Catégorisez automatiquement des questions

Soutenance du projet n°5 : Parcours « Ingénieur Machine Learning »

Luke Duthoit

Plan de la présentation :

- Présentation de la problématique et du prétraitement effectué, *cleaning*, *feature engineering* et exploration
- Présentation de l’approche non supervisée
- Présentation de l’approche supervisée et du modèle final sélectionné
- Démonstration de l’API et conclusion

La problématique

- *Dataset* : Données textuelles issues de *Stackoverflow.*

- Paramètres : Des questions d'utilisateurs (titre + corps) ainsi que 5 *tags* déjà suggérés par *Stackoverflow.*

- Objectif : développer notre propre algorithme de suggestion de *tags*.

Le pré-traitement effectué

- Données extraites avec la requête SQL suggérée par OC :

```sql
SELECT TOP 500000 Title, Body, Tags, Id, Score, ViewCount, FavoriteCount, AnswerCount
FROM Posts
WHERE PostTypeId = 1 AND ViewCount > 10 AND FavoriteCount > 10
AND Score > 5 AND AnswerCount > 0 AND LEN(Tags) - LEN(REPLACE(Tags, '<','')) >= 5
```

- *Title*, *Body* et *Tags* (données textuelles) servent à construire le jeu de données.

- Les autres colonnes (*features* numériques) ne servent qu'à la sélection de certains documents.

- Rigoureusement 5 mots-clefs (*tags*) par question (cf Notebook).

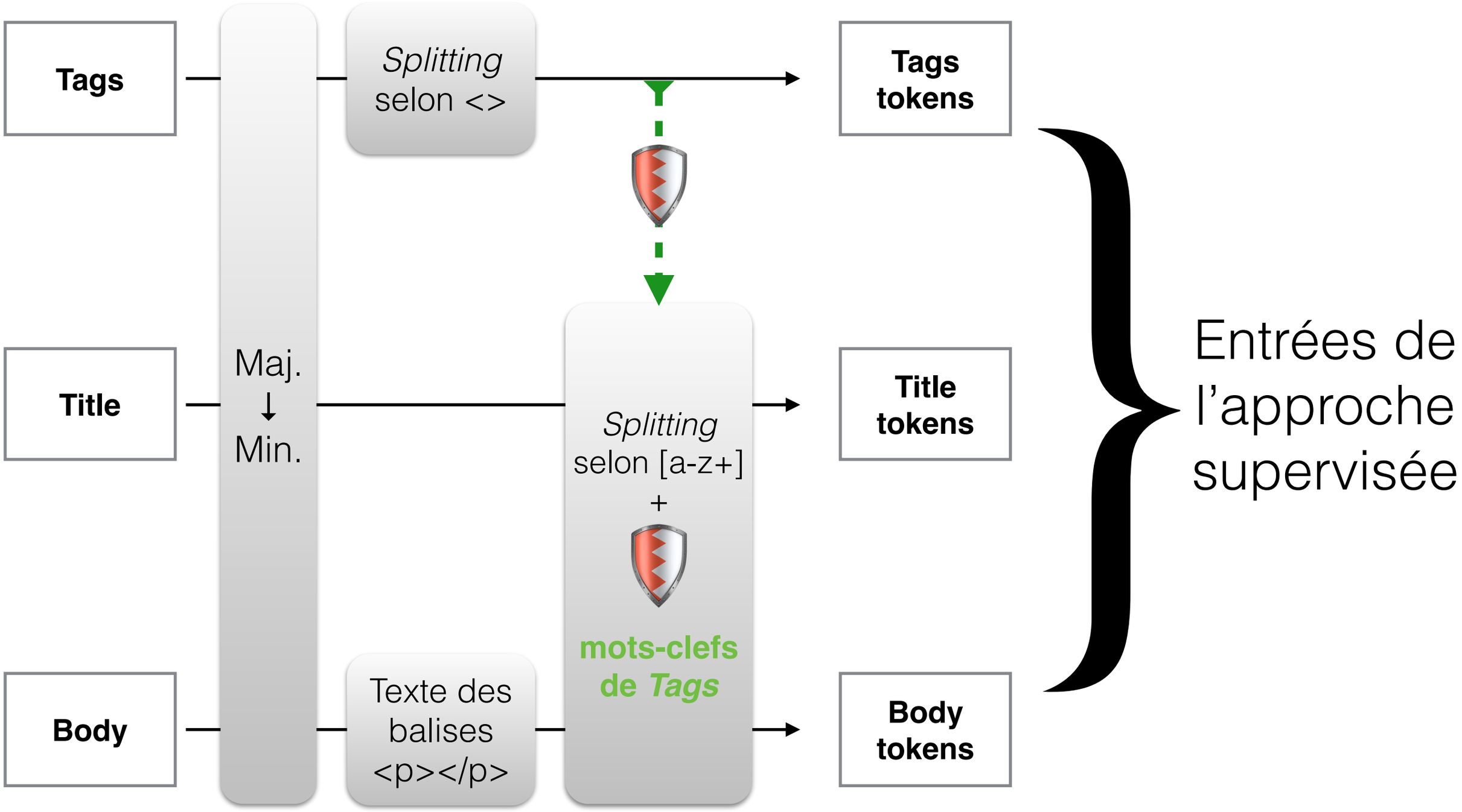
Réduction dimensionnelle supplémentaire sur la base du score

- Corpus initial > 27000 documents
  - ✦ Trop lourd à traiter
  - ✦ Taille à diviser par 2

	Score	ViewCount	FavoriteCount	AnswerCount
Score	1.000000	0.680333	0.891282	0.396596
ViewCount	0.680333	1.000000	0.510110	0.478100
FavoriteCount	0.891282	0.510110	1.000000	0.315248
AnswerCount	0.396596	0.478100	0.315248	1.000000

- La sélection doit être pertinente :
  - ✦ Basée sur les meilleurs scores (paramètre numérique le mieux corrélé aux autres)

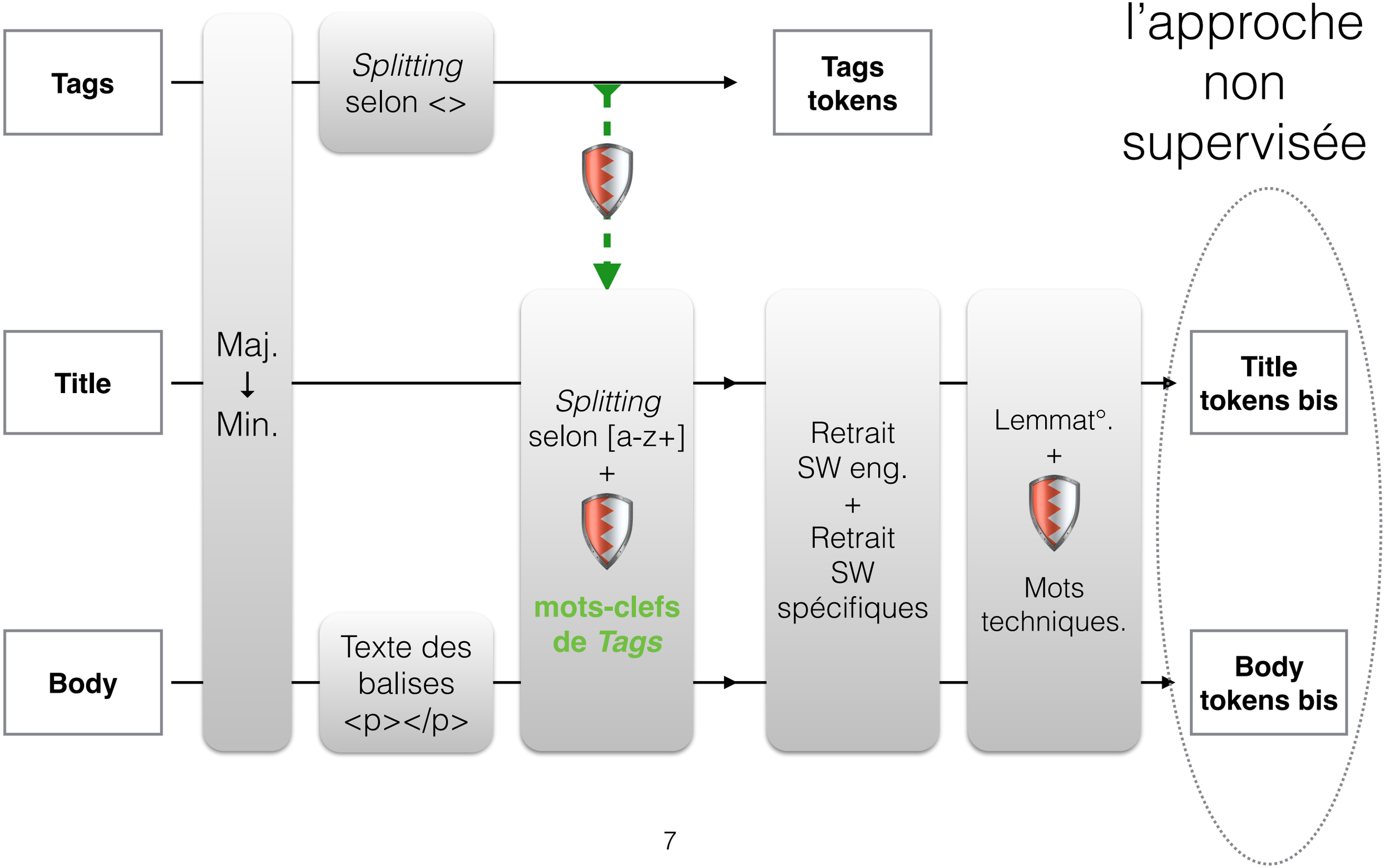
Le cleaning


Tags
Title
Body
Maj.
↓
Min.
Splitting
selon <>
Texte des
balises

Splitting
selon [a-z+]
+
mots-clefs
de Tags
Tags
tokens
Title
tokens
Body
tokens
Entrées de
l'approche
supervisée

Le cleaning

Entrées de l'approche non supervisée

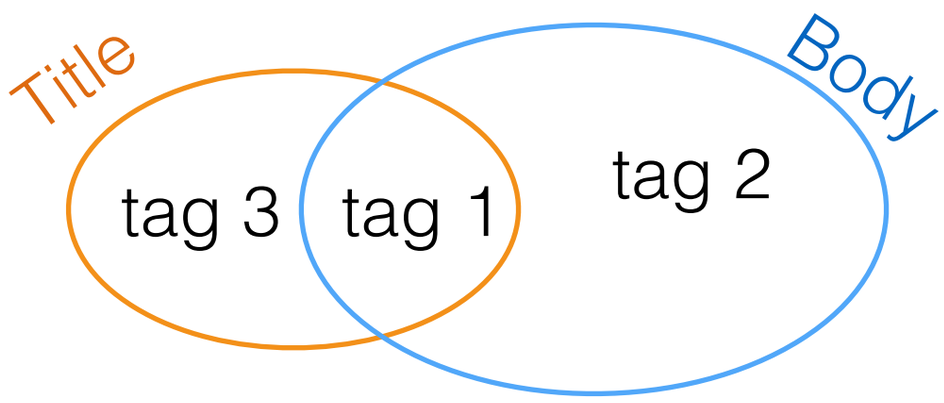


Les corpus : *Title* + corpus alternatif

Au final, on a travaillé sur deux corpus de documents :

- *Title* (vocabulaire - riche, - de tokens $\Longrightarrow$ + rapide à traiter) ;
- Un corpus « alternatif », faits de documents de *Title* et/ou *Body* .

💡 Pour chaque document, le + petit ensemble qui contient le + de mots-clefs :



Presents dans title	6516
Presents dans body	5691
Presents dans les deux	1351

Feature engineering

Des *features* adaptées aux approches imposées :

- Approche non supervisée :
  - *Bag of words* pondéré (TF-IDF).
- Approche supervisée : réduction de dimension avec plongement de mots
  - Word2Vec ;
  - BERT (de type *Hugging-Face*) ;
  - USE.

Plan de la présentation :

- Présentation de la problématique et du prétraitement effectué, cleaning, feature engineering et exploration
- Présentation de l’approche non supervisée
- Présentation de l’approche supervisée et du modèle final sélectionné
- Démonstration de l’API et conclusion

Utilisation de la métrique : score de cohérence

- Librairie python *gensim* [1] utilisée dans le notebook pour générer les BOW et faire de la LDA.

  → nombreux hyper-paramètres ;

  → optimisation nécessaire.

- Utilisation le score de « cohérence » de type *c_v* [2] :
  - pas la + simple à comprendre, mais mise en oeuvre rapide.
  - confirme supériorité de TF-IDF sur TF.

- Optimisation sur :
  - nombre de sujets à inférer ;
  - probabilité initiale de la distribution de thèmes par documents (**α**) ;
  - probabilité initiale de la distribution de mots par thèmes (**η**).

- On associe à chaque document le thème le plus probable.

[1] https://radimrehurek.com/gensim/models/ldamodel.html
[2] https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0

Succès TRES mitigé sur le corpus de *Title*

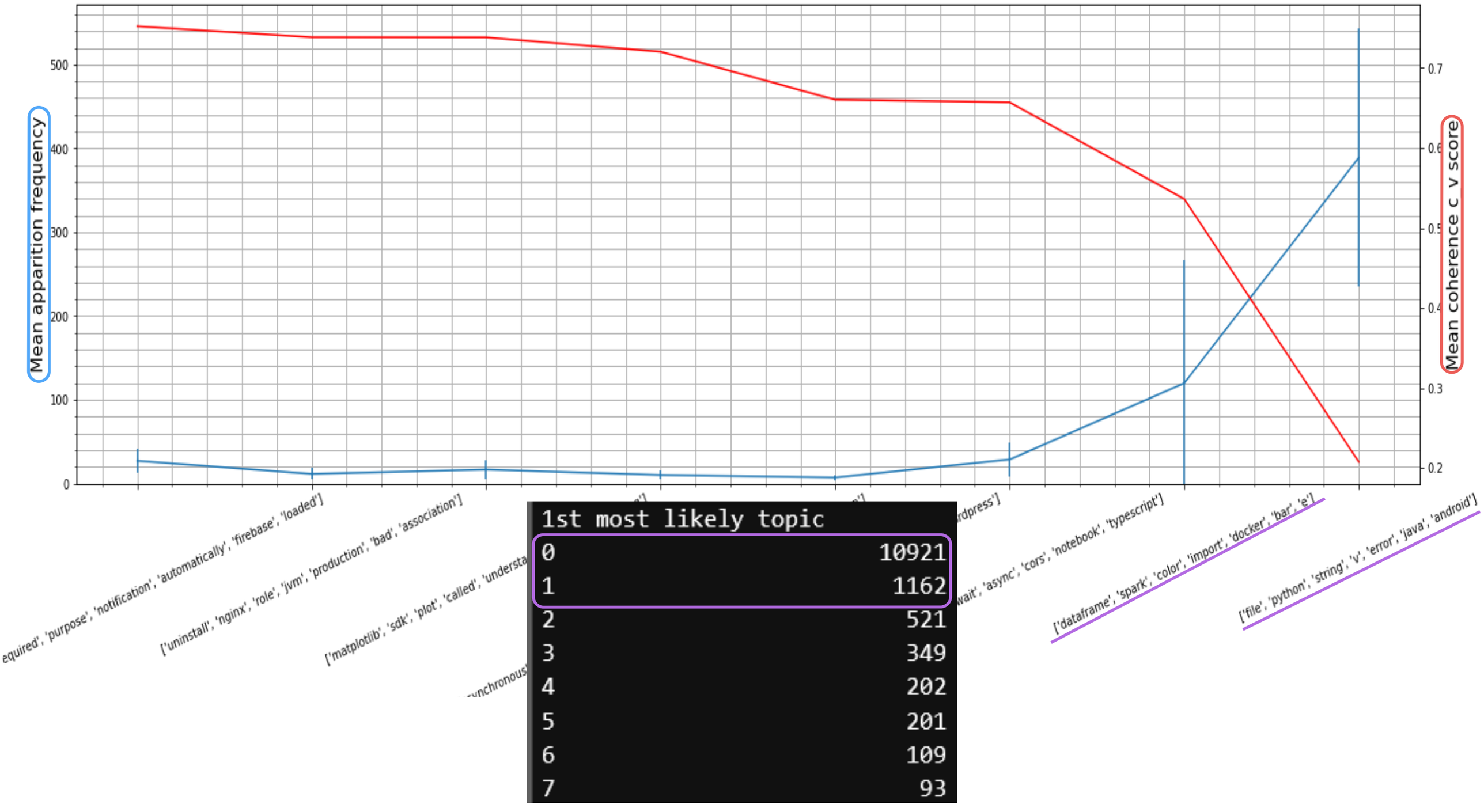
Appliquée à *Title* :

- quasiment tous les documents dans le même thème (mots + fréquents) ;
- difficile d'interpréter les thèmes inférés (seuls quelques mots/thème semblent concorder)

1st most likely topic	
0	10921
1	1162
2	521
3	349
4	202
5	201
6	109
7	93

	Topic n°0	Topic n°1	Topic n°2	Topic n°3	Topic n°4	Topic n°5	Topic n°6	Topic n°7
Top word n°0	file	dataframe	delete	react	uninstall	matplotlib	encrypt	asynchronous
Top word n°1	python	spark	required	webpack	nginx	sdk	turning	dataframes
Top word n°2	string	color	purpose	await	role	plot	decrypt	concatenation
Top word n°3	v	import	notification	async	jvm	called	diagram	restful
Top word n°4	error	docker	automatically	cors	production	understanding	merging	iterator
Top word n°5	java	bar	firebase	notebook	bad	sublime	cast	effect
Top word n°6	android	e	loaded	typescript	association	trying	wordpress	explanation
Top word n°7	function	scala	virtualenv	jupyter	aggregation	seaborn	life	versus
Top word n°8	value	fragment	fit	exactly	deleted	anaconda	uml	volume
Top word n°9	object	container	bundle	babel	legacy	eslint	imagemagick	modern

Cohérence VS Fréquence ?



1st most likely topic	
0	10921
1	1162
2	521
3	349
4	202
5	201
6	109
7	93

Amélioration des résultats en ne conservant que les mots les + fréquents (1)

Appliquée à *Title* **- {mots de fréquence < 50}** :

- 2 x + de thèmes, mais bien meilleure répartition documents/thème ;
- thèmes + facilement interprétables.

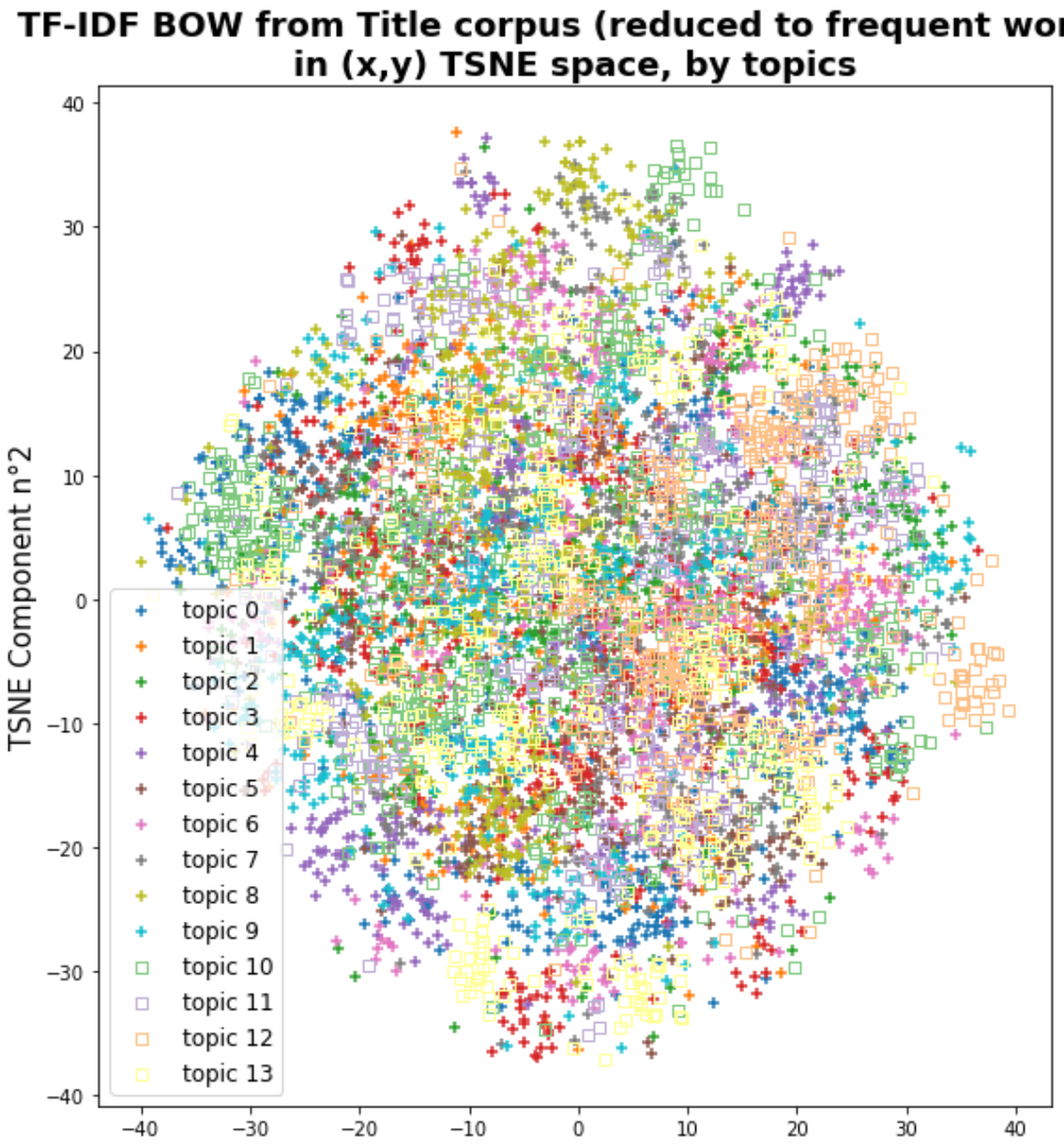
1st most likely topic	
10	1119
0	1046
6	1031
2	945
9	930
8	913
13	901
7	894
3	883
12	862
11	859
1	808
5	793
4	763

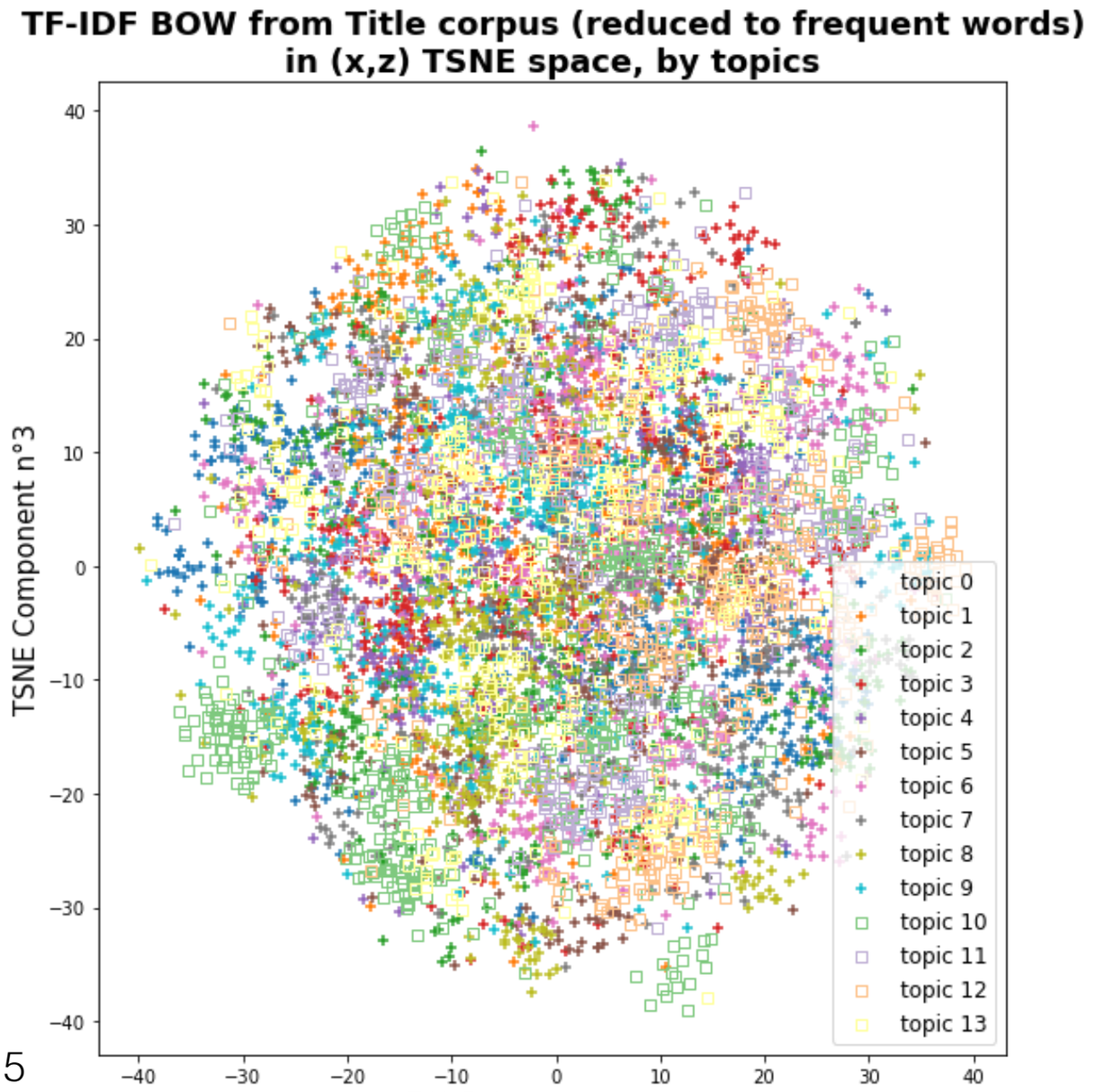
	Topic n°0	Topic n°1	Topic n°2	Topic n°3	Topic n°4	Topic n°5	Topic n°6	Topic n°7	Topic n°8	Topic n°9	Topic n°10	Topic n°11	Topic n°12	Topic n°13
n°0	difference	type	function	object	java	data	v	python	file	error	list	array	c	io
n°1	value	javascript	window	app	method	make	set	android	test	spring	studio	image	text	project
n°2	panda	swift	add	code	create	name	return	string	line	api	git	view	convert	parameter
n°3	column	multiple	json	core	http	web	run	read	command	failed	install	variable	server	date
n°4	change	class	find	xcode	angular	application	google	r	element	access	visual	time	key	page
n°5	dataframe	request	cannot	build	without	check	task	dependency	asp.net	version	work	database	sql	chrome
n°6	color	found	std	module	framework	service	query	mysql	mvc	number	php	numpy	performance	linux
n°7	bootstrap	header	laravel	memory	load	static	c++	like	html	property	update	loop	component	remove
n°8	row	button	table	django	mean	jpa	url	format	j	working	package	call	operator	input
n°9	getting	library	running	support	directory	docker	bash	folder	node	jquery	rail	form	token	c#

Amélioration des résultats en ne conservant que les mots les + fréquents (2)

Si on représente les documents par thèmes dans un espace réduit (par TSNE) :

- (+) régulièrement des blocs compacts de documents de même thème ;
- (-) mais pas de « grandes » frontières de décisions évidentes entre thèmes.





Conclusions de l'approche non supervisée

- Résultats reproduits avec le corpus « alternatif ».

- Nécessité d'exclure les mots rares pour pouvoir interpréter les thèmes inférés.

- « Cohérence VS Fréquence » met en question la pertinence de cette métrique.

- De fait, questionne aussi le nombre de thème « optimal » qui en découle :
  - nombre faible, pratique pour l'esprit humain ;
  - mais << au nombre de mots-clefs…

Plan de la présentation :

- Présentation de la problématique et du prétraitement effectué, cleaning, feature engineering et exploration
- Présentation de l'approche non supervisée
- Présentation de l'approche supervisée et du modèle final sélectionné
- Démonstration de l'API et conclusion

Modus operandi

- Inspiration du *notebook* de classification de tweets (ressource OC) [1].
- Par manque de temps, seul le corpus issu de *Title* a été testé.
- La division *train*/*test set* est réalisée dans les promotions 70%/30% (pour gagner du temps de calcul) :
  - X = les matrices issues du plongement de mots, de dimension = (taille du jeu, dimension de l'espace de plongement)
  - Y = matrice de mots-clefs, de dimension = (taille du jeu, 5)
- Pour les 3 types de *features*, même procédure :
  - optimisation d'UN hyper-paramètre ;
  - 5 tags par documents → classification avec *MultiOutputClassifier* ;
  - 3-4 estimateurs de classification « connue » par *MultiOutputClassifier.*

[1] https://s3.eu-west-1.amazonaws.com/course.oc-static.com/projects/Data_Scientist_P6/Exemple_Tweets_Feature-extraction_Sentence+Embedding_V1.1.ipynb

Fabrication d'une métrique pour évaluer la performance : « intersection score »

- Le score de précision risque d'être insuffisant du fait des 5 prédictions indépendantes du *MultiOutputClassifier* :

	Tag 1	Tag 2	Tag 3	Tag 4	Tag 5
True	python	python-3.x	java	open	oriented
Pred	python	androïd	python-3.x	file	platform
Correct	☑	☒	☒	☒	☒
Accuracy	0.2				

- Notre métrique calcule l'intersection des prédictions et des *targets* pour compenser une prévision à la mauvaise position :

	Tag 1	Tag 2	Tag 3	Tag 4	Tag 5
True	python	python-3.x	java	open	oriented
Pred	python	androïd	python-3.x	file	platform
Correct	☑	☒	☑	☒	☒
IS	0.4				

Les différences entre les modèles d'extraction de *features*

- Quand on a utilisé Word2Vec :
  - ✦ optimisation de la taille de fenêtre ;

  (→ peu/pas d'impact sur le score sur jeu d'entraînement/de test)
  - ✦ on a pris 300 vecteurs dans l'espace de plongement (∼3% jeu d'entraînement) ;
  - ✦ les mots constitués d'une seule lettre sont pris en compte.

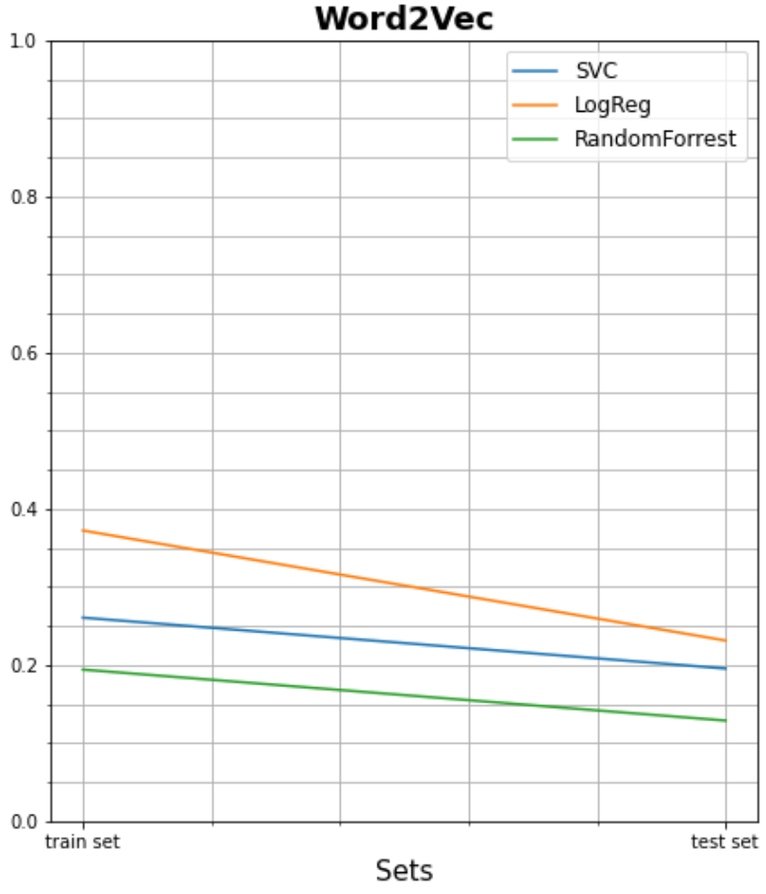
- Quand on a utilisé BERT ou USE :
  - ✦ optimisation de la taille du batch ;

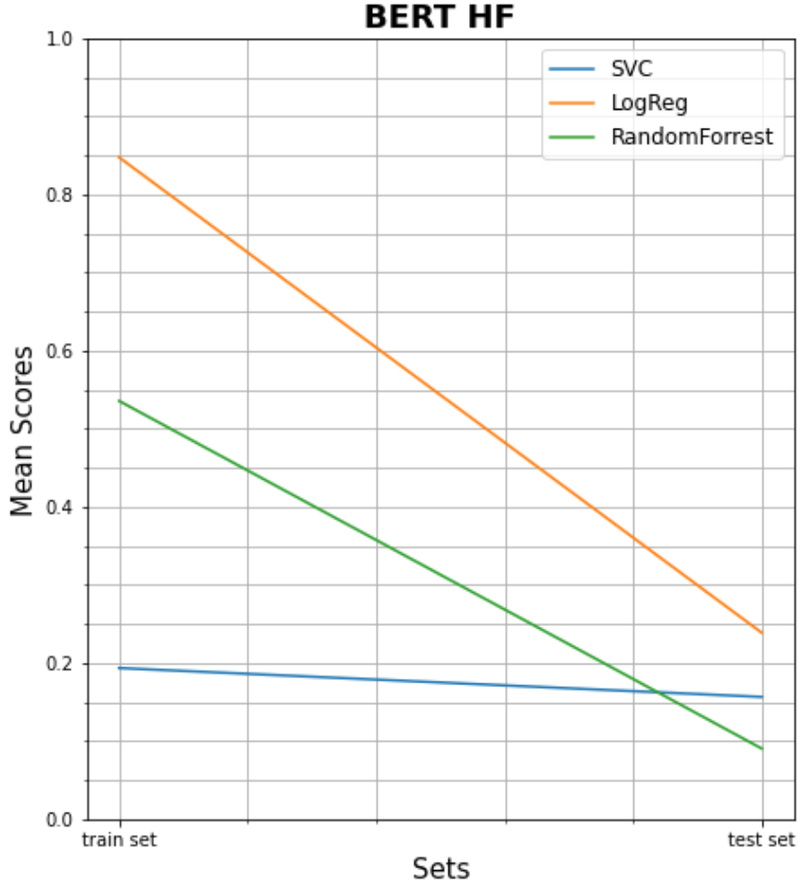
  (→ pas d'impact sur les score sur les deux jeux)
  - ✦ dimension l'espace de plongement calculée par le modèle :
    - ∘ ∼750 vecteurs de base avec BERT ;
    - ∘ ∼500 vecteurs de base avec USE.

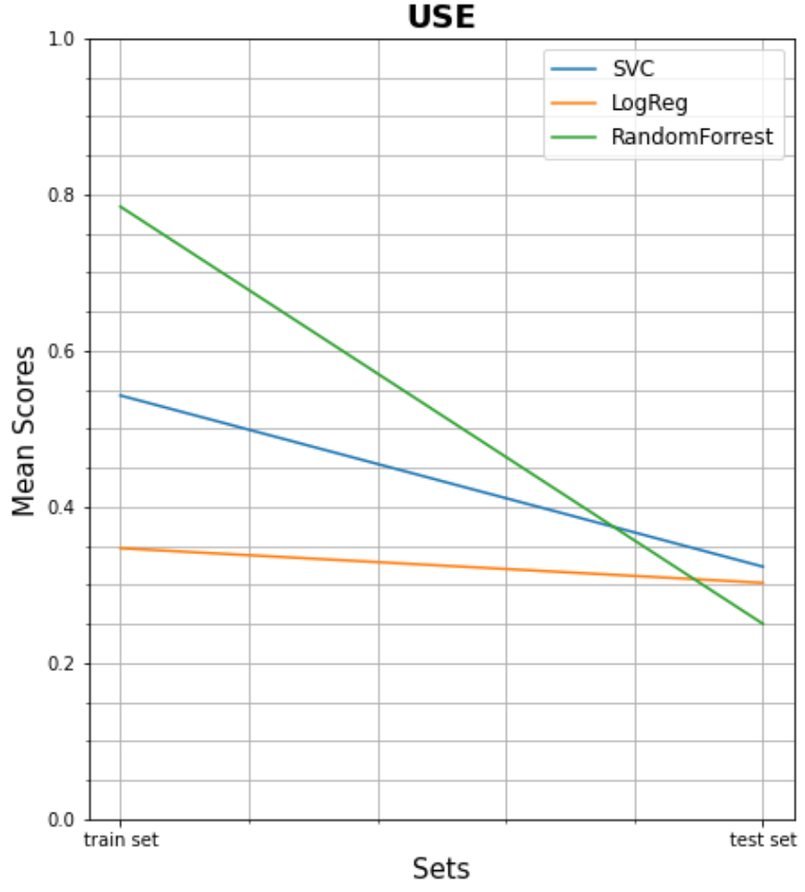
[1] https://s3.eu-west-1.amazonaws.com/course.oc-static.com/projects/Data_Scientist_P6/Exemple_Tweets_Feature-extraction_Sentence+Embedding_V1.1.ipynb

Comparaison des modèles

- Sur-apprentissage régulier.
- Scores sur jeu de test assez faibles [*baseline* sur matrice TF-IDF après réduction de dimension → ∼0.12 sur jeu d'entraînement/test]
- On choisit USE + régression logistique (2nd meilleur score, et surtout calculs + rapide qu'avec SVC).







Modèle final sélectionné

On préfère le modèle issu de USE + réversion logistique, plutôt que la LDA appliquée sur Title réduit aux mots très fréquents :

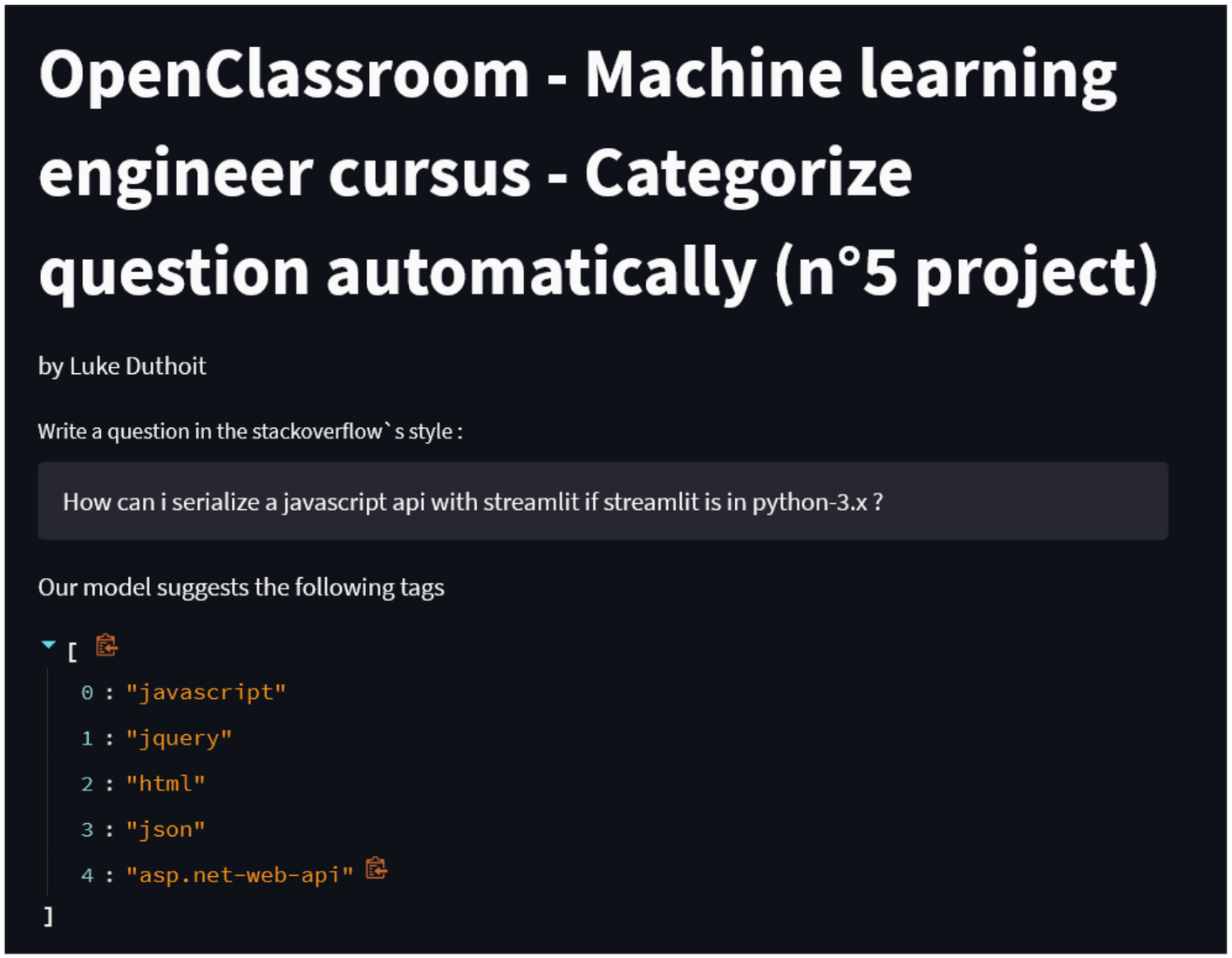
- Score médiocre, mais moins de soucis d'interprétabilité.
- Comparaison directe possible avec les questions de StackOverflow.
- Facilite la mise en place de la pipeline pour l'API.

Plan de la présentation :

- Présentation de la problématique et du prétraitement effectué, cleaning, feature engineering et exploration
- Présentation de l’approche non supervisée
- Présentation de l’approche supervisée et du modèle final sélectionné
- Démonstration de l’API et conclusion

Démonstration de l'API

1. Ouvrir un terminal;
2. Se rendre dans le répertoire de stockage du fichier python;
3. Taper « streamlit run fichier.py »;
4. Taper une phrase dans le style des questions Stackoverflow.



Conclusion…

- Projet très long et laborieux.

- Sélection des documents selon popularité pour réduire drastiquement la taille du *data set*.

- L'approche non supervisée par LDA infèrent des *topics* interprétables SEULEMENT avec un vocabulaire de mots + fréquents.

- Modèle « optimal » : Classification multi-sortie avec régression logistique sur des *features* extraites par plongement de mots de type USE.

… et perspectives

Pistes d'améliorations :

- Pour gagner du temps :
  - {*Title* + *Body*} pour TOUS les documents,
  - utiliser directement les fonctions de nettoyage/tokenisation des modèles.
- Pour l'approche non supervisée :
  - Essayer de remplacer le *BOW* par un corpus de *n-grams*.
  - Tester d'autres scores que la « cohérence » *c_v*.
- Tester d'autres modèles pour l'approche supervisée :
  - une « simple » classification directe [présence ou non du tag dans le document] en guise de *baseline* ;
  - utiliser le BERT de hub *Tensorflow*.