terms.shape[0]):
    plt.annotate(terms.Word[i],(terms.V1[i],terms.V2[i]),c='blue')
#les documents
for i in range(desc_doc.shape[0]):
    plt.annotate(str(i),(desc_doc.C1[i],desc_doc.C2[i]),c='red')
plt.show()
```



Quelques commentaires :

- Les documents n°3 et n°4 sont confondus puisqu'ils sont composés des mêmes termes, lesquels sont dupliqués 3 fois pour le n°4 (la moyenne n'est pas altérée).
- Le document n°3 est bien ''au centre'' des termes (''imag'', ''databas'', ''store'', ''data'').

4 Conclusion

Dans ce tutoriel, nous nous sommes placés dans un cadre très simple où il est possible de représenter les termes et les documents dans le plan (vec_size = 2).

En pratique, sur les corpus réalistes, on choisit une plus grande dimensionalité pour pouvoir appréhender toute la complexité des corpus (volumétrie, richesse du dictionnaire…). Il est dès lors possible d'appliquer les algorithmes de machine learning (clustering, classement, etc.) dans ce nouvel espace de description.

Pour ce qui est des représentations graphiques dans ce contexte (dimensionalité > 2), nous pouvons calculer les distances par paires des termes (et des documents), et utiliser des techniques de projection telles que le positionnement multidimensionnel (MDS – multidimensional scaling).

5 Références

Coelho, L. P., Richert, W., ''Building Machine Learning Systems with Python'', Packt Publishing, 2015.

H2O, ''Docs – Algorithms – Word2Vec'', http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/word2vec.html

NLTK, ''Natural Language Toolkit'', https://www.nltk.org/

Tutoriel Tanagra, ''Deep learning : l'algorithme Word2Vec'', décembre 2019.