

# Développement d'un système de suggestion de tags pour Stack Overflow



Un système de suggestion de tags revient à effectuer une classification multi-labels à partir du texte et/ou du titre d'une question. Le traitement du langage naturel implique l'utilisation de techniques spécifiques de vectorisation du texte, ou de plongements de mots ou de phrases. Deux approches sont possibles pour le modèle de classification : traiter directement l'information disponible ou passer par une étape intermédiaire de factorisation afin de réduire la dimension des variables.

# Sommaire

<b>1. Récupération et préparation des données</b> .....	1
1.1. Téléchargement des données .....	1
1.2. Préparation des données .....	1
1.2.1. Body .....	1
1.2.2. Tags .....	1
1.2.3. Title .....	1
1.3. Exploration des données .....	1
1.3.1. Analyse des tags .....	1
1.3.2. Texte des questions .....	2
1.3.3. Analyse des titres .....	3
1.4. A propos des stop-words .....	3
<b>2. Modélisation des sujets (topic modeling)</b> .....	3
2.1. Approche du problème .....	3
2.1.1. Les différentes approches testées .....	3
2.1.2. Les algorithmes de topic modeling .....	4
2.1.3. Le score de cohérence Cv .....	4
2.2. Analyse de la vectorisation des données textuelle .....	5
2.3. Analyse des résultats du topic modeling .....	5
<b>3. Apprentissage semi-supervisé</b> .....	6
3.1. Approche théorique .....	6
3.1.1. Etapes du pipeline .....	6
3.1.2. La classification multi-labels .....	6
3.1.3. Les algorithmes de classification multi-labels .....	7
3.2. Résultats .....	7
3.2.1. Les indicateurs de mesure de performance .....	7
3.2.2. Les résultats de l'apprentissage semi-supervisé .....	7
<b>4. Apprentissage supervisé</b> .....	8
4.1. Approche théorique .....	8
4.1.1. Etapes du pipeline .....	8
4.1.2. Le plongement de phrases .....	8
4.2. Résultats .....	9
<b>5. Choix du modèle final</b> .....	10
5.1. Comparaison des performances .....	10
5.2. Modèle choisi .....	10
5.3. Evaluation de la performance de généralisation .....	10
<b>Conclusion</b> .....	10
<b>Références</b> .....	10

## 1. Récupération et préparation des données

## 1.1. Téléchargement des données

J'utilise l'outil d'exploration des données "Data Explorer" mis à disposition par Stack Overflow. Cet outil permet de requêter une base SQL qui contient les questions soumises à la communauté Stack Overflow, ainsi que les commentaires et réponses qui y ont été apportés.

Le nombre de lignes renvoyées pour une requête est limité à 50 000. J'effectue 10 requêtes consécutives afin de disposer d'une base de données conséquente de 500 000 lignes.

Ma requête demande l'identifiant (Id), le texte (Body), les tags, le score et le nombre de vues des questions, pour les seules questions qui comportent des tags.

Les questions sont triées :

- par ordre décroissant de score pour favoriser les questions les mieux notées ;
  - par ordre croissant de vue, pour favoriser à score équivalent, les questions qui ont atteint ce score le plus rapidement ;
  - par ordre décroissant d'identifiant, afin d'obtenir d'abord les questions les plus récentes.

## 1.2. Préparation des données

### 1.2.1. Body

Le corps des questions est dans le format html, avec les tags html. J'utilise la librairie BeautifulSoup pour extraire le texte du corps des questions.

Je procède en trois étapes. Je retire d'abord les exemples de code compris entre les tags <code> </code>. Ensuite, je récupère le texte restant sans les tags html grâce à la fonction get\_text. Enfin, je remplace les retours à la ligne, symbolisés par "\n", par un espace.

### 1.2.2. Tags

J'utilise une nouvelle fois la librairie BeautifulSoup pour transformer la liste de tags en liste de chaînes de caractères exploitable.

1.2.3. Title

Les titres sont déjà sous la forme de strings. Ils sont exploitables tels quels.

### 1.3. Exploration des données

### 1.3.1 Analyse des tags

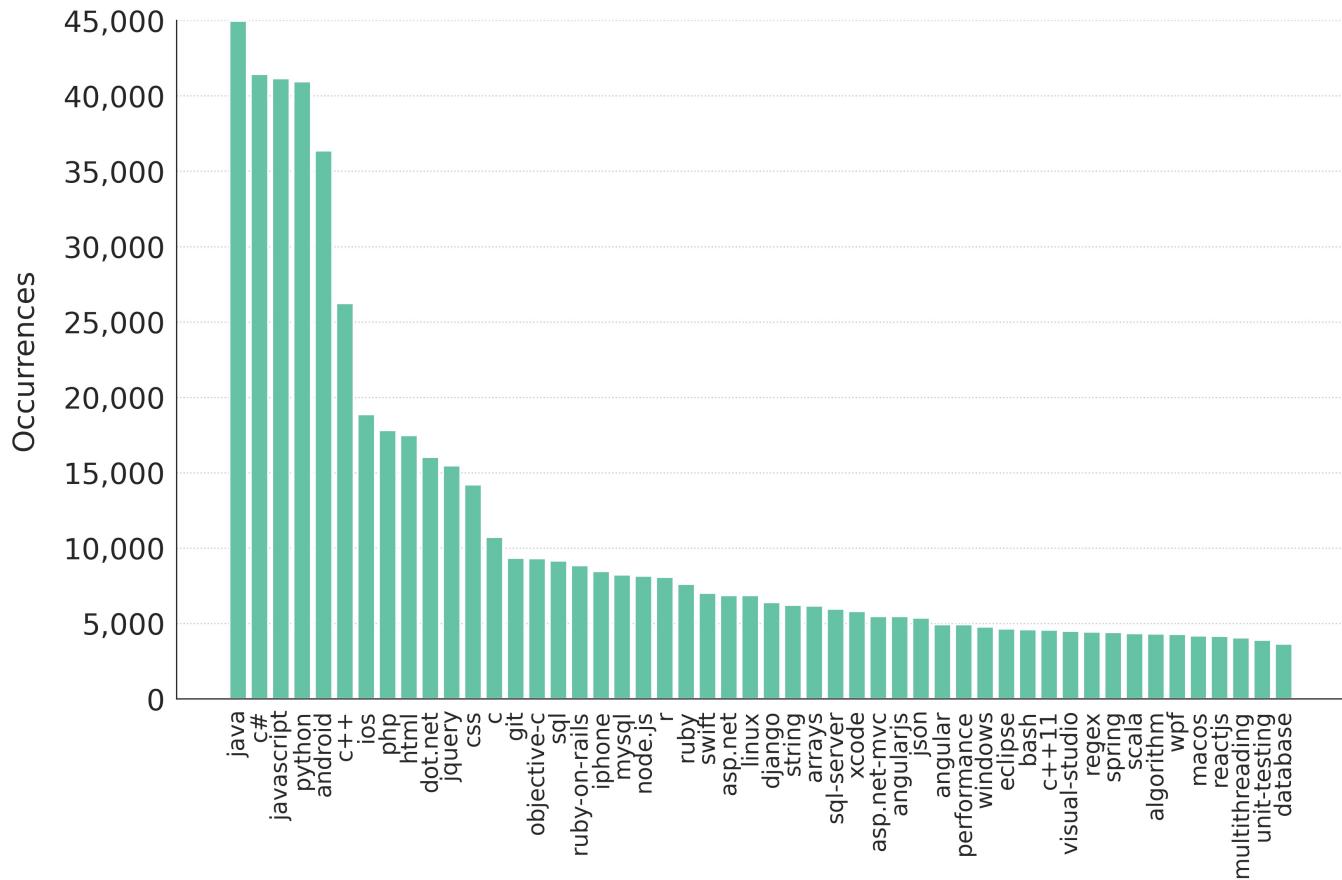


Figure 1 : tags les plus fréquents dans notre jeu de données

Le nombre de tags par question s'étend de 1 à 6. La grande majorité des questions comprend entre 2 et 4 tags. Les tags les plus fréquents (figure 1) sont très significatifs du domaine concerné par la question : java, C#, javascript, python, android, c++, ios, php, html...

Je limite mon étude au 50 tags les plus fréquents. Le cinquantième tag est présent dans moins de 1% des questions (figure 2). Les tags suivants ne sont pas suffisamment représentés pour être étudiés. Je supprime les questions qui ne contiennent aucun des 50 tags les plus fréquents.

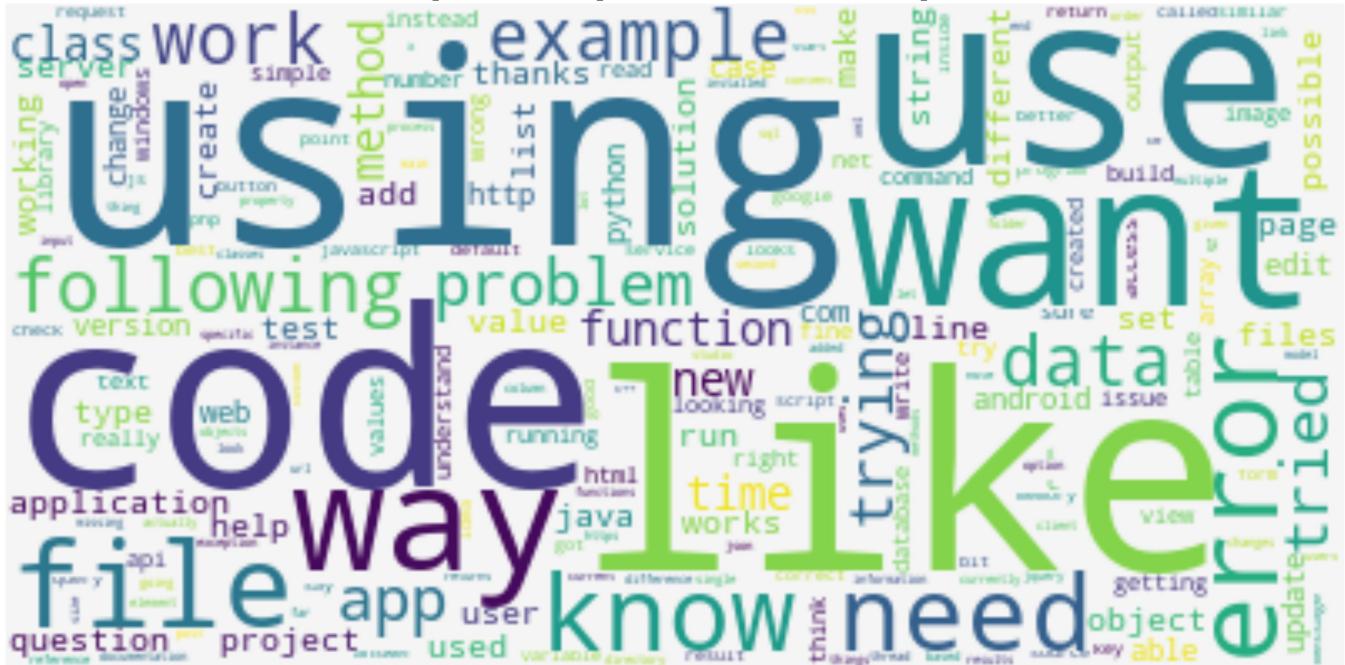
# Les 50 tags les plus fréquents



*Figure 2 : distribution des tags les plus fréquents*

### 1.3.2. Texte des questions

## Mots les plus fréquents dans les questions



*Figure 3 : mots les plus fréquents dans les questions*

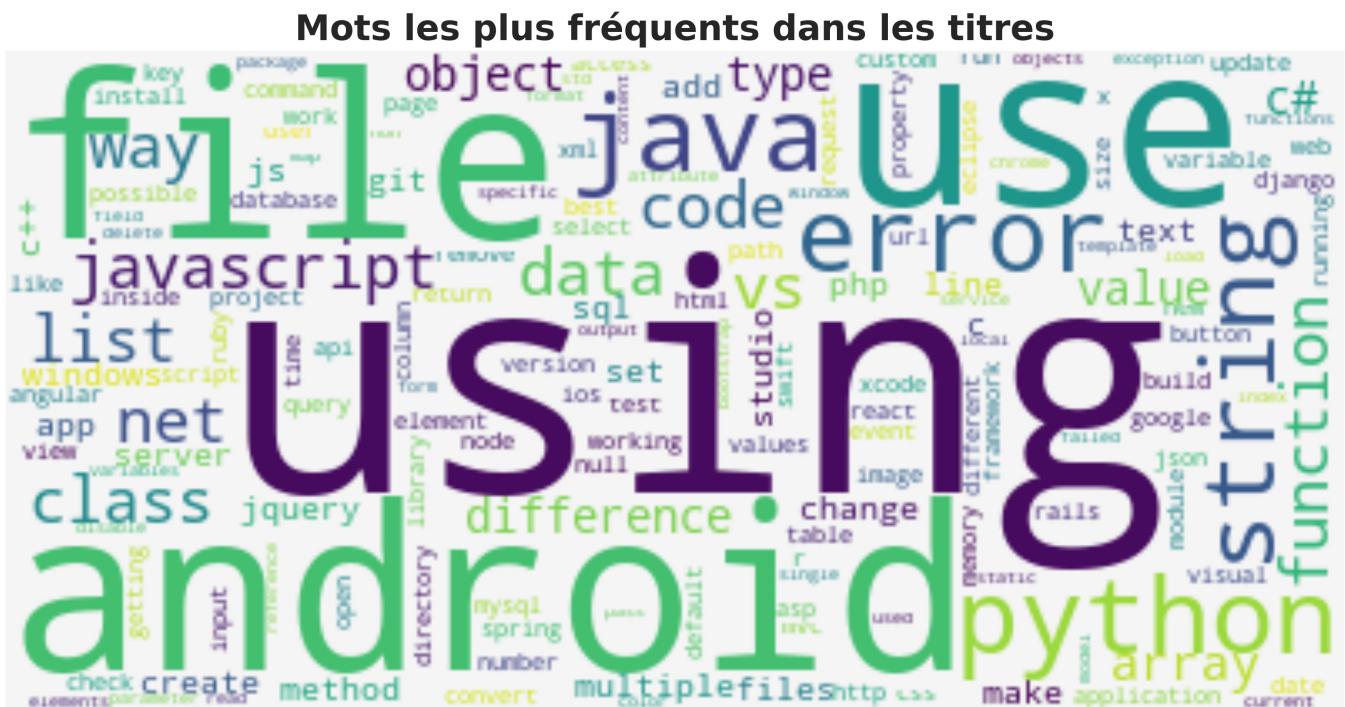
Les questions contiennent entre 1 et 3928 mots, qui se répartissent entre 1 et 780 mots uniques. La taille de la grande majorité des questions reste raisonnable : entre 1 et 222 mots, avec entre 1 et 133 mots uniques. Seules quelques questions sont beaucoup plus longues et représentent des outliers du point de vue de leur longueur.

Les mots les plus fréquents dans le texte des questions (figure 3), une fois les stopwords retirés, sont des mots très génériques qui ne semblent pas permettre de définir le sujet de la question.

### 1.3.3. Analyse des titres

Les titres sont, de manière logique, plus courts que le texte des questions. La plupart des titres comprend entre 1 et 18 mots (avec un maximum de 31 mots), quasiment tous uniques dans un même titre.

Les mots les plus fréquents dans les titres (figure 4), une fois les stopwords retirés, comprennent à la fois des mots génériques et des mots intéressants pour cerner le sujet de la question, comme android, python, java, javascript, C#, sql, jquery... Il sera donc intéressant de tester le topic modeling et la classification des labels sur la base des questions, mais également de leur titre ou de la combinaison des deux.



*Figure 4 : mots les plus fréquents dans les titres*

#### 1.4. A propos des stop-words

Différentes libraires de traitement du langage naturel proposent des listes de stop-words, plus ou moins complètes, et donc complémentaires. Les questions du site Stack Overflow étant rédigées en Anglais, je dois utiliser une liste de stop-words anglais.

J'ai choisi de créer ma propre liste de stop-words, en additionnant les listes de la librairie nltk et de la librairie sklearn. La liste de nltk possède l'avantage d'intégrer les abréviations usuelles du langage mais ne compte que 179 mots. La liste de sklearn est plus fournie avec 318 mots. En concaténant les deux listes, sans les doublons, j'obtiens une liste de 378 mots.

## 2. Modélisation des sujets (topic modeling)

### **2.1. Approche du problème**

### 2.1.1. Les différentes approches testées

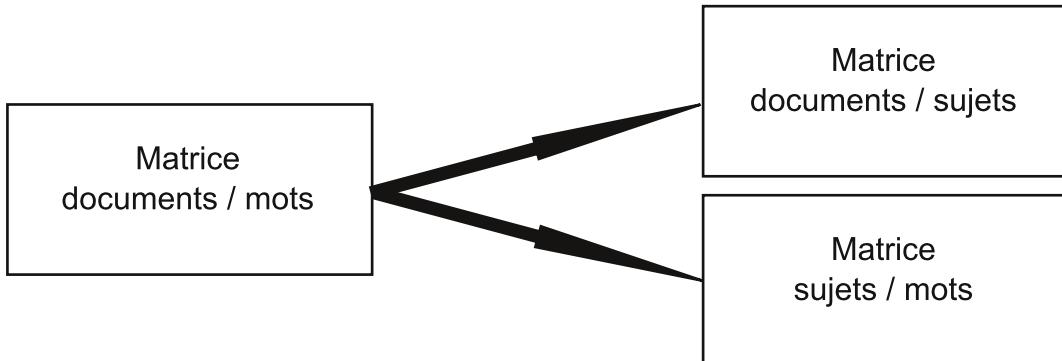
Je teste différents traitements des données textuelles et deux algorithmes de factorisation :

- la vectorisation des données sur la base d'un Bag Of Words ou d'un TF-IDF. A noter q'un plongement de mots ou de phrases est inadapté ici, car la théorie de la LDA nous pousse à nous baser sur un bag of words, et une factorisation par NMF n'accepte que des valeurs positives en entrée ;

- l'utilisation des mots simples, ou l'ajout des bigrammes, voire des trigrammes ;
- l'utilisation du texte du corps de la question, ou du titre, ou de la combinaison des deux ;
- le topic modeling avec la Latent Dirichlet Allocation (LDA) ou avec la Non-negative Matrix Factorization (NMF).

### 2.1.2. Les algorithmes de topic modeling

La modélisation de sujets consiste à décomposer la matrice vectorisée des documents en deux matrices : une matrice documents / sujets et une matrice sujets / mots. La première donne une estimation des sujets contenus dans chaque document, la seconde une appréciation des mots représentatifs de chaque sujet.



Les algorithmes LDA et NMF cherchent tous deux à estimer les deux matrices documents/sujets et sujets/mots dont la multiplication minimise l'erreur de reconstruction avec la matrice initiale. Ils se différencient par leur approche mathématique.

La LDA est un modèle probabiliste graphique. Les matrices obtenues représentent la proportion des thèmes dans chaque document et la probabilité de retrouver les mots dans chaque thème.

La NMF repose sur l'algèbre linéaire. Les deux matrices obtenues représentent le poids des thèmes dans chaque document et le poids des mots dans chaque thème.

### 2.1.3. Le score de cohérence Cv

Pour analyser les résultats, je me base sur le score de cohérence Cv, une mesure récente qui présente l'avantage de présenter une forte corrélation avec la modélisation de sujet par les humains. Ce score de cohérence se base sur la liste des documents tokénisés et se calcule en 4 étapes :

1) Segmentation des données tokénisées en paires de mots : le score crée toutes les paires possibles de mots, entre les N mots les plus probables déterminés par l'algorithme (N pouvant être modifié par l'utilisateur), puis crée les N ensembles Si qui relient chaque mot i aux autres mots.

2) Calcul des probabilités des mots et des paires de mots : ces probabilités sont estimées par le nombre de documents où le mot ou la paire de mots apparaît divisé par le nombre total de documents. Pour le score Cv, ce calcul est fait sur un jeu de documents virtuels créés avec une fenêtre coulissante sur les documents initiaux. Afin d'essayer de capturer en partie la proximité des mots au sein des documents, l'algorithme crée des documents virtuels composés de n mots qui sont décalés les uns par rapport aux autres d'un token. Le premier document virtuel comprend les n premiers tokens (de 1 à n), le second document virtuel est décalé d'un mot et comprend également n tokens (de 2 à n+1), et ainsi de suite. L'algorithme procède ainsi pour tous les documents.

3) Calcul d'une mesure de confirmation pour chaque paire de mots : elle quantifie le lien entre les deux mots à partir d'une mesure de leur similarité en relation avec tous les mots de l'espace. La similarité entre deux mots est calculée avec l'indicateur Normalised Pointwise Mutual Information (NPMI). La NPMI présente l'avantage de sur-pondérer les similarités les plus élevées et d'être très corrélé à l'appréciation humaine, d'après les études menées (cf G. Bouma, "Normalized (Pointwise) Mutual Information in Collocation Extraction", in *Proceedings of German Society for Computational Linguistics (GSCL 2009)*, 2009, p. 31-40). Un vecteur de contexte est créé pour chaque mot et est constitué des NMPI de ce mot avec tous les autres mots. La mesure de confirmation d'une paire de mots est obtenue avec la similarité cosinus entre les deux vecteurs de contexte.

4) Calcul du score de cohérence : le score de cohérence Cv est la moyenne arithmétique des mesures de confirmation de toutes les paires de mots.

## 2.2. Analyse de la vectorisation des données textuelles

La LDA est un modèle de factorisation probabilistique. Il repose sur la distribution des mots dans les documents. Il faut donc utiliser en théorie une représentation des documents en Bag Of Words (BOW), car la représentation en TF-IDF change la distribution initiale des mots dans les documents.

Nos résultats (figure 5) montrent que l'utilisation de TF-IDF au lieu de BOW donne un score de cohérence très légèrement supérieur. Une étude plus précise pointe sur l'équivalence des scores de cohérence qui sont toujours très proches et dont l'ordre varie en fonction de l'échantillon étudié. Je décide donc de coller à la théorie et de baser mon étude de la LDA sur une vectorisation en BOW.

### LDA - Coherence score according to analysis strategies

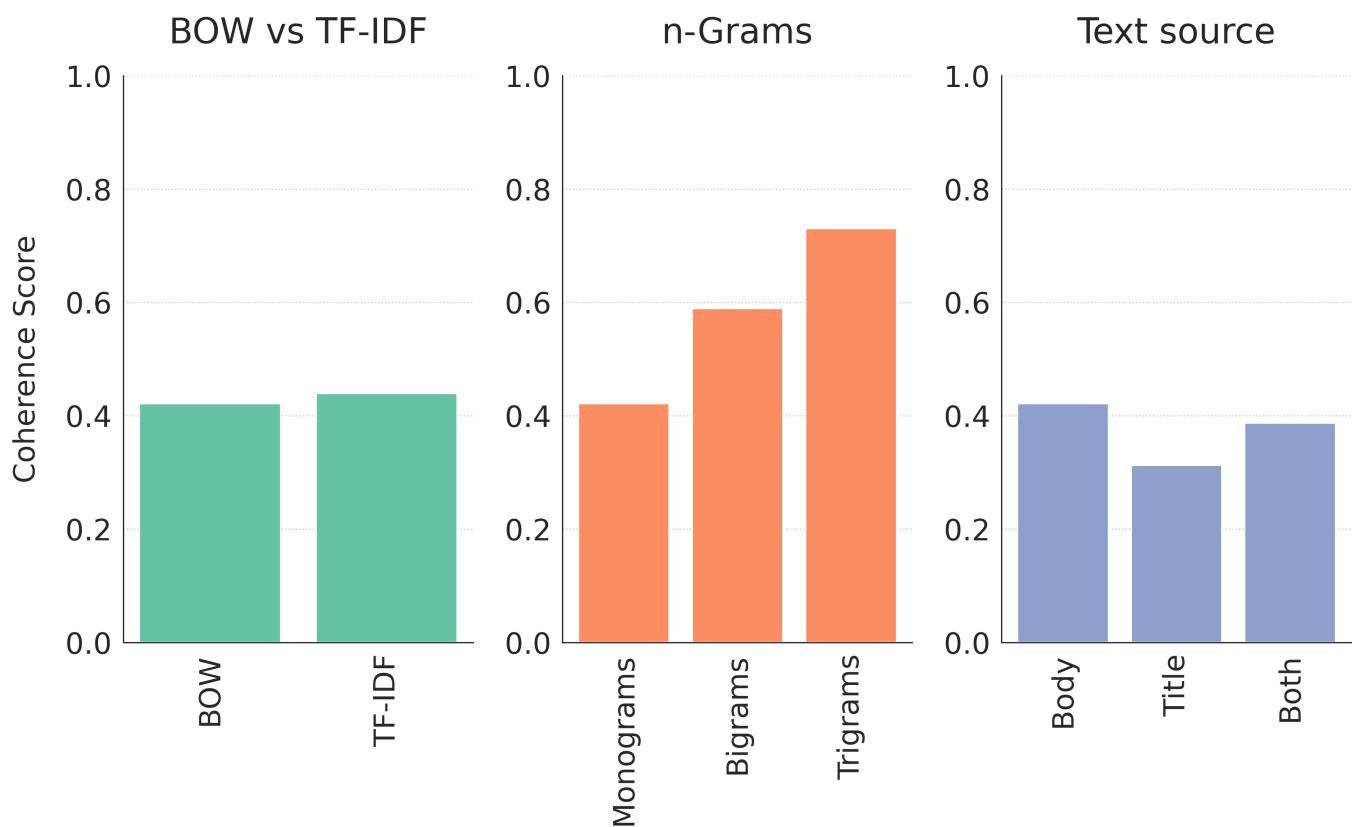


Figure 5 : impact des options de vectorisation sur le score de cohérence

L'exploration des données laissait penser que le titre serait le meilleur outil pour déterminer les sujets des questions. Les résultats contredisent cette intuition. Le meilleur score de cohérence est obtenu en utilisant le seul texte des questions.

Le critère le plus impactant sur le score de cohérence est la liste des mots utilisés. Si on base la factorisation sur une vectorisation des mots simples, le score de cohérence est plutôt bas (légèrement supérieur à 0,4). En ajoutant les bigrammes et les trigrammes aux mots simples, le score de cohérence augmente sensiblement pour dépasser les 0,7.

## 2.3. Analyse des résultats du topic modeling

J'étudie les paramètres optimaux pour la factorisation avec LDA, sur la base d'une vectorisation en Bag Of Words issue des mots simples, des bigrammes et des trigrammes du texte des questions.

Les paramètres optimaux obtenus sont un min\_df de 2% pour enlever les mots ou ensembles de mots les moins fréquents et un max\_df de 95.5% pour enlever les mots ou ensembles de mots les plus fréquents.

L'optimisation des hyperparamètres de la LDA indique un nombre optimal de 17 sujets, avec un alpha de 1,5 et un beta de 50.

Toutefois l'analyse détaillée des topics obtenus n'est pas satisfaisante pour plusieurs raisons. Les mots les plus représentatifs de chaque sujet ne permettent pas de comprendre le thème des questions de chaque topic. Ces mots les plus représentatifs sont souvent en grande partie les mêmes d'un sujet à l'autre. Cela se

traduit par le chevauchement de beaucoup de sujets sur la figure 6  
 Donc, malgré un score de cohérence élevé de plus de 78%, la factorisation par LDA n'est pas satisfaisante.  
 L'étude de la factorisation par NMF n'est pas plus satisfaisante. Elle aboutit à choisir un nombre optimal de 2 sujets, bien insuffisant par rapport aux 50 labels que nous avons conservés.

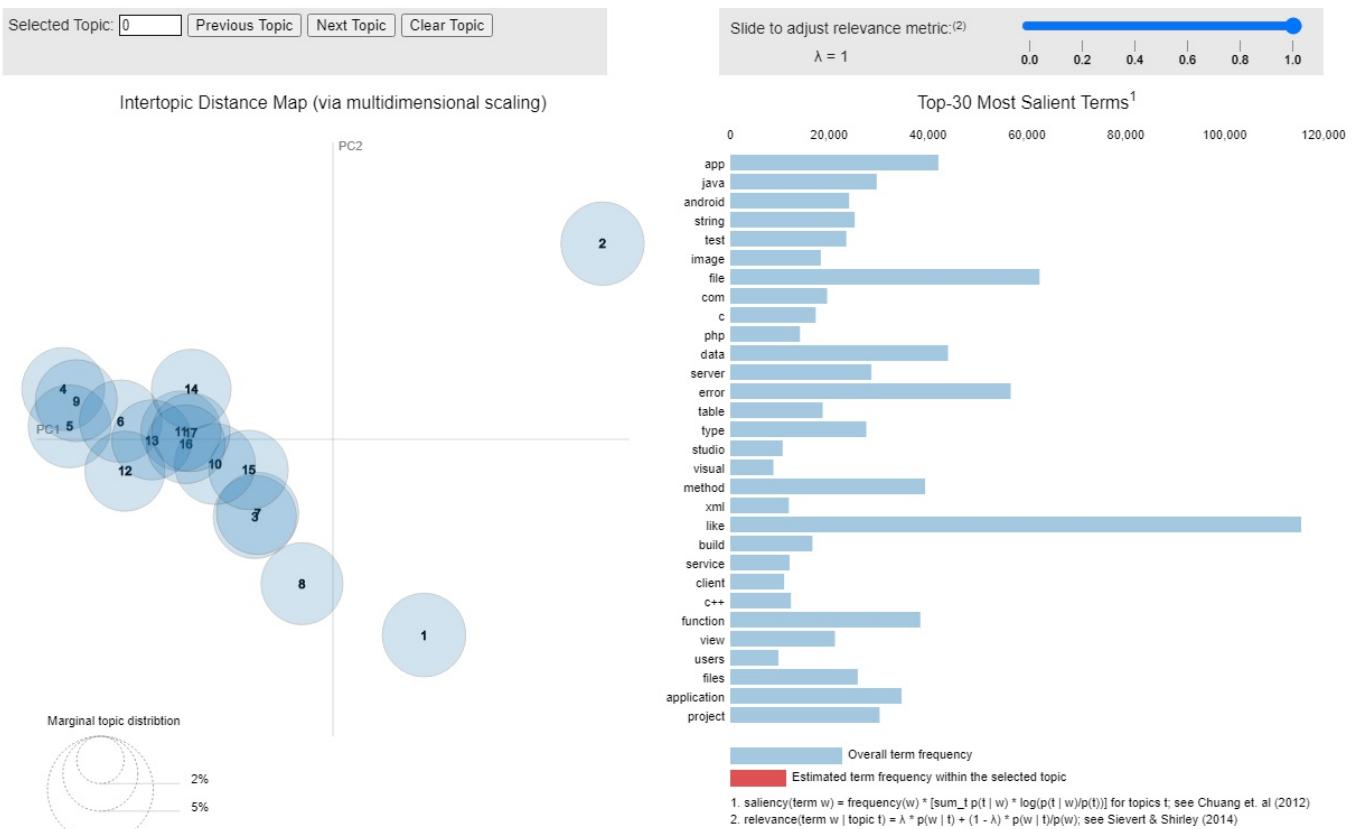


Figure 6 : les topics obtenus par la LDA ne sont pas satisfaisants

### 3. Apprentissage semi-supervisé

#### 3.1. Approche théorique

##### 3.1.1. Etapes du pipeline

L'apprentissage semi-supervisé des tags des questions repose sur le topic modeling. La factorisation préalable des données textuelles, à l'aide d'une LDA ou d'une NMF, permet de diminuer le nombre de features traitées par les modèles.

Le pipeline de l'apprentissage semi-supervisé comporte donc quatre étapes :

- une vectorisation des données textuelles en Bag Of Words ou en TF-IDF ;
- une factorisation (topic modeling) avec une LDA ou une NMF ;
- une normalisation des résultats de la factorisation avec un Standard Scaler ;
- un algorithme de classification.

##### 3.1.2. La classification multi-labels

Il existe différents types de classification.

La classification binaire consiste à attribuer à chaque individu la classe la plus probable entre deux choix, par exemple vrai/faux. Tous les algorithmes permettent de traiter ce type de problème.

La classification multi-classe cherche à déterminer à quelle classe un individu appartient parmi plusieurs choix. Une seule classe est déterminée pour chaque individu : celle avec la plus grande probabilité. Beaucoup d'algorithmes de classification peuvent nativement effectuer des classifications multi-classes. Sinon, les stratégies OVR (one vs rest) ou OVO (one vs one) permettent d'exercer une telle classification à partir de classifications binaires.

La classification multi-labels consiste à attribuer une ou plusieurs classes à un même individu. Il ne s'agit plus de trouver une seule classe, la plus probable, pour chaque individu mais de déterminer quelles sont les classes (une ou plusieurs) auxquelles l'individu appartient probablement. Notre projet se situe dans ce contexte.

### 3.1.3. Les algorithmes de classification multi-labels

La classification multi-labels nécessite une approche spécifique. J'utilise la librairie scikit-multilearn qui implémente des algorithmes spécifiques. J'ai sélectionné quatre algorithmes :

- Binary Relevance : cette approche transforme la classification multi-labels avec L labels en L problèmes de classification binaire. La prédiction est l'union des résultats des L classifieurs. La relation entre les labels est ignorée.
- Classifier Chain : cette méthode entraîne L classifieurs ordonnés les uns après les autres en fonction de la règle de la chaîne bayésienne. Le premier classifieur est entraîné sur les données d'entrée. Les suivants sont entraînés sur ces mêmes données auxquelles on ajoute les résultats des classifieurs précédents. Cela permet de prendre en compte les relations entre les labels, et peut généraliser sur de nouvelles combinaisons de labels.
- Label Powerset : cette solution ramène la classification multi-labels à une classification multi-classe où toutes les combinaisons de labels existantes dans le jeu d'entraînement constituent une classe différente. Elle prend en compte la relation entre les labels mais ne peut pas généraliser sur de nouvelles combinaisons de labels.
- MLkNN : cette adaptation de la classification kNN aux multi-labels utilise une approche kNN pour trouver les exemples les plus proches puis utilise une inférence bayésienne pour assigner les labels à un individu.

Les trois premiers algorithmes reposent sur un classifieur de base de scikit-learn tel que la régression logistique, les forêts aléatoires, les SVM, un classifieur naïf bayésien multinomial ou encore XGBoost.

## 3.2. Résultats

### 3.2.1. Les indicateurs de mesure de performance

La performance de classification multi-labels peut être mesurée par de nombreux indicateurs :

- la hamming loss est une mesure de perte, ce qui signifie qu'une valeur moins élevée est meilleure. Elle mesure la proportion de labels qui sont incorrectement prédis, en calculant cette fraction sur tous les labels pour chaque individu, puis en faisant une moyenne par individu puis par label. Elle reflète donc la proportion de labels incorrects ou manquants.
- l'accuracy score donne le pourcentage d'invidus pour lesquels le modèle prédit correctement tous les labels. Elle est très exigeante car elle exclut tout individu pour lequel un label est manquant ou incorrect.
- le F1 score est une moyenne de la précision (le taux des prédictions positives qui sont réellement positives) et du rappel (le pourcentage de valeurs positives qui sont correctement prédictes). Le calcul "micro" utilise les valeurs globales, en additionnant tous les vrais positifs, faux négatifs et faux positifs de tous les labels. Le calcul "macro" effectue une moyenne du F1 score de chaque label. Il est plus exigeant en présence d'un grand nombre de labels, surtout lorsque certains sont peu représentés et susceptibles d'avoir des F1 scores plus faibles. Le calcul "weighted" reprend le calcul "macro" mais en pondérant les labels en fonction de leur fréquence, pour compenser les déséquilibres de fréquence des labels.
- le score de similarité de Jaccard calcule la moyenne de la proportion de labels correctement prédis par individu.

J'ai décidé de baser mes tests sur la hamming loss car je considère qu'il est préférable d'avoir moins de labels manquants ou faux par individu plutôt que d'avoir le maximum d'individus avec exactement tous les labels prédis.

### 3.2.2. Les résultats de l'apprentissage semi-supervisé

Les premières expérimentations montrent que les approches Binary Relevance et Classifier Chain obtiennent une meilleur hamming loss que Label Powerset. Cette dernière, qui souffre d'un temps de calcul particulièrement long, présente par contre un meilleur accuracy score. Je continue la suite de ma

recherche avec les seules approches Binary Relevance et Classifier Chain.

J'effectue une optimisation simultanée des options et hyperparamètres de :

- vectorisation : BOW ou TF-IDF, et leurs paramètres sous-jacents ngram range, min df et max df ;
- factorisation : LDA ou NMF, avec le paramètre n\_components (plus alpha et beta pour LDA) ;
- algorithme multi-label : Binary Relevance ou Classifier Chain ;
- algorithme de classification sous-jacent : régression logistique, forêt aléatoire, SVM, modèle naïf bayésien multinomial ou XGBoost, et leurs hyperparamètres respectifs.

Je compare ces résultats à une optimisation de l'algorithme MLkNN (optimisation de la vectorisation, de la factorisation, et des hyperparamètres k et s de MLkNN).

Les meilleurs résultats correspondent à une vectorisation en Bag Of Words sur les mots simples (pas de bigrammes ou de trigrammes), avec un min df de 0,3% et un max df de 96,8%. La factorisation est basée sur une NMF avec 90 composantes. Le score de cohérence qui correspond à cette modélisation des sujets est de 29,36%. On voit donc que les meilleurs résultats de classification ne résultent pas des meilleurs scores de cohérence.

La classification multi-labels la plus performante est ensuite obtenue avec un Classifier Chain sur une forêt aléatoire. La hamming loss de validation est de 0,0183, pour un accuracy score de 32,51% et un F1 score weighted de 53,60%.

## 4. Apprentissage supervisé

### 4.1. Approche théorique

#### 4.1.1. Etapes du pipeline

L'apprentissage supervisé des tags des questions reprend la même logique que la modélisation semi-supervisée sans l'étape intermédiaire de la factorisation. Par conséquent, nos algorithmes d'apprentissage doivent traiter beaucoup plus de features. Afin de conserver un temps de calcul raisonnable, je n'ai conservé que trois algorithmes de classification : la régression logistique, le SVM linéaire, et le modèle naïf bayésien multinomial.

Le pipeline de l'apprentissage supervisé comporte donc trois étapes :

- une vectorisation des données textuelles : soit à l'aide d'un Bag Of Words ou en TF-IDF, soit à l'aide d'un plongement de phrases avec la version 4 du Universal Sentence Encoder de Google ;
- une normalisation des données vectorisées avec un Standard Scaler ;
- un algorithme de classification multi-label : je conserve les deux options Binary Relevance et Classifier Chain, avec comme algorithme sous-jacent les trois algorithmes cités ci-dessus.

#### 4.1.2. Le plongement de phrases

Une alternative à la vectorisation des données textuelles en Bag Of Words ou en TF-IDF est le plongement de mots ou le plongement de phrases.

Le plongement vise à obtenir un vecteur de dimension inférieure fixe pour chaque document (ici question), qui traduise de manière intelligente la relation entre les mots ou entre les phrases, et leur sens, pour fournir aux algorithmes des features plus intéressantes.

Le plongement de mots, implémenté notamment par la librairie word2vec, utilise un espace de représentation des mots basé sur leur similarité. Il permet d'analyser les mots d'un document en fonction de leur contexte.

Le plongement de phrase (sentence embedding) est encore plus puissant puisqu'il se base non pas sur la similarité entre des mots mais entre des phrases. Le contexte est donc encore mieux pris en compte. Nous testons cette possibilité avec la version 4 de la librairie Universal Sentence Encoder, qui présente également l'avantage d'être déjà entraînée sur un ensemble très vaste de données. Elle transforme chaque document en un vecteur dense de 512 valeurs float.

## 4.2. Résultats

Les premiers résultats de l'optimisation de l'apprentissage supervisé montrent que cette approche est plus performante que l'apprentissage semi-supervisé, avec une hamming loss inférieure à 0,015. L'utilisation du plongement de phrases sur la concaténation du texte et du titre de la question donne les meilleurs résultats (figure 7). La classification par binary relevance sur une régression logistique est la plus efficace.

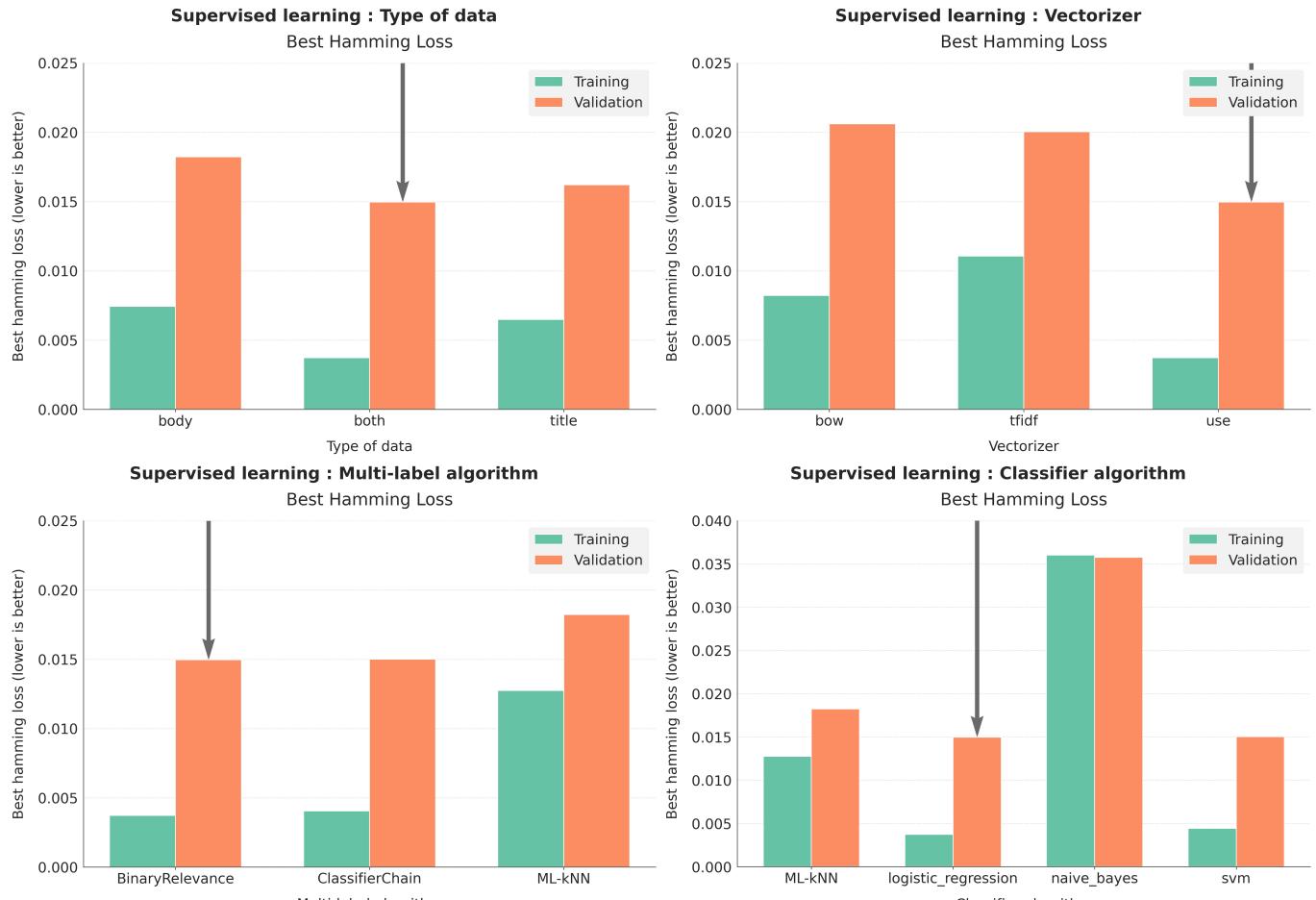


Figure 7 : résultats préliminaires de l'apprentissage supervisé

L'optimisation de la régression logistique, comme algorithme de base de la Binary Relevance, montre une hamming loss de validation minimale pour des valeurs de C comprises entre 0,008 et 0,020 (figure 8).

Je privilégie un C de 0,008, afin de limiter l'overfitting qui s'accroît à mesure que la valeur de C augmente. Limiter l'overfitting augmente la probabilité que le modèle généralise bien sur de nouvelles données.

La hamming loss de validation de ce modèle est de 0,0150, pour un accuracy score de 46,03% et un F1 score weighted de 67,67%.

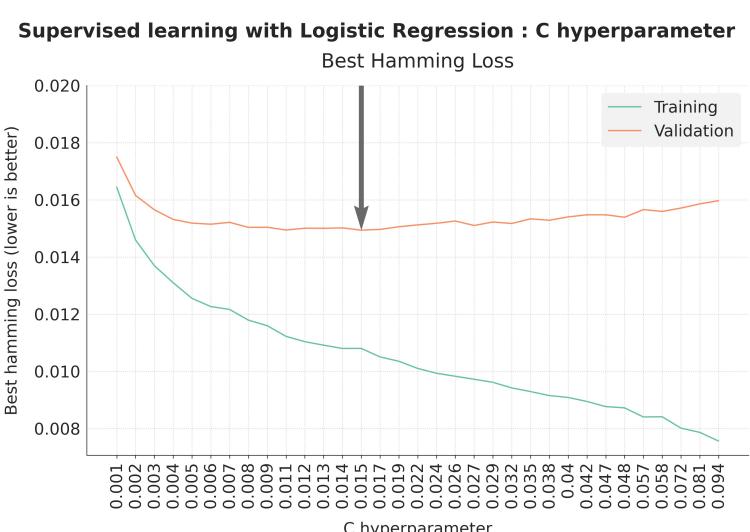


Figure 8 : sélection du C optimal

## 5. Choix du modèle final

### 5.1. Comparaison des performances

Je compare les performances des apprentissages semi-supervisé et supervisé afin de choisir le modèle final.

	Hamming loss	Accuracy score	F1 score micro	F1 score macro	F1 score weight
Modèle semi-supervisé	0,0183	32,51%	55,16%	52,34%	53,60%
Modèle supervisé	0,0150	46,03%	69,23%	61,50%	67,67%

Tableau 1 : comparaison des performances de validation

Le tableau 1 confirme que le modèle supervisé est plus performant, quelque soit la métrique utilisée.

### 5.2. Modèle choisi

Le modèle choisi est donc le modèle supervisé avec les caractéristiques suivantes :

- vectorisation de la concaténation du texte et du titre de la question avec un plongement de phrases (Universal Sentence Encoder, version 4) ;
- normalisation des données avec un Standard Scaler
- classification multi-labels avec une Binary Relevance sur une régression logistique avec C=0,008.

### 5.3. Evaluation de la performance de généralisation

Dès le début de ma recherche, j'avais isolé 20% des données sélectionnées aléatoirement en respectant la répartition des labels. J'utilise ces données de test jamais vues auparavant pour évaluer la performance de généralisation du modèle retenu.

J'obtiens des résultats au-dessus de mes performances de validation (hamming loss de 0,0137, accuracy score de 49,98% et F1 score weighted de 71,71%), ce qui laisse présager que le modèle généralise bien sur de nouvelles données et est assez efficace.

Je peux donc mettre en production ce modèle.

## Conclusion

Je développe une API basée sur le modèle choisi, dont j'anticipe qu'il prédira avec exactitude les 50 labels les plus fréquents dans la moitié des cas. En moyenne, le modèle devrait prédire correctement 64% des 50 labels les plus fréquents (valeur du Jaccard Similarity Score sur les données de test). Il devrait donc se tromper ou omettre un tiers des labels.

## Références

Text classification in Python - Towards Data Science - <https://towardsdatascience.com/text-classification-in-python-dd95d264c802?gi=af1fa49c1059>

A Comprehensive Guide to Understand and Implement Text Classification in Python - Analytics Vidhya - <https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/>

Topic Modeling with Gensim - Machine Learning Plus - [www.machinelearningplus.com/nlp/topic-modeling-gensim-python/](http://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/)

Deep dive into multi-label classification - Towards Data Science - <https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff>

Baselines and Bigrams: Simple, Good Sentiment and Topic Classification - Sida Wang and Christopher D. Manning - Association for Computational Linguistics - 2012

Full-text or Abstract ? Examining Topic Coherence Scores Using Latent Dirichlet Allocation - Shaheen Syed and Marco Spruit - Utrecht University - 2017