

Topic Modeling

NLP



Le topic modeling, c'est quoi ?

Un topic model fait référence à un modèle probabiliste, définissant l'appartenance de documents à des thématiques précises : on identifie le ou les sujets d'un texte.

Par exemple, des livres sur Python ont une forte probabilité de contenir des termes en rapport avec le code et la data science, comme « statistiques », « package », « fonctions », etc à l'inverse d'un livre de Spinoza 🐱.



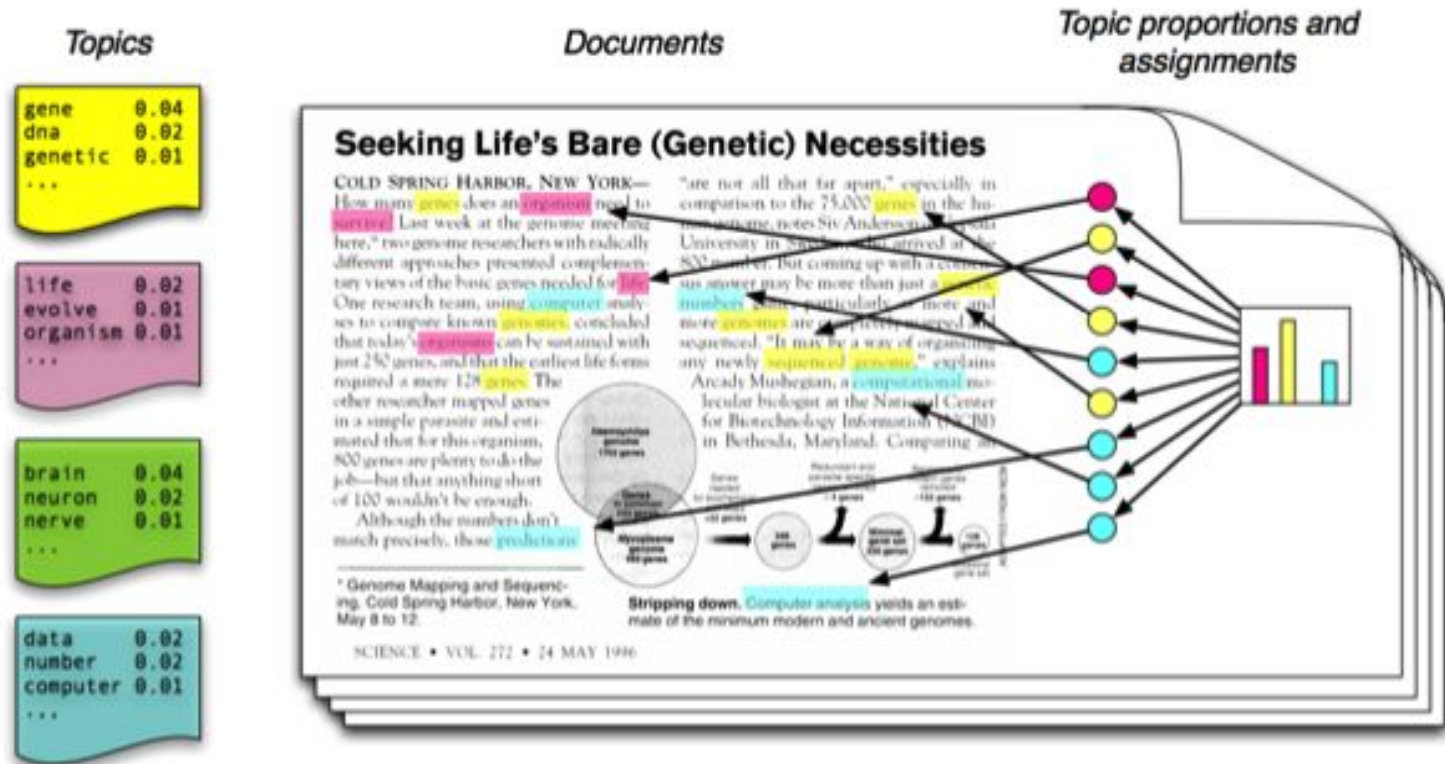
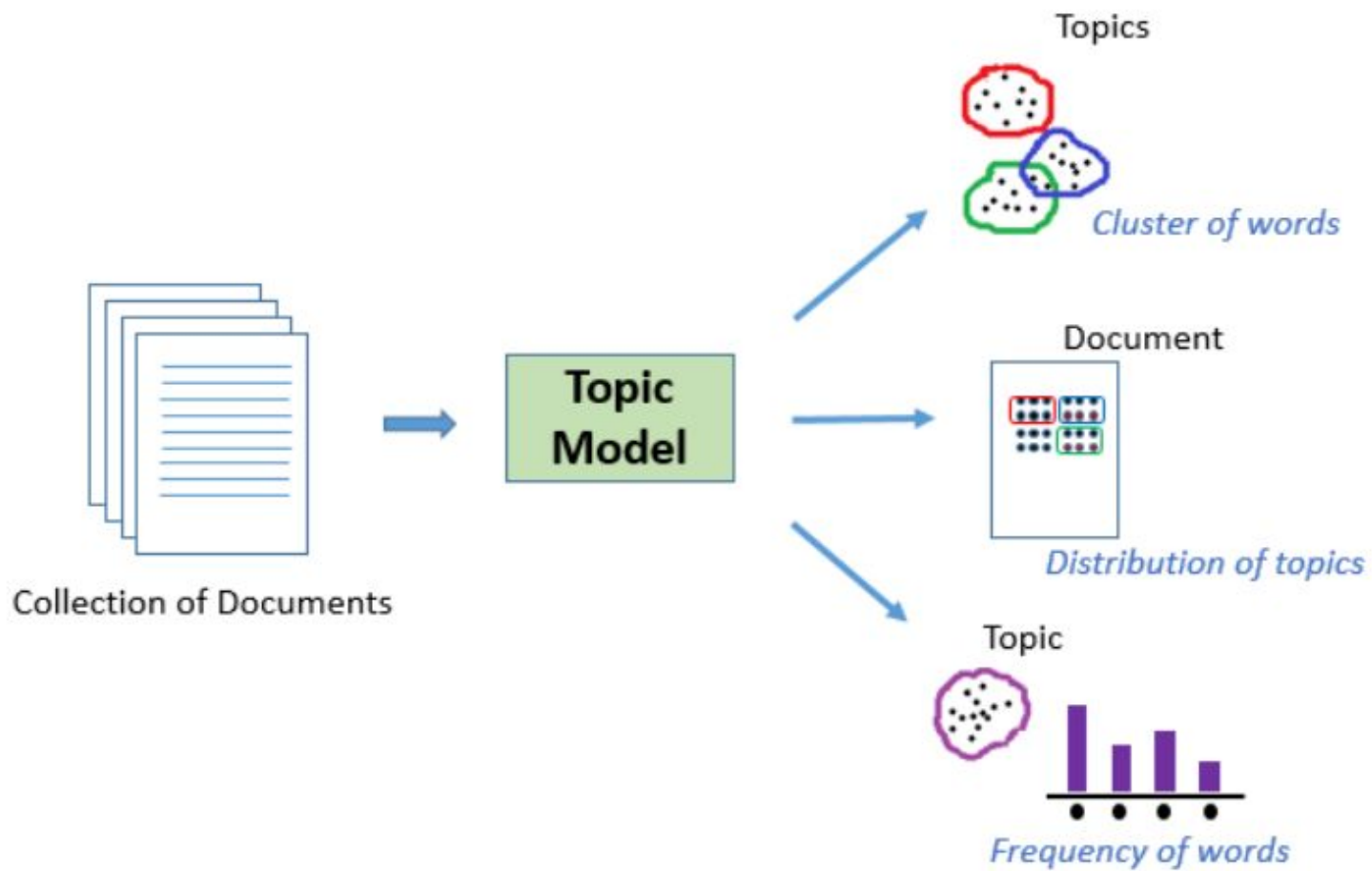


Figure source: Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, 55(4), 77-84.



Comment ça marche ?

01

Data

Récupération par
scraping ou dans
des datasets (avec
ou sans aide de Regex)

02

Traitement & Analyse

Tokenisations, lemmatisations,
stopwords, TD-IDF, n-gramms, bag
of words, corpus, ponctuations,
espace vides extra

03

04

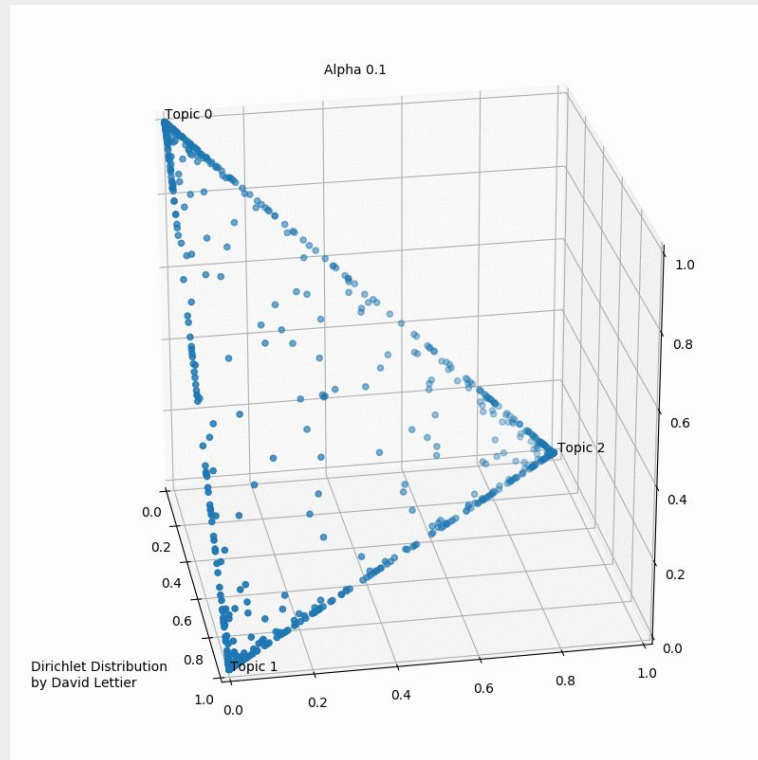
Techniques

LDA
Word embedding

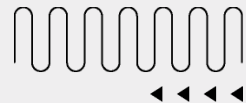
LDA

Chaque document peut être décrit comme une distribution de sujets et chaque sujet peut être décrit par une distribution de mots.

- Le nombre de mots dans le document est déterminé.
- Un mélange de sujets pour le document sur un ensemble fixe de sujets est choisi.
- Un sujet est sélectionné sur la base de la distribution multinomiale du document.
- Maintenant, un mot est choisi sur la base de la distribution multinomiale du sujet.



Représentation de la distribution des certain sujets dans un document.



LDA

LDA peut également être utilisé à l'envers :

- D'abord, chaque mot de chaque document est assigné aléatoirement à l'un des sujets.
- Maintenant, on suppose que toutes les affectations de sujets, à l'exception de celle-ci, sont correctes.
- La proportion de mots dans le document 'd' qui sont actuellement assignés au sujet 't' est égale à $p(\text{sujet } t \mid \text{document } d)$ et la proportion d'assignations du sujet 't' sur tous les documents qui appartiennent au mot 'w' est égale à $p(\text{mot } w \mid \text{sujet } t)$.
- Ces deux proportions sont multipliées et un nouveau sujet est attribué en fonction de cette probabilité.

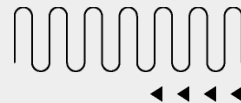
LDA suppose que les mots de chaque document sont liés.

Puis, après avoir suivi les étapes susmentionnées, il détermine comment un certain sujet a pu être créé.

Et c'est cette même solution qui sera utilisée pour générer des distributions de sujets et de mots sur un corpus.



CountVectorizer



From SKLearn

Création du matrice qui
va tokeniser les mots
présent dans le Corpus

```
In [4]: 1 corpus = [  
2         'my name is cedric, and my age is 24,'  
3         'the fun fact is that there is many ways to prove this',  
4         'i will try to ... to ... to ... i dont know,'  
5         'but i will prove it',  
6     ]
```

```
In [6]: 1 from sklearn.feature_extraction.text import CountVectorizer  
2     vectorizer = CountVectorizer()
```

```
In [14]: 1 X = vectorizer.fit_transform(corpus)
```

```
In [16]: 1 vectorizer.get_feature_names()
```

```
Out[16]: ['24',  
         'age',  
         'and',  
         'but',  
         'cedric',  
         'dont',  
         'fact',  
         'fun',  
         'is',  
         'it',  
         'know',  
         'many',  
         'my',  
         'name',  
         'proove',  
         'that',  
         'the',  
         'there',  
         'this',  
         'to',  
         'try',  
         'ways',  
         'will']
```

```
In [11]: 1 print(X.toarray())
```

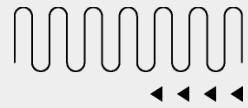
```
[[1 1 1 0 1 0 1 1 4 0 0 1 2 1 1 1 1 1 1 0 1 0]  
 [0 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 3 1 0 2]]
```

```
In [17]: 1 X
```

```
Out[17]: <2x23 sparse matrix of type '<class 'numpy.int64'>'  
         with 25 stored elements in Compressed Sparse Row format>
```



OneVsRest Classifier (OVR)



From SKLearn

Le principe est de fit un classifieur par classe
Pour chaque classifieur, la classe est comparée à toutes les autres classes.

Ce combine a different model de ML

```
1 import numpy as np
2 from sklearn.multiclass import OneVsRestClassifier
3 from sklearn.svm import SVC
```

```
1 X = np.array([
2     [10, 10],
3     [8, 10],
4     [-5, 5.5],
5     [-5.4, 5.5],
6     [-20, -20],
7     [-15, -20]
8 ])
```

```
1 y
array([0, 0, 1, 1, 2, 2])
```

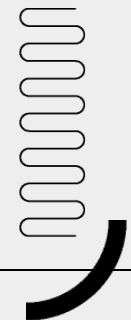
```
1 y = np.array([0, 0, 1, 1, 2, 2])
```

```
1 clf = OneVsRestClassifier(SVC()).fit(X, y)
```

```
1 clf.predict([[-19, -20], [9, 9], [-5, 5]])
array([2, 0, 1])
```

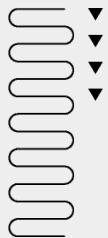
Dans notre cas, combiné avec un LR

```
1 clf = OneVsRestClassifier(LogisticRegression(C = 10, n_jobs=-1))
2 _ = clf.fit(trn2, trn[TARGET_COLS])
3
4 val_preds = clf.predict_proba(val2)
```



Notre modèle

LDA
(non supervisé)



Le dataset

Notre dataset contient des articles de recherche scientifique.

Nous devons utiliser l'ensemble de données pour prédire le sujet d'un article à l'aide de son résumé (colonne ABSTRACT).

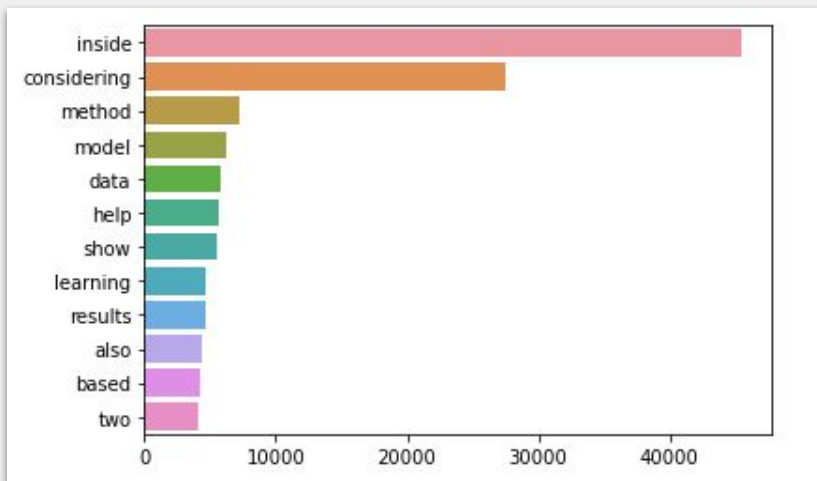
train.csv

	id	ABSTRACT	Computer Science	Mathematics	Physics	Statistics	Analysis of PDEs	Applications	Artificial Intelligence
0	1824	a ever-growing datasets inside observational a...	0	0	1	0	0	0	0
1	3094	we propose the framework considering optimal \$...	1	0	0	0	0	0	0
2	8463	nanostructures with open shell transition meta...	0	0	1	0	0	0	0

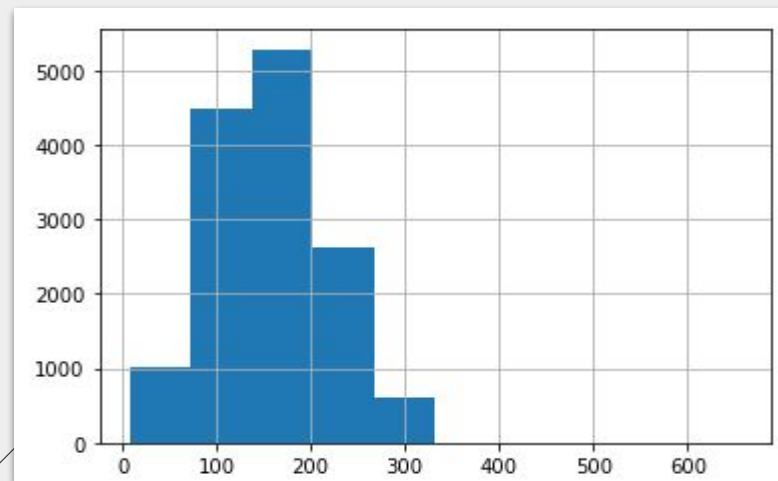


Les étapes 1. Analyse

- **Exploration**, analyse et visualisation



Nombre de mots les plus fréquents
hors stopwords



Nombre de mots par résumé
(colonne ABSTRACT)



Les étapes 2. Preprocessing



Tokenisation, lemmatisation et suppression des stopwords

```
# tokenize / remove stopwords / lemmatize

def preprocess_news(df):
    corpus=[]
    stem=PorterStemmer()
    lem=WordNetLemmatizer()
    for news in df_train['ABSTRACT']:
        words=[w for w in word_tokenize(news) if (w not in stop)]

        words=[lem.lemmatize(w) for w in words if len(w)>2]

        corpus.append(words)
    return corpus

corpus=preprocess_news(df_train)
```

Conversion du corpus en dictionnaire puis bag of words

```
# bag of words" avec gensim

dic=gensim.corpora.Dictionary(corpus)
bow_corpus = [dic.doc2bow(doc) for doc in corpus]
```

Preprocessing avec les librairies

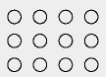
NLTK et Gensim > Identification des stopwords, création du corpus, tokenisation, lemmatisation et suppression des stopwords, conversion du corpus en dictionnaire (mappage des mots avec leur ID) puis bag of words (nombre de fois où le mot apparaît).





Les étapes 3. Le modèle LDA/Multicore

Le modèle LDAMulticore utilise tous les cœurs de processeur pour accélérer la formation du modèle.



```
lda_model = gensim.models.LdaMulticore(bow_corpus,  
                                         num_topics = 25,  
                                         id2word = dic,  
                                         passes = 10,  
                                         workers = 2)
```

bow_corpus : notre corpus transformé en bag of words

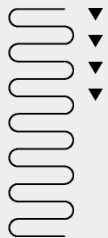
num_topics = 25 : nombre de sujets que l'on souhaite trouver

id2word = dic : déterminer la taille du vocabulaire, ainsi que pour le débogage

passes = 10 : nombre de passages dans le corpus pendant l'entraînement.

workers = 2 : propre au LDAMulticore pour optimiser le temps/performance





Métrique

Afin de mesurer et analyser notre modèle, nous avons utilisé un métrique :

Coherence score / Topic coherence : Le score de cohérence thématique évaluent le degré de similitude sémantique entre les mots les mieux 'notés' dans le sujet.

```
coherence_model_lda = CoherenceModel(model=lda_model, texts=corpus, dictionary=dic, coherence='c_v')
```



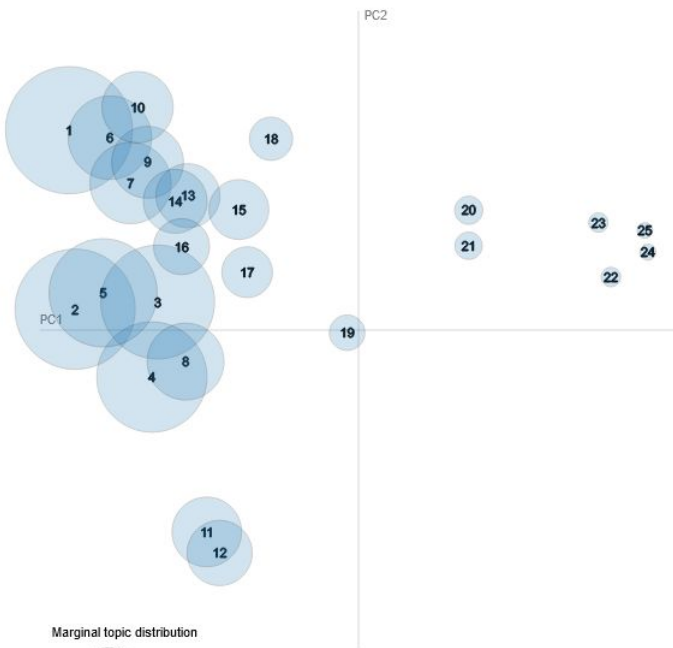
Coherence score de 0.47 avec notre modèle non supervisé avec LDA



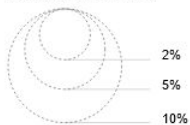
Visualisation des termes fréquents

```
pyLDAvis.enable_notebook()
vis = gensimvis.prepare(lda_model, bow_corpus, dic)
vis
```

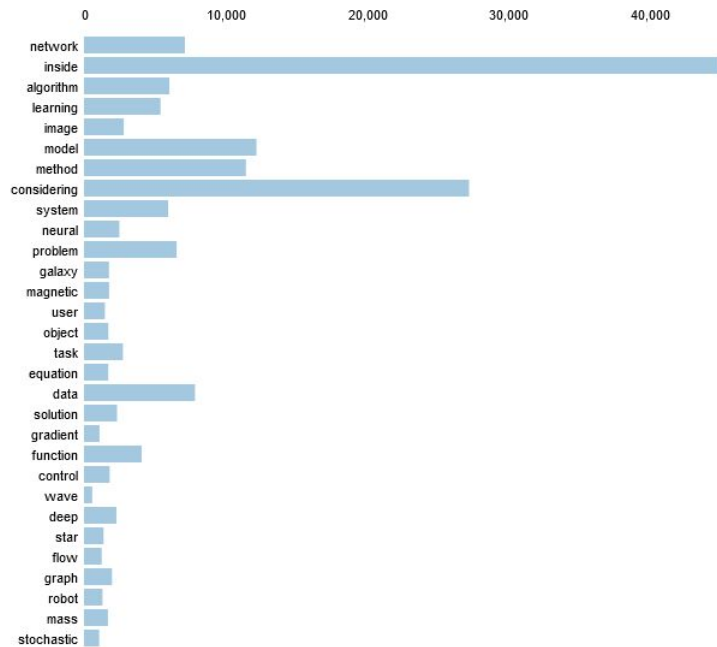
Intertopic Distance Map (via multidimensional scaling)



Marginal topic distribution



Top-30 Most Salient Terms¹



Overall term frequency

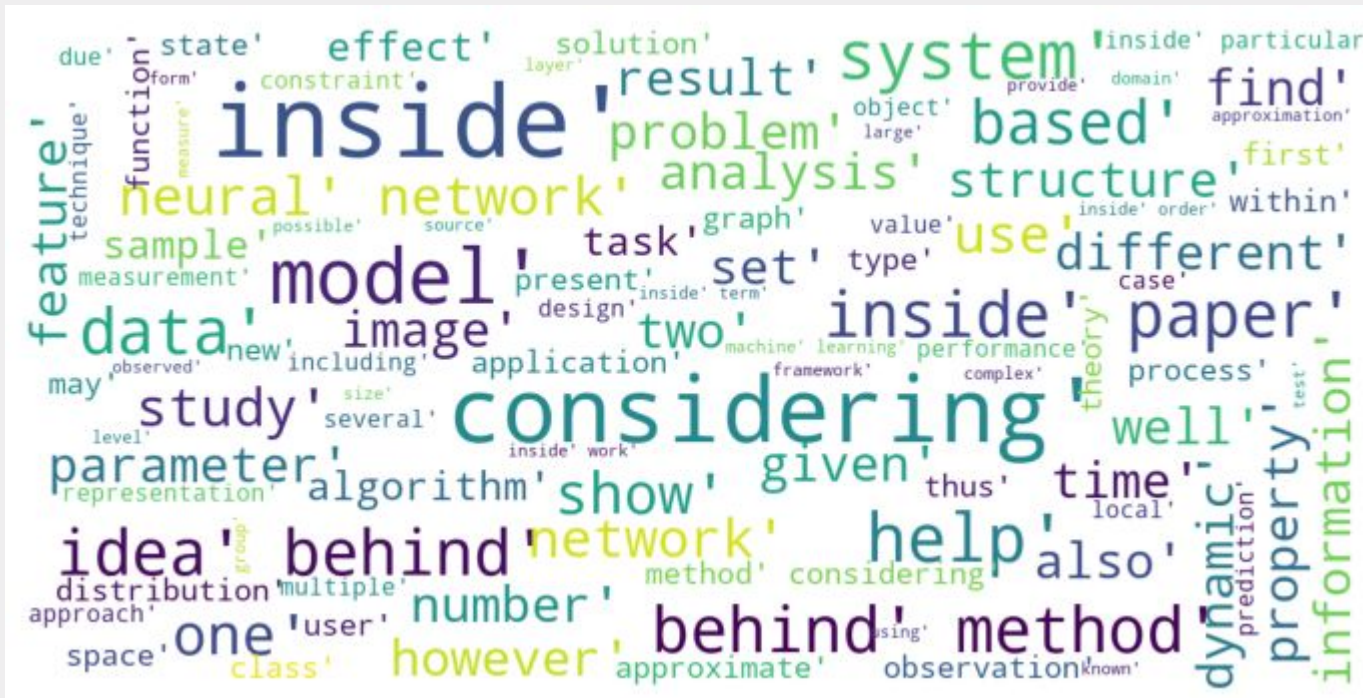
Estimated term frequency within the selected topic

¹ $salency(term, w) = frequency(w) * (\sum_{l \in topics} \log(p(l|w)/p(l)))$, see Chuang et al. (2012)

² $relevance(term, w, l, topic) = \lambda * p(w, l) + (1 - \lambda) * p(w, l) / p(w)$, see Sievert & Shirley (2014)

Visualisation des termes fréquents

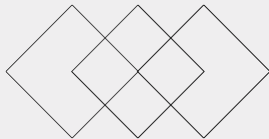
Wordcloud





Amélioration possible de notre modèle

Nous vous proposons de tester d'autres modèles grâce à des techniques et/ou algorithmes comme LDA2vec en deep learning.





LDA2vec

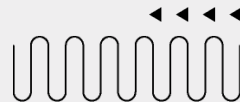
Algorithme “hybride” de deep learning créé, en 2016, par Christopher Moody. Il associe ainsi les algorithmes de LDA et word2vec, dans le but d’optimiser le topic modeling.

[Slideshow de Christopher Moody](#)

[Publication de Christopher Moody](#)

Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec

[Github de LDA2vec](#)





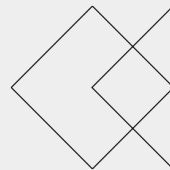
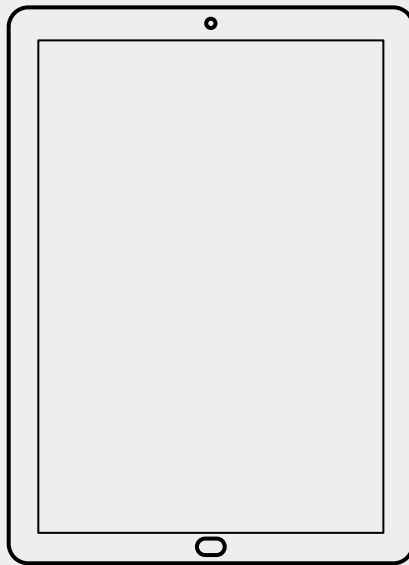
Liens

[LDA avec Gensim](#)

[Modèle LSTM et Bert avec Pytorch](#)

[Word Embedding avec Gensim](#)

[Liste de 25 algorithmes de Word embedding](#)





MERCI !

