OpenClassrooms - P5
Catégorisez automatiquement des questions

1. Introduction

Dans le but d'aider la communauté StackOverflow, nous proposons une application permettant de suggérer des tags pertinents à l'utilisateur, en fonction de la question posée.

Deux approches sont envisagées pour l'entraînement des modèles: supervisée ou non.

En non supervisée, nous entraînons un modèle sur une base de référence, afin qu'il génère une liste de tags à proposer, adaptés aux sujets rencontrés sur StackOverflow. En soumettant une nouvelle question à ce modèle, il nous proposera les tags les plus pertinents pour cette question parmi les tags générés pendant l'entraînement.

En supervisé, on entraîne le modèle avec les tags existants dans la base de référence. En soumettant une nouvelle question à ce modèle il proposera parmi ces tags celui ou ceux les plus pertinents.

2. Récupération des données

Nous récupérons un échantillon des questions taggées via une requete SQL sur Query StackOverflow (https://data.stackexchange.com/stackoverflow/query/new)

Dans notre cas, nous avons récupéré un échantillon de 8069 questions récentes.

3. Prétraitement (preprocessing)

On utilise principalement la librairie NLTK

1. Suppression des tags HTML, on ne conserve que le “corps”, c'est-à-dire le texte à proprement parler. Nous utilisons la librairie BeautifulSoup à cet effet.

2. Suppression des apostrophes, retours à la ligne et espaces surnuméraires, au moyen d’expressions régulières

3. Passage en minuscule

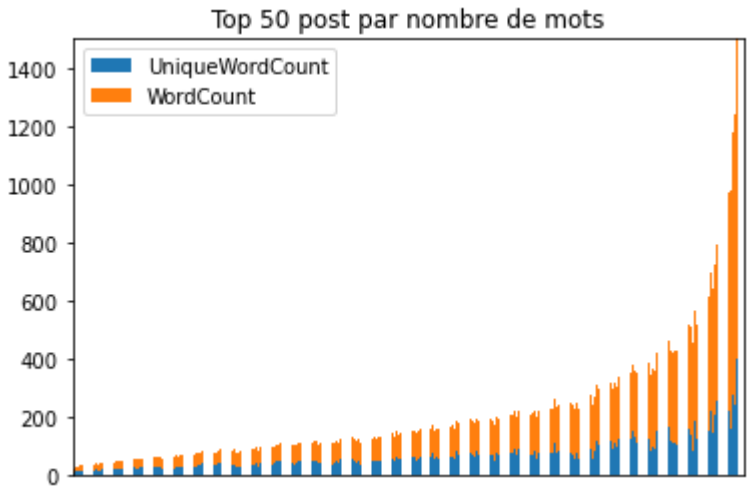
4. Suppression de tous les caractères autres que les lettres
   On pourrait éventuellement conserver les nombres inclus dans un mots (ex: n° de version)

5. Suppression des “stop_words” anglais
   Les mots fréquents dans la langue (ex: the), mais qui n’apportent pas de sens, en tout cas pas en NLP à ce niveau.

6. Stemming / Lemmatisation
   Le Stemming consiste à supprimer les préfixes et suffixes connus de chaque mot. La Lemmatisation transforme le mot en sa forme basique, la racine du mot en quelque sorte. Il demande une connaissance grammaticale de la langue.
   Dans notre cas, vu que les textes sont assez simples grammaticalement parlant, la lemmatisation est suffisante.

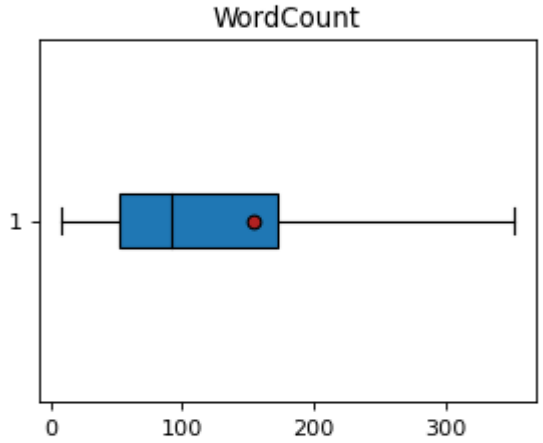
7. Fusion du titre et du corps des questions
   Nous ne ferons pas de différence entre le titre et le corps de la question, et donc nous fusionnons ces deux colonnes en une seule.

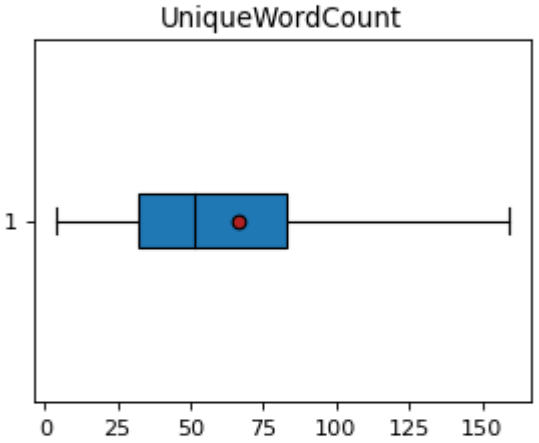
4. Analyses univariés

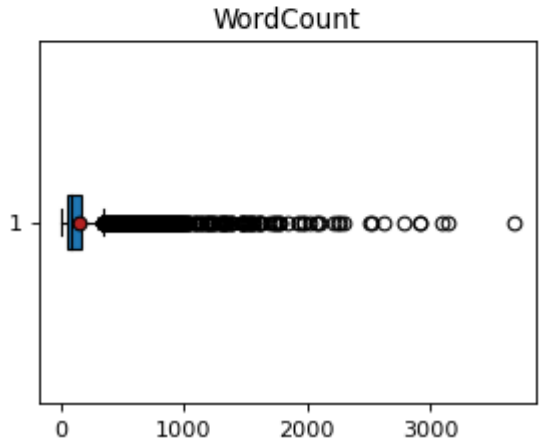


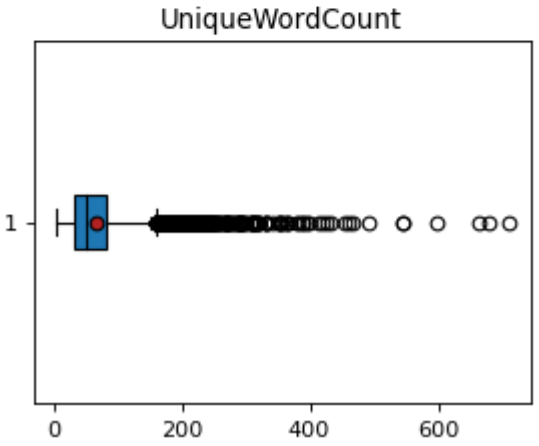
Description basique de WordCount
  min max 8 3679
  moyenne 154.045
  medianne 91.0
  mode 32
  variance empirique 47651.138
  ecart-type 218.291

Description basique de UniqueWordCount
  min max 4 710
  moyenne 66.136
  medianne 51.0
  mode 38
  variance empirique 2812.201
  ecart-type 53.03

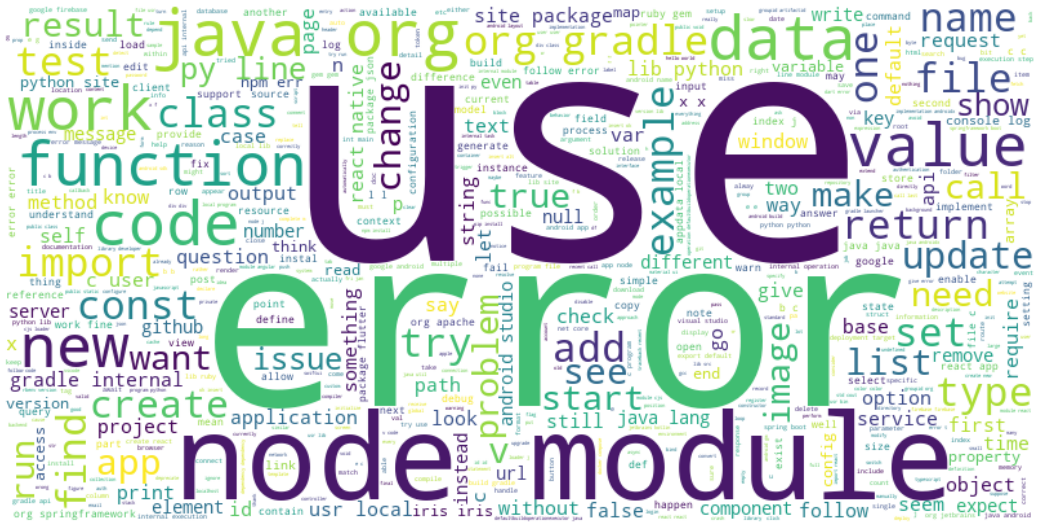




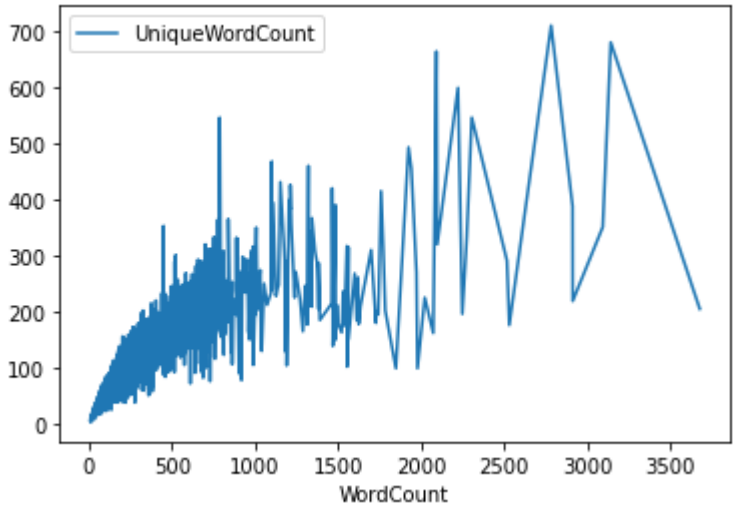




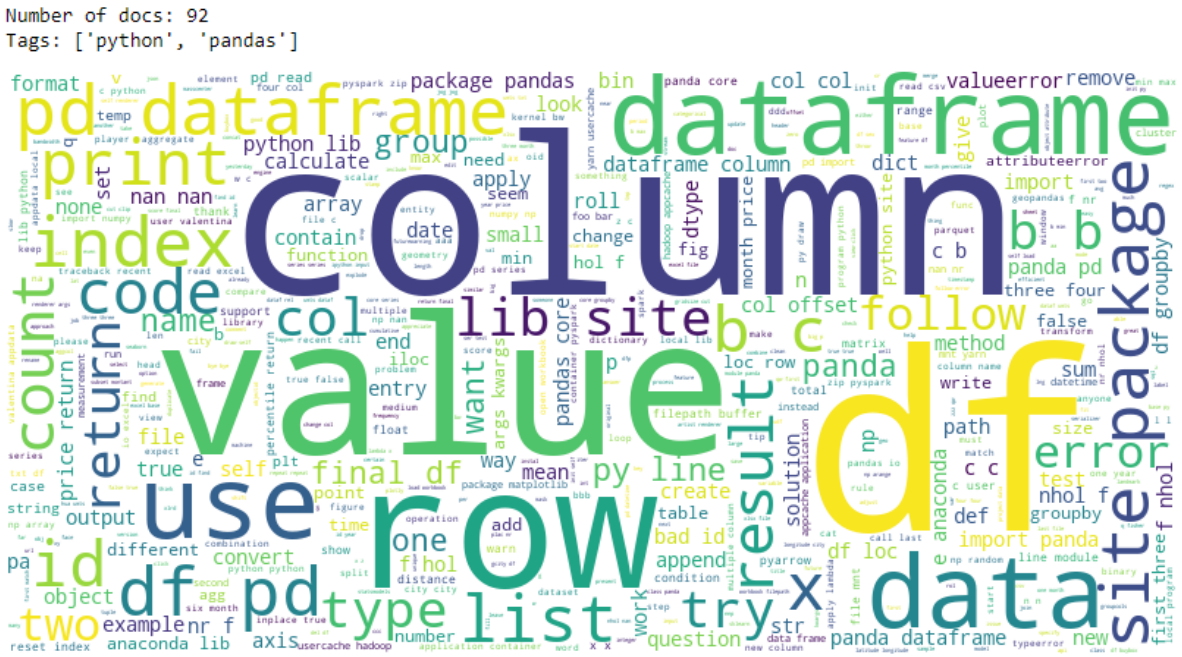
WordCloud



5. Analyses bivariées



Je ne vois pas d'analyse multivariée qui fasse réellement sens dans notre cas, mais nous pouvons utiliser le WordCloud pour afficher les mots les plus présents pour une sélection de tags. L'analyse est multivariée dans le sens ou on affiche les mots (une dimension) en fonction de N tags choisis (N dimensions d'étiquettes).
Ici les mots les plus fréquents dans les documents taggés par "python" ET "pandas".

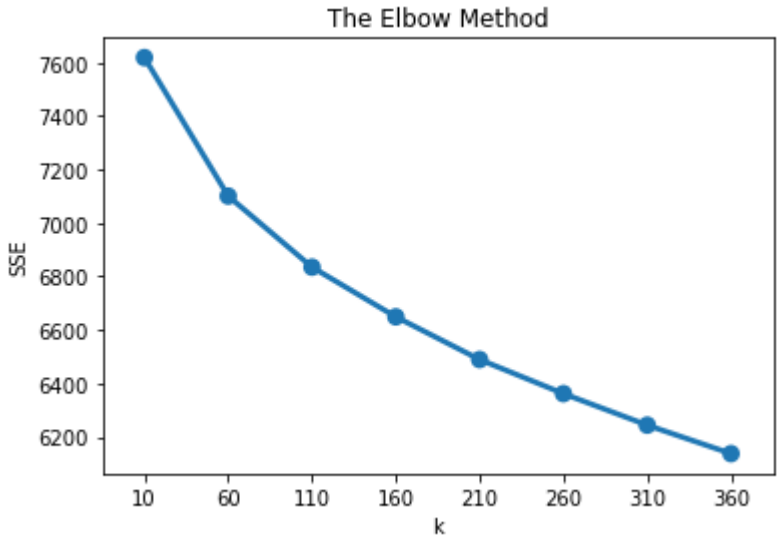


6. Approche non supervisée : Topic Modeling

KMeans et NMF

Le premier point consiste à avoir des Features exploitables.
On va ici utiliser le TF-IDF (term frequency-inverse document frequency) avec mots simples (unigram)

En première approche, nous utilisons la méthode du coude pour approximer le nombre de topic optimal avec KMeans.



Cette méthode nous recommande 40 topics. (après plusieurs tests sur des plages différentes)

Un KMeans avec 40 clusters (topics) donne un score “silhouette” de 0.018. Le fit se fait en 10.3 s

On essaye aussi l’algorithme NMF, avec le même nombre de topics. On obtient un score silhouette de 0.019, avec un fit en 4.1 s

Dans ces deux cas le score est très faible, nous tentons donc une autre approche.

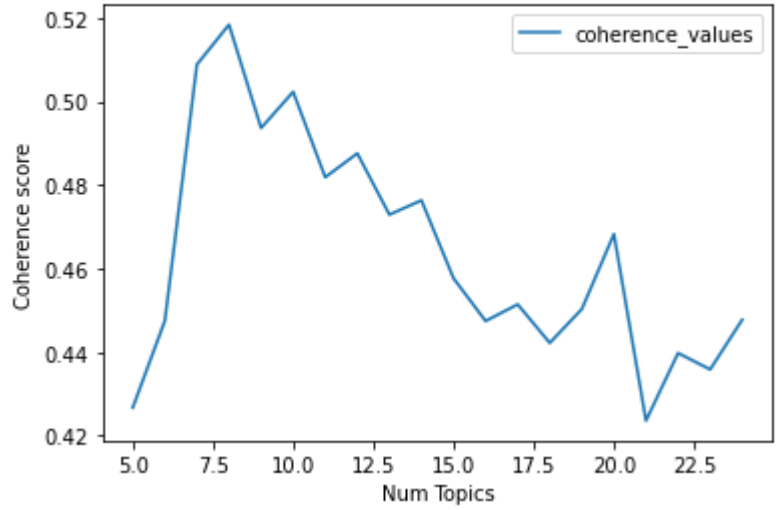
LDA et Mallet LDA

Les algorithmes LDA et Mallet LDA utilisent le “bag of words”. (~ TF sans IDF)

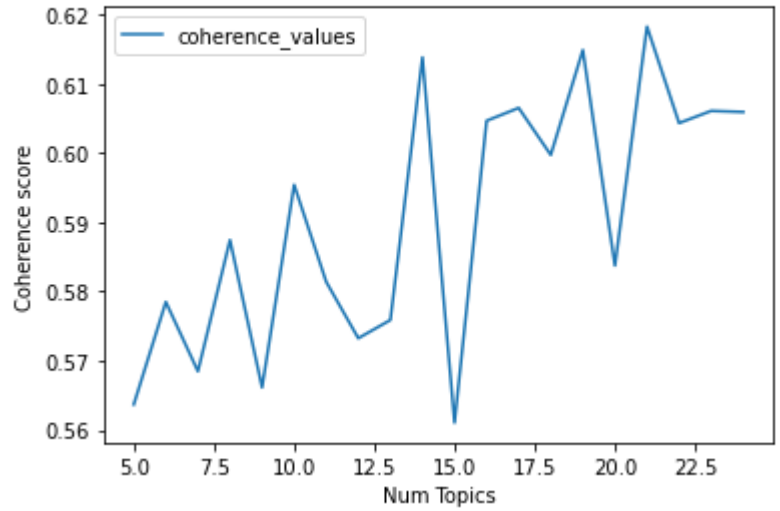
Nous testons une première fois ces modèles avec un nombre de topic arbitrairement fixé à 10. Le score utilisé ici est le score de Cohérence.
Avec LDA, 10 topics : 0.48
Avec Mallet LDA, 10 topics: 0.58

Nous testons différents nombres de topics pour LDA et Mallet LDA :

```
---------- LDA -------------
```

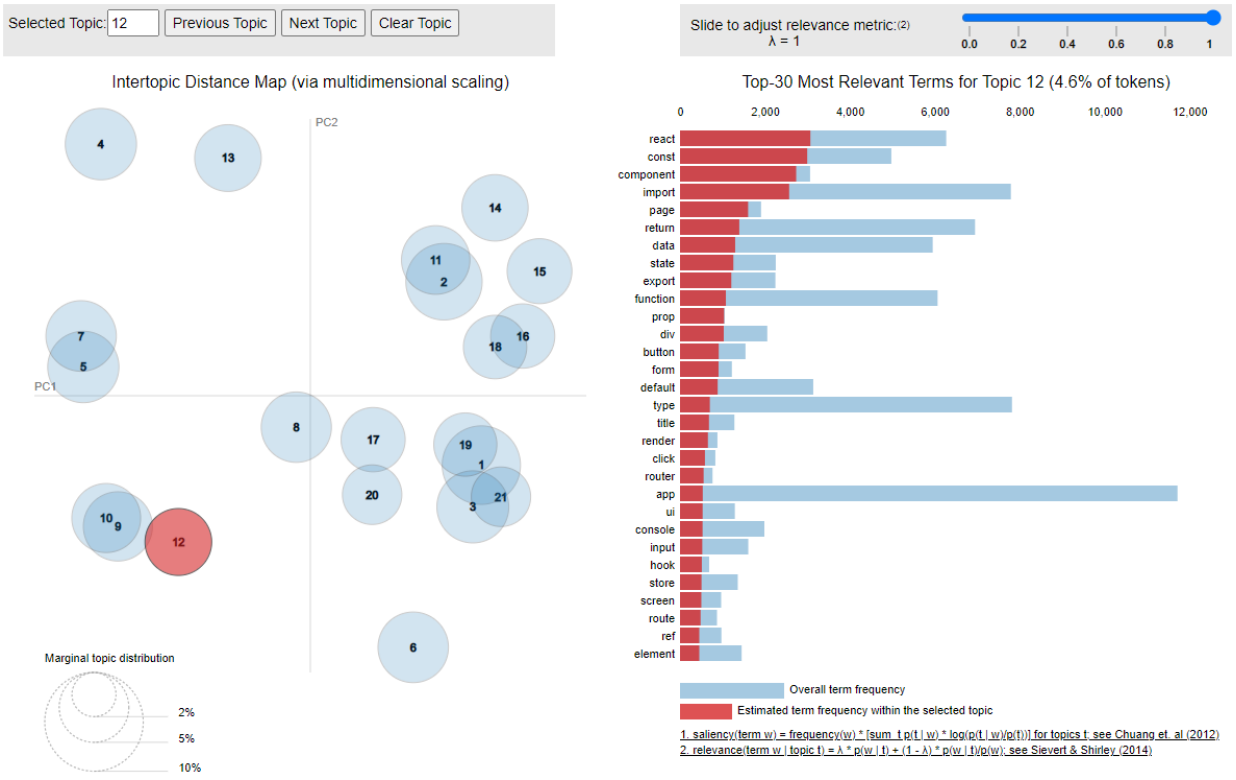



```
---------- Mallet LDA ------
```



Le meilleur score est obtenu pour Mallet LDA avec 21 topics (0.62)

Les topics suggérés sont consultables de diverses manières. J'utilise ici la librairie pyLDAvis, qui fonctionne bien avec la librairie Gemsim (utilisée pour LDA et Mallet LDA). Elle affiche interactivement pour chaque topic les mots les plus représentatifs.



7. Approche supervisée

TF-IDF

Ici aussi, nous vectorisons le texte en entrée avec le TF-IDF

MultiLabelBinarizer & reduction

Les tags à proposer sont potentiellement multiples, les étiquettes utilisées pour l'apprentissage étant elles-mêmes le plus souvent multiples. Pour gérer cela, nous utilisons le MultiLabelBinarizer (MLB)

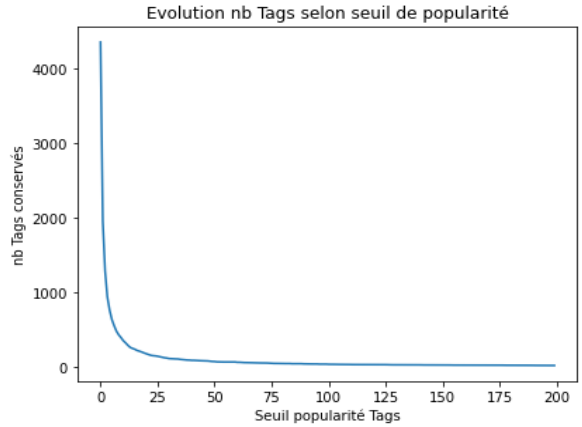
Appliqué à notre jeu de données, nous avons donc 4356 tags potentiels différents

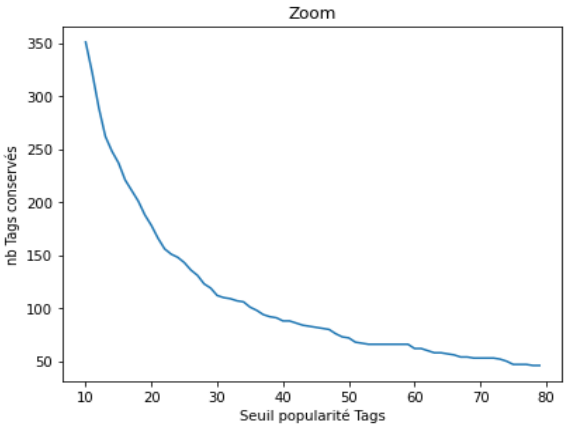
Malheureusement, avec le MLB, nous aurons énormément de classes très peu représentées sur l'ensemble du jeu de données, et même certaines qui ne seront pas du tout présentes après le découpage train/test pour cross validation. C'est un problème car les algorithmes que nous allons tester n'acceptent pas de classes cibles vides.

exemple: Après découpage train/test, Y_train ne contient aucun document taggé 'zstack'. La colonne 'zstack' est donc entièrement nulle (que des zéros), et l'apprentissage ne fonctionne pas du tout (message d'erreur sur colonne(s) avec une seule classe, à savoir 0 puisqu'il n'y a que ça: *ValueError: The number of classes has to be greater than one; got 1 class*).

Pour contourner ce problème, nous allons réduire le nombre de tags utilisés, en ne conservant que les plus populaires, et donc les plus représentatifs. Cela fait sens car notre auto tagger a surtout vocation à couvrir les cas génériques et pas les cas (très) particuliers. A noter qu'une base d'apprentissage plus conséquentes réduirait nécessairement le problème, mais sans le corriger totalement car on aurait certainement beaucoup plus de tags "rares".

Afin de réduire aux tags les plus "populaires", nous allons utiliser un seuil de popularité, qui correspond au nombre de fois qu'un tag est utilisé dans la base complète.



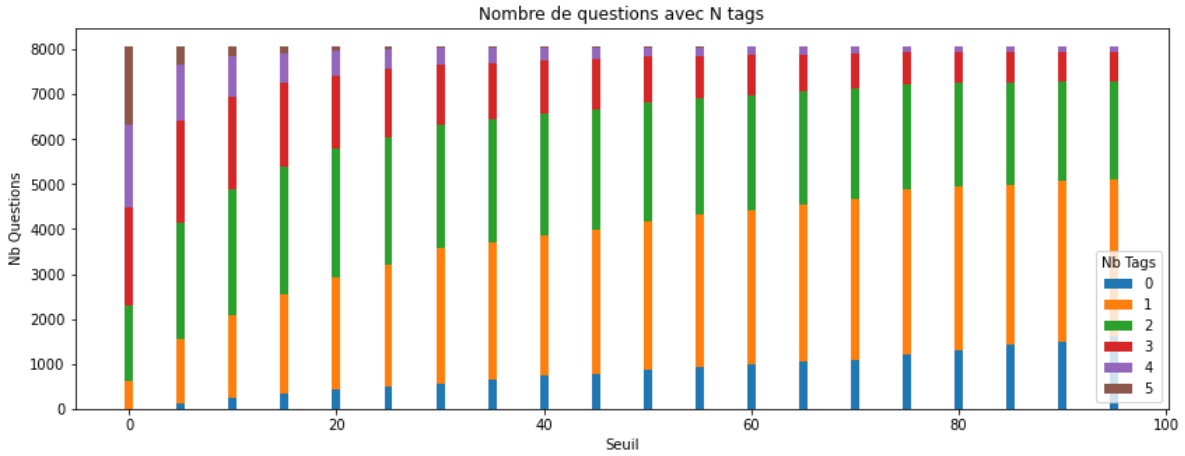


On observe que le nombre de tags décroît très rapidement.

Il faut cependant en conserver suffisamment pour que l'auto tagger reste pertinent. En effet, supprimer des tags appauvrit les données, jusqu'à vider les étiquettes de nombreuses questions. A l'extrême, lorsqu'on a supprimé tous les tags, le modèle est "parfait" puisqu'il prédit en permanence "aucun" tag, vu qu'il n'en voit jamais en apprentissage. C'est bien évidemment un très mauvais tagger dans ce cas, malgré son excellent score…

Pour illustrer ce problème, le graphique suivant montre le nombre de questions avec respectivement 0, 1, .. , 5 tags, en fonction du seuil.

On voit bien que le taggage s'appauvrit, et que le nombre de questions non taggées devient vite bien trop important.



Ainsi, nous allons donc tester différentes valeurs de seuil, en le souhaitant le plus bas possible, tout en étant suffisant pour que les algorithmes acceptent de tourner

Après tests, on fixe un seuil à 45, ce qui nous donne 82 tags conservés.

Test de différents modèles

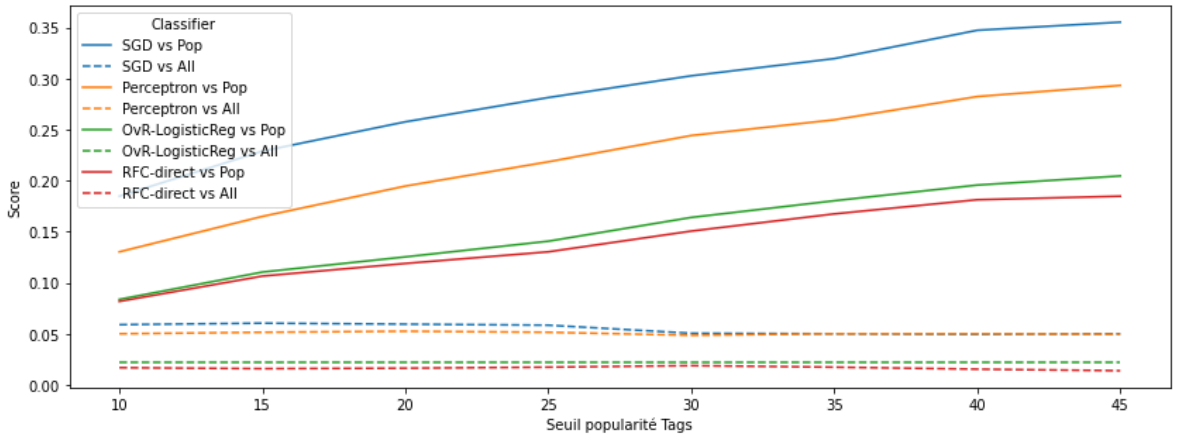
Nous avons des labels multiples. Or certains modèles ne fonctionnent pas en multi label, et doivent être adaptés. Différentes techniques sont envisageables:

- Le MultiOutputClassifier, qui va créer un modèle par label, donc 82 modèles qui vont chacun effectuer une classification binaire sur leur tag dédié.
- Le OneVsRestClassifier, similaire au précédent sauf que le modèle d'un label est fitté avec les autres labels.
- Certains modèles gèrent nativement le multi label, comme les modèles d'arbres, et certains autres appliquent eux-mêmes le OneVsRest quand ils sont en multi label.

J'ai testés plusieurs modèles, dont 4 ont retenu mon attention:

'SGD': MultiOutputClassifier(SGDClassifier()),

'Perceptron': MultiOutputClassifier(Perceptron()),
'OvR-LogisticReg': OneVsRestClassifier(LogisticRegression(), n_jobs=-1),
'RFC-direct': RandomForestClassifier(n_estimators=10, random_state=0, n_jobs=-1)



SGD (*stochastic gradient descent*) obtient les meilleurs scores, c'est donc ce modèle que l'on va utiliser

On observe aussi que les différents modèles testés obtiennent un meilleur score avec un seuil plus élevé, donc avec moins de tags.

Note: ici le seuil est descendu jusqu'à 10, car par un heureux hasard mon découpage train/test n'a pas créé de classe vide ici. Mais ce n'est pas assez robuste pour une vraie cross-validation, ou il y a plusieurs *folds* et donc découpages. Les tests d'hyper-paramétrage ont permis de fixer le seuil à 45 pour ne jamais avoir de cas problématique.

Hyper-paramétrage

On effectue donc un GridSearch avec cross-validation sur notre pipeline, en faisant varier différents hyper-paramètres.

```
GridSearchCV(estimator=Pipeline(steps=[('countvectorizer', CountVectorizer()),
                                       ('tfidftransformer', TfidfTransformer()),
                                       ('multioutputclassifier',
                                        MultiOutputClassifier(estimator=SGDClassifier()))]),
```

On obtient les résultats suivants:

```
Best score: 0.342
Best parameters set:
        countvectorizer__max_df: 0.5
        countvectorizer__max_features: 10000
        countvectorizer__ngram_range: (1, 2)
        multioutputclassifier__estimator__alpha: 1e-05
        multioutputclassifier__estimator__loss: 'log'
        multioutputclassifier__estimator__max_iter: 100
```

multioutputclassifier__estimator__penalty: 'l1'
tfidftransformer__norm: 'l2'
tfidftransformer__use_idf: True

Le meilleur score est donc obtenu avec des bi-grammes (ou tri-grammes, les scores étant très proches j'ai parfois eu l'un et parfois l'autre), et en utilisant IDF (du TF-IDF). Je passe sur les autres paramètres qui ne relèvent que de l'optimisation.

Scores precision/recall et f1-score pour la base test (sur les tags populaires)

	precision	recall	f1-score	support
micro avg	0.99	0.98	0.98	3285
macro avg	0.99	0.97	0.98	3285
weighted avg	0.99	0.98	0.98	3285
samples avg	0.89	0.89	0.89	3285

Conclusions

Approche non-supervisée : les tags à proposer doivent être interprétés, et ne correspondent pas nécessairement aux tags utilisés sur StackOverflow.

Approche supervisée : La réduction du nombre de tags induit une certaine perte de données, mais est nécessaire pour l'entraînement au vu des limitations des algorithmes utilisés.

En non-supervisé, chaque document est classé dans un unique topic, alors que l'approche supervisée est capable de prédire plusieurs tags pertinents par document. On pourrait s'approcher de ce fonctionnement en sélectionnant non plus le "meilleur" topic, mais tous ceux qui ont un score supérieur à un certain seuil.
Il est difficile d'établir un score pertinent en non supervisé, ou du moins qui permette de comparer avec l'approche supervisée. Il faudrait attribuer de "vrais" tags concordants avec ceux existants en se basant sur les topics générés afin d'avoir les moyens de comparer les deux approches.

En tout état de cause, dans notre cas, l'approche supervisée permet de générer des tags consistants avec ceux habituellement employés, et avec des résultats relativement satisfaisants.