



# Rapport de modélisation

## Formation Data scientist

### Projet 6 : Catégorisez automatiquement des questions

12/08/2020

---

François BOYER  
francois.boyer@gmail.com

<b>Objet du document</b>	<b>2</b>
<b>Objectif du projet</b>	<b>2</b>
Description générale de l'objectif	2
Métriques de mesure de l'objectif	2
<b>Récupération des données</b>	<b>3</b>
<b>Nettoyage et exploration des données, choix des tokens et tags à utiliser</b>	<b>3</b>
Nettoyage	3
Exploration des données	4
Choix des tokens à utiliser pour les questions	4
Choix des tags à utiliser	4
<b>Modélisation des documents sous forme d'un "embedding"</b>	<b>5</b>
<b>Test de différents algorithmes de modélisation sur la base de l'embedding</b>	<b>6</b>
Analyse du niveau d'apprentissage en fonction de la quantité de données fournies	6
Algorithme KNN (approche supervisée)	7
Algorithme LDA (approche non supervisée)	9
Algorithme CART (approche supervisée avec arbre de décision)	10
Algorithme Perceptron (approche supervisée)	10
<b>Sélection du meilleur modèle et optimisation des hyper paramètres</b>	<b>11</b>
<b>Compromis precision / rappel : optimisation du seuil de probabilité pour les prédictions</b>	<b>12</b>
<b>Réalisation d'une API avec interface utilisateur pour le modèle final</b>	<b>13</b>
<b>Annexes</b>	<b>15</b>
Annexe 1 : requêtes de récupération des données sur le site de stackexchange	15
Annexe 2 : analyse du résultat produit par doc2vec sur un exemple	16
Annexe 3 : explication de l'algorithme Latent Dirichlet Allocation	16
Annexe 4 : explication de l'algorithme Doc2Vec	17

## Objet du document

L'objet de ce document est de présenter la démarche qui a été suivie afin de réaliser le projet 6 de la formation **Data Scientist** proposée par **OpenClassrooms** en partenariat avec **CentraleSupélec**.

On y trouvera un cadrage initial de l'objectif du projet, ainsi que les différentes étapes de modélisation avec une explication synthétique du fonctionnement des algorithmes utilisés.

## Objectif du projet

### Description générale de l'objectif

L'objectif du projet est de catégoriser automatiquement des questions : il s'agit de mettre en oeuvre **un système de suggestion de tags** associés aux questions posées par les utilisateurs sur le site **Stackoverflow**.

Nous avons formulé l'objectif comme :

- un problème de classification **supervisée** car nous allons utiliser les tags déjà assignés aux données d'apprentissage, qui feront office d'étiquettes, ou labels en anglais
- Une classification **multi label** (car chaque question peut être associée à 1 ou plusieurs tags), avec 2 classes possibles pour chaque label ("1" si le label est présent, "0" sinon)

A noter que nous verrons aussi une approche totalement non supervisée (donc n'utilisant pas du tout les tags des posts) et mettrons en évidence la limite de cette approche.

### Métriques de mesure de l'objectif

Dans une classification supervisée on distingue 2 mesures principales :

la précision (precision en anglais) et le rappel (recall en anglais)

Nous avons fait le choix de **privilégier la précision** par rapport au rappel car nous souhaitons en priorité que les tags suggérés par le modèle (donc prédits à 1) le soient à juste titre, plutôt que de suggérer des tags de façon exhaustive : en d'autres termes nous privilégions la qualité des suggestions à la quantité des tags suggérés.

Néanmoins nous avons également souhaité que le modèle suggère au moins 1 tag par question dans 80 ou 90% des cas.

De plus, nous avons retenu la métrique de **Precision micro** plutôt que macro, car la precision macro accorde une importance égale à toutes les labels (y compris les minoritaires ou les plus difficiles à prédire), alors que la precision micro accorde une importance égale à chaque prédiction quel que soit son label.

En effet la precision macro est une moyenne des scores de precision de chaque tag pris individuellement alors que la precision micro réalise d'abord le comptage individuel des vrais positifs et faux positifs de toutes les instances avant d'appliquer la formule de la precision.

En synthèse nous évaluerons donc les performances des modèles sur la base :

- Du score precision micro qui devra être le plus élevé possible
- Du pourcentage de documents ayant au moins 1 tag suggéré (prédit à 1) par le modèle qui ne devra pas descendre en dessous de 80%

## Récupération des données

Les données sont accessibles via l'url stackexchange explorer : pour ce faire il est nécessaire d'y saisir des requêtes SQL (voir les détails en Annexe)

Nous avons réalisé les choix suivants pour l'extraction :

- Récupération de 300 000 posts : ce nombre semblait empiriquement raisonnable pour les performances sur le poste de travail utilisé (i7-4700HQ CPU @ 2.40GHz 8 CPU, 8 Go de RAM)
- Récupération des données entre le 01/01/2019 et le 28/02/2019 : données relativement récentes, mais datant de plus d'1 an afin de laisser le temps aux tags de se stabiliser
- Nous avons fait attention de filtrer la requête pour ne récupérer que les **questions**, sans les réponses (qui ne font pas partie de l'objectif)

## Nettoyage et exploration des données, choix des tokens et tags à utiliser

### Nettoyage

Nous avons conservé 3 données texte pour chaque document (chaque question) : Title, Body et Tags

Tout d'abord nous avons nettoyé les données en supprimant les posts avec un texte vide, et en supprimant les tags HTML rajoutés par Stackoverflow dans le corps des posts (mais nous avons conservé les tags faisant partie du code des utilisateurs).

Les tags étaient présents entourés de caractères < et > (ex: <tag1><tag2>) que nous avons supprimés, et nous avons encodé les tags sous forme d'1 colonne par tag avec valeur 0 ou 1

## Exploration des données

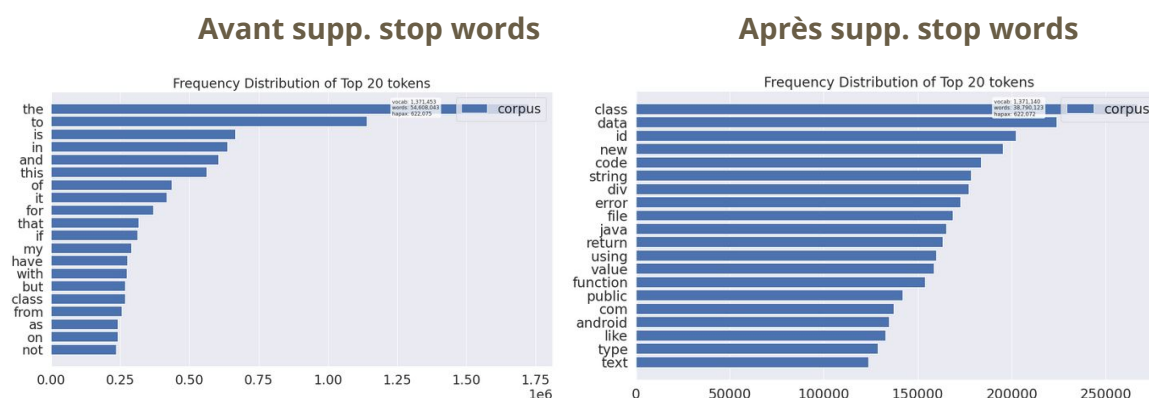
Nous avons :

- 299 798 documents
- un vocabulaire de 1.6 Millions de tokens différents. Ce nombre est élevé (si on le compare au nombre de 170 000 mots couramment utilisés en langue anglaise), cela s'explique par la nature technique du vocabulaire employé.
- 26431 tags distincts

## Choix des tokens à utiliser pour les questions

Nous avons fait le choix de supprimer les "stop words" issus de la langue anglaise (définis dans scikit learn).

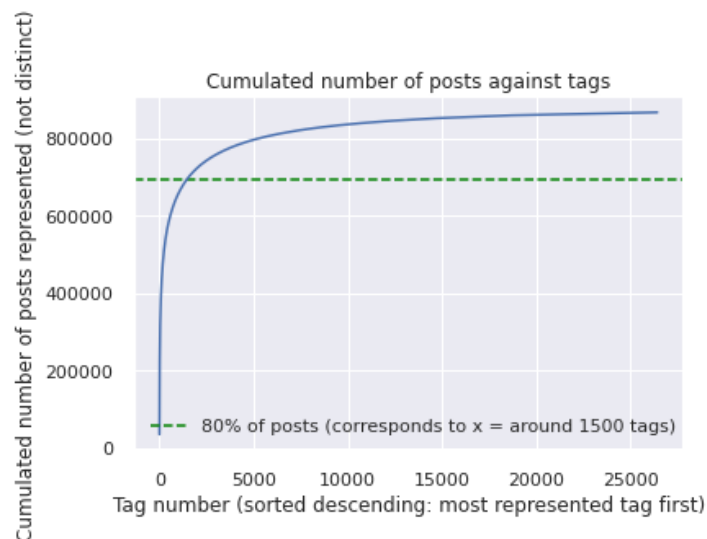
Ci-dessous la représentation du top 50 des tokens avant et après suppression des stop words



## Choix des tags à utiliser

Pour optimiser les performances et la fiabilité des métriques associées à leur mesure nous avons choisi de ne retenir que les tags les plus représentés dans le dataset.

Ainsi nous avons d'abord représenté le nombre d'occurrences cumulées de chaque tag dans les posts, en fonction des tags classés par ordre décroissant d'occurrence.



On voit que 80% des occurrences de post (ce sont donc des posts non distincts, un tag apparaissant dans plusieurs posts) sont couverts par 1500 tags, ce qui donne une première fourchette haute du nombre de tags à retenir.

Néanmoins pour des raisons de performance nous avons ajouté une limite supplémentaire: nous n'avons récupéré que les tags présents dans plus de 0.1% des documents.

Cela nous donne **373 tags** retenus pour la suite de la modélisation

Sur ces tags nous avons ensuite mesuré qu'ils couvrent **92%** des posts distincts (c'est à dire que 92% des posts ont au moins 1 tag de prédit)

Le fait de monter à 1000 tags aurait augmenté la couverture à 95% des posts distincts.

## Modélisation des documents sous forme d'un "embedding"

Afin de transformer le texte des documents en features, nous avons employé une technique de vectorisation sous forme d'un plongement (embedding en anglais)

Tout d'abord chaque document est représenté sous forme d'un vecteur qui comprend les différents mots (tokens) du document. A noter que nous avons concaténé le titre et l'objet de chaque document ce qui nous a permis de conserver les titres, qui peuvent être porteurs de sens.

Ensuite nous appliquons l'algorithme d'apprentissage non supervisé **doc2vec** (fondé sur le principe que les mots apparaissant dans des contextes similaires ont des significations apparentées) pour convertir chaque vecteur de document en un nouveau vecteur de

dimensions plus réduite (que nous avons fixée à **200**) qui sera porteur de la sémantique du document.

Nous avons fixé les autres hyper paramètres suivants : fenêtre glissante de **5** (distance minimum entre un mot du texte et le prochain mot prédit par le modèle), fréquence minimum d'occurrence des mots pour qu'ils soient pris en compte par l'algorithme : **5**

Enfin nous effectuons un Standard Scaling : on transforme chaque vecteur de sorte que la moyenne de tous les éléments soit de 0 avec un écart type de 1 (on soustrait chaque élément à la moyenne, et on divise par l'écart type)

Après avoir entraîné l'algorithme doc2vec, nous avons pris un document au hasard et récupéré les 10 documents qui lui sont le plus proche dans l'espace de plongement : **il en ressort que les tags des documents obtenus sont cohérents avec le document d'origine** : voir annexe.

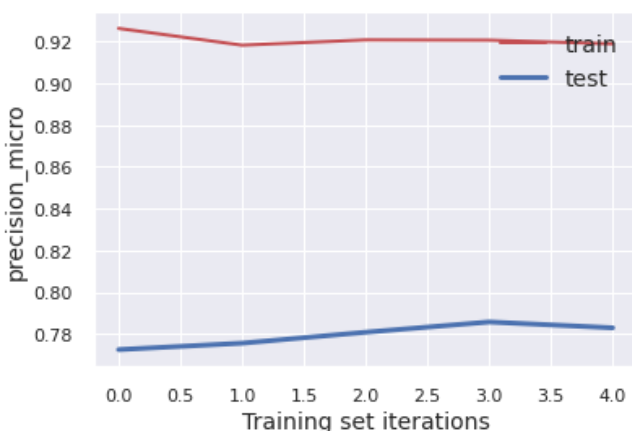
## Test de différents algorithmes de modélisation fondés sur l'embedding

### Analyse du niveau d'apprentissage en fonction de la quantité de données fournies

Afin d'appliquer un compromis entre les performances des modèles à tester et leur temps d'entraînement, nous avons successivement entraîné un prédicteur KNN sur un nombre d'instances de données d'entraînement de plus en plus grand, et comparé les precision scores micro obtenus, afin de choisir le nombre d'instances à utiliser à partir desquelles le modèle ne progresse plus beaucoup.

Ci-dessous l'itération 0 (en abscisse) correspond à 20 000 documents dans le training set, et l'itération 4 correspond à 100 000 documents

**Precision against training set iterations**



⇒ Nous avons retenu **90 000** instances pour les tests de modèle qui ont suivi

Ainsi sur la base de l'embedding doc2vec obtenu précédemment et des 90 000 instances / 373 tags retenus, nous avons testé les modèles suivants :

### Algorithme KNN (approche supervisée)

Nous avons entraîné un algorithme K Nearest Neighbours (avec  $k = 5$ ) et obtenu les résultats suivants :

	<b>Precision micro</b>	Exact match (accuracy)	<b>% docs avec au moins 1 tag prédit</b>	Recall
Training set	84%	16%	13%	17%
Test set	62%	11%	13%	10%

Sachant que Precision micro et % docs avec au moins 1 tag de prédit sont les indicateurs que nous avons privilégié dans la définition des objectifs : ce modèle produit déjà de bons résultats et nous verrons que c'est celui-ci que nous retiendrons par rapport aux autres pour la suite, afin de l'optimiser davantage.

Aussi, nous avons cherché à répondre aux questions suivantes :

- Le modèle est-il plus performant pour prédire les tags qui sont les plus représentés dans le training set (en nombre d'occurrences) ?
- Est-ce que le modèle généralise bien ou est-il en overfit ?

Pour cela nous avons comparé pour chaque tag, les scores de precision obtenus en fonction du nombre d'occurrences de ce tag dans les labels du training set :





Nous constatons qu'à partir d'environ 500 instances par tag, le precision score ne descend plus en dessous de 0.7 : il y a donc un lien entre les performances et le nombre de tags représentés dans le training set, mais ce lien est faible, car la grande majorité des instances du training set sont déjà à moins de 500 instances et plus de 0.7 de precision.

En revanche, si nous regardons le même graphe sur le test set (ci-dessous), nous constatons un fort overfit avec de nombreuses instances ayant une précision à 0 : nous faisons l'hypothèse que l'augmentation du nombre de voisins de l'algorithme KNN, actuellement à 5, permettra d'atténuer ce problème.



## Algorithme LDA (approche non supervisée)

Toujours sur la base de la transformation doc2vec vue précédemment, nous avons expérimenté une approche totalement non supervisée afin d'extraire des tags (donc, sans utiliser les tags fournis dans le jeu de donnée). Pour ce faire nous avons inféré des topics pour les documents, via l'approche LDA (Latent Dirichlet Allocation)

⇒ Vu la nature très technique du sujet des posts, les tags obtenus via cette méthode ne sont pas très pertinents. Ils portent toutefois un certain sens, mais moins précis que les vrais tags.

Topic #0: state let view title return

Topic #1: com google example https app

Topic #2: data 10 using like code

Topic #3: 2019 01 00 10 12

Topic #4: org java apache version core

Topic #5: input array form field type

Topic #6: http html content server application

Topic #7: lib py local file line

Topic #8: java exception method main run

Topic #9: log json console event result

Topic #10: php https url com link

etc.... (50 topics)

## Algorithme CART (approche supervisée avec arbre de décision)

Les résultats obtenus avec l'approche arbre de décision sont nettement moins bons qu'avec l'approche KNN.

Nous n'avons pas analysé davantage ce point dans le cadre de ce projet, néanmoins, l'hypothèse suivante pourrait l'expliquer : l'algorithme CART tel qu'il est conçu ne permet pas de lier les valeurs entre elles de façon relative (il peut par exemple apprendre que  $A1 < 2$  et  $A2 > 1$ , mais pas  $A1 > A2$ ), or dans l'espace de plongement doc2vec, nous pensons que cela est important d'apprendre cela afin de capter des vecteurs colinéaires : or KNN permet nativement de le faire et nous pensons que c'est pour cela qu'il fonctionne mieux que CART.

Cette hypothèse nécessiterait toutefois d'être validée plus précisément.

	<b>Precision micro</b>	Exact match (accuracy)	<b>% docs avec au moins 1 tag prédit</b>	Recall
Training set	18%	12%	Non mesuré	19%
Test set	7%	Non mesuré	Non mesuré	2%

## Algorithme Perceptron (approche supervisée)

Le Perceptron 1 couche, dans son implémentation par défaut avec scikit learn, n'a pas fourni non plus de très bons résultats comparé à KNN.

	<b>Precision micro</b>	Exact match (accuracy)	<b>% docs avec au moins 1 tag prédit</b>	Recall
Training set	28%	Non mesuré	Non mesuré	Non mesuré
Test set	Non mesuré	Non mesuré	Non mesuré	Non mesuré

## Sélection du meilleur modèle et optimisation des hyper paramètres

doc2vec : n dim	knn : n neighbors	mean test score
200	10	78%
200	5	62%
10	10	56%
10	5	40%

Voici les résultats du meilleur modèle parmi les 4 ci-dessus, avec 200 dimensions doc2vec et 10 voisins :

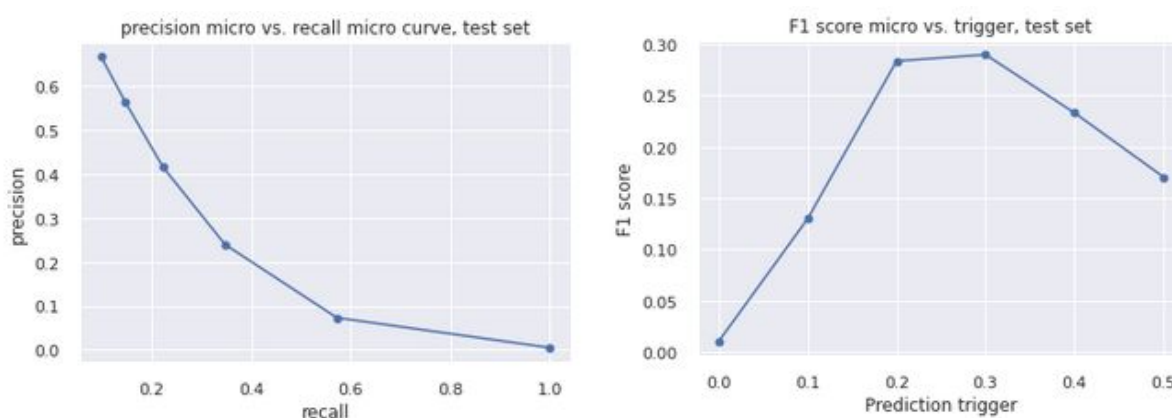
	<b>Precision micro</b>	Exact match (accuracy)	<b>% docs avec au moins 1 tag prédit</b>	Recall
Training set	87%	14%	Non mesuré	12%
Test set	77%	11%	13%	6%

On note que le pourcentage de documents avec au moins 1 tag de prédit, 13%, est très faible ⇒ Nous allons voir dans la prochaine section comment l'améliorer.

## Compromis precision / rappel : optimisation du seuil de probabilité pour les prédictions

Nous nous étions fixé comme objectif au départ de privilégier le score de precision par rapport au rappel (recall).

Pour ce faire nous avons visualisé les valeurs de précision et rappel et le F1 score en fonction de différents seuils de probabilité pour les prédictions.



⇒ Ainsi nous avons choisi un seuil à 0.3, ce qui conduit à une precision d'environ 0.4 et un rappel d'environ 0.2

	Precision micro	Exact match (accuracy)	% docs avec au moins 1 tag prédit	Recall
Training set	60%	13%	Non mesuré	34%
Test set	41%	11%	64%	22%

⇒ Le recall et le % de docs avec au moins 1 tag de prédit sont nettement améliorés

⇒ C'est donc ce modèle que nous retiendrons pour l'interface utilisateur finale

Si on visualise de nouveau la précision sur le test set en fonction des tags représentés dans les labels du training set :



On constate toujours de nombreux tags ayant une précision à 0 : 144 sur le test set vs 9 sur le training set.

Nous n'avons pas analysé ce point davantage dans le cadre de ce projet car le modèle déjà obtenu fournit des prédictions que nous jugeons acceptables.

## Réalisation d'une API avec interface utilisateur pour le modèle final


L'API est disponible sous forme d'une interface utilisateur implémentée via la librairie Streamlit.

Elle permet de saisir un objet, un corps de texte, et fournit les prédictions du modèle ainsi que les probabilités qu'il a fournies pour chaque tag, lorsqu'elles sont positives.

**Model analysis**  
☐ Display debug data

## Openclassrooms Data Science training project 6 : categorize questions

François BOYER



Enter object and body

Post object:

MATERIAL-UI React - Popper of another Popper

Post body:

```
};
}
```

hope it was clear enough, let me know if anything else is needed. Thank you

Model predictions :

Tags_javascript	Tags_reactjs
0	1

Tag probabilities

	Tags_angular	Tags_html	Tags_javascript	Tags_reactjs
0	0.1000	0.1000	0.5000	0.6000


Au moment de la rédaction de ce rapport et pour la soutenance, l'API est accessible à l'adresse : <https://pj6.analysons.com/>


Son code source est également accessible via Github : voici le lien vers le guide d'installation :


<https://github.com/Severac/openclassrooms/blob/master/PJ6/README%20API.txt>


Voici quelques exemples de commits du code de l'API sous GIT :

Commits on Jul 18, 2020


**PJ6 API : initialisation**  
 Severac committed 14 days ago


 10c446c <>

**PJ6 GridSearch avancée : fonction scoring precision micro, et prepara...**  
 Severac committed 15 days ago

 d600479 <>

Commits on Jul 28, 2020

**PJ6: update mineure UI**  
 Severac committed 4 days ago

 b0139f9 <>

## Perspectives

Nous avons identifié quelques perspectives d'amélioration du modèle et de la démarche :

- Trouver une formalisation plus précise du choix du dernier modèle (avec le trigger de probabilité) : par exemple : utilisation du score F beta avec une pondération attribuée au score de precision qui soit plus importante que le recall. Et relancer une validation croisée sur cette base.
- Analyser davantage les différents cas pour lesquels le modèle fait des erreurs : par exemple, analyser un échantillon de cas avec precision à 0 afin d'identifier des améliorations possibles
- Analyser plus précisément les résultats d'autres modèles comme le Decision Tree
- Pour la validation croisée : faire un stratified split fondé sur une répartition uniforme des différentes features : via un clustering fondé sur les features du doc2vec par exemple, en faisant le stratified split sur la base des clusters ainsi obtenus.

## Annexes

### Annexe 1 : requêtes de récupération des données sur le site de stackexchange

Les données ont été récupérées sur le site de stackexchange via requêtes SQL :


<https://data.stackexchange.com/stackoverflow/query/new>

Il a été nécessaire d'exécuter plusieurs requêtes car le nombre de lignes (= nombre de posts) renvoyées pour chaque requête est limité à 50000.

Voici les requêtes qui ont été utilisées pour ce faire :

```
Select * from posts where CreationDate > '2019-01-01' and ParentId is NULL  
order by Id
```





```
Select * from posts where CreationDate > '2019-01-01' and ParentId is NULL  
and Id > 54150048 order by Id
```

```
Select * from posts where CreationDate > '2019-01-01' and ParentId is NULL  
and Id > 54306674 order by Id
```

```
Select * from posts where CreationDate > '2019-01-01' and ParentId is NULL  
and Id > 54462919 order by Id
```

```
Select * from posts where CreationDate > '2019-01-01' and ParentId is NULL  
and Id > 54619439 order by Id
```

```
Select * from posts where CreationDate > '2019-01-01' and ParentId is NULL  
and Id > 54774301 order by Id
```

## Annexe 2 : analyse du résultat produit par doc2vec sur un exemple

Si on prend le document dont le texte est le suivant :

```
<?php
```

```
$str = "xyz@gmail.com";

$username = strstr($str, '@', true);

echo $username;
```

```
?>
```

**I want to remove sender name before @ tag and output should be @gmail.com. Now, It shows me xyz output which I don't want. So, How can I do this? Please help me.**

**I want to output @gmail.com.**

Et que l'on regarde les tags des documents qui lui sont proches dans l'espace de l'embedding, on constate une cohérence : la quasi totalité des documents proches ont le tag php, ce qui correspond au langage de la question de départ. D'autres tags sont également cohérents comme regex, string.

	Tags_arrays	Tags_file	Tags_firebase	Tags_google-cloud-functions	Tags_google-cloud-platform	Tags_heroku	Tags_laravel	Tags_macos	Tags_php	Tags_python	Tags_regex	Tags_string	Tags_windows
1000	0	0	0	0	0	0	0	0	1	0	0	0	0
3669	0	0	0	0	0	0	0	0	1	0	0	0	0
52968	1	0	0	0	0	0	0	0	1	0	0	1	0
94276	0	0	1	1	1	0	0	0	1	0	0	0	0
110681	0	0	0	0	0	0	0	0	1	0	0	1	0
119275	0	0	0	0	0	1	1	0	1	0	0	0	1
130575	0	0	0	0	0	0	0	0	0	1	0	0	0
147035	0	1	0	0	0	0	0	0	1	0	1	0	0
163205	0	0	0	0	0	0	0	1	0	0	0	0	0
197221	0	0	0	0	0	0	0	0	1	0	1	0	0

## Annexe 3 : explication de l'algorithme Latent Dirichlet Allocation

Source : [https://fr.wikipedia.org/wiki/Allocation\\_de\\_Dirichlet\\_latente](https://fr.wikipedia.org/wiki/Allocation_de_Dirichlet_latente)

**LDA** est un modèle génératif probabiliste de type "topic modeling" : il suppose que chaque document est un mélange d'un petit nombre de sujets ou thèmes, et que la génération de chaque occurrence d'un mot est attribuable (probabilité) à l'un des thèmes du document.

Fonctionnement en synthèse :

- On fixe d'abord aléatoirement un nombre de thèmes selon une distribution de Dirichlet sur un ensemble de K thèmes
- Puis on cherche à apprendre les thèmes représentés dans chaque document et les mots associés à ces thèmes, par itérations successives.

Plus précisément : pour chaque mot  $w$  de chaque document  $d$  , on calcule deux quantités pour chaque thème  $t$  :

$p(t \mid d)$  : la probabilité que le document  $d$  soit assigné au thème  $t$

$p(w \mid t)$  : la probabilité que le thème  $t$  dans le corpus soit assigné au mot  $w$

On choisit alors le nouveau thème  $t$  avec la probabilité  $p(t \mid d) \times p(w \mid t)$

Ceci correspond à la probabilité que le thème  $t$  génère le mot  $w$  dans le document  $d$

En répétant les étapes précédentes un grand nombre de fois, les assignations se stabilisent.

## Annexe 4 : explication de l'algorithme Doc2Vec

Doc2Vec est un algorithme permettant un "plongement" de mots (word embedding en anglais) créé par Tomas Mikolov.

Le papier de recherche est accessible ici :

[https://cs.stanford.edu/~quocle/paragraph\\_vector.pdf](https://cs.stanford.edu/~quocle/paragraph_vector.pdf)

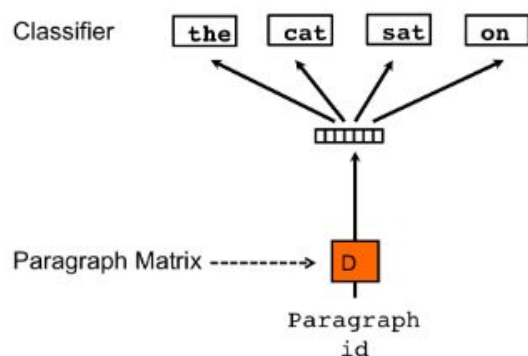
Doc2Vec permet de convertir une liste de documents (chaque document en entrée étant représenté sous forme d'une liste de tokens de nombre variable) en une liste de vecteurs de dimension fixe, qui vont représenter la signification sémantique des documents.

L'apport de doc2vec comparé à une approche classique de type bag of words, est que des mots apparaissant dans des contextes similaires ont des significations apparentées dans l'espace sémantique cible. L'approche bag of words, quant à elle, ne prend pas en compte l'ordre et le contexte d'apparition des mots.

Il existe 2 implémentations de doc2vec : PV-DBOW (Paragraph Vectors - Distributed Bag of Words) et PV-DM (Paragraph Vectors Distributed Memory).

Pour ce projet nous avons utilisé l'implémentation PV-DBOW

Le modèle PV-DBOW est un réseau de neurones à 3 couches :



Ce modèle est entraîné pour prédire, pour chaque document, un ensemble de mots (à chaque itération d'apprentissage, à partir d'un mot pris au hasard dans le document, on va prédire un ensemble de mots qui lui sont proches) :

*"at each iteration of stochastic gradient descent, we sample a text window, then sample a random word from the text window and form a classification task given the Paragraph Vector"*

**Et c'est ensuite la couche du milieu qui est utilisée pour former l'embedding du modèle de notre projet décrit dans ce rapport .**