Rapport de projet 6
Stack OverFlow

Janvier 2019 - Xavier Montamat

Sommaire

1. Introduction
    a. Description et analyse du projet
    b. Récupération des données
2. Pré traitement des données
    a. Transformations du texte
    b. Fréquences de mots
3. Réduction de dimensions
    a. Association de tags
    b. Réduction du nombre de features
4. Categorisation des posts
    a. Mise en place d'un modèle supervisé
    b. Résultats
    c. Modèle non supervisé
    d. Comparaison
5. Conclusion

1. Introduction

a. Description et analyse du projet

Ce projet concerne le forum d'entraide en programmation : **StackOverflow**. Il nous faudra récupérer un échantillon de questions (posts) posées sur le site et analyser leurs catégories (tags) afin de pouvoir les prédire avec un maximum de précision.

Chaque post contient un titre et une description de taille non limité. Ainsi qu'un ou plusieurs tags. Il nous faudra donc utiliser des algorithmes multi classes afin d'être capable de prédire plusieurs catégories. Les catégories étant saisies par l'utilisateur de façon libre, il est probable que nous retrouvions des catégories exceptionnelles qui seront trop anecdotiques pour être prédites. Il faudra donc les éliminer.

Il y aura potentiellement de nombreux mots différents, dont des extraits de code. Il sera nécessaire d'appliquer des transformation de texte afin de réduire le nombre de mots distincts et si possible de ne garder que les plus pertinents. Ici le sens des phrases et les mots du langage courant auront surement moins d'importance que les mots clés techniques.

b. Récupération des données

La première étape est de récupérer nos données sur data.stackexchange.com/stackoverflow. Pour éviter que la requête ne soit refusée et que nos modèles ne prennent trop de temps, nous souhaitons récupérer entre 25 et 50K lignes.

Nous n'avons besoin que de peu de champs :

- Le titre du post
- Le corps du post
- Les tags associés

Nous ne prenons que les posts de type 'question' et non les réponses.

Il faut ensuite vérifier qu'aucun des champs ne soit vide. Également que chaque post ai un minimum de mots, afin de pouvoir l'analyser (100 caractères). Et afin de réduire le nombre de résultats nous ne prendrons que ceux ayant au moins une réponse. Cela devrait favoriser les posts correctement écrits et tagués. Sans pour autant favoriser les thèmes populaires qui sont davantage consultés.

Nous obtenons finalement un csv **43K posts.**

2. Pré traitement des données

a. Transformations du texte

Nous allons maintenant traiter notre texte de façon à ce qu'il soit exploitable par nos algorithmes d'apprentissage.

Première constatation, notre corps de post est sous forme HTML.

Prenons un exemple de post :

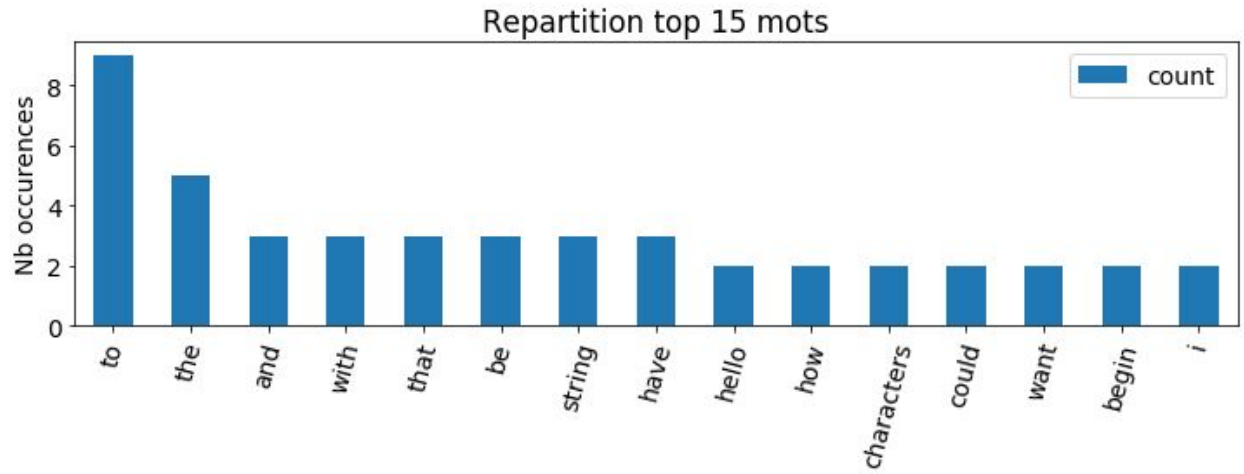
```
"<p>I'm new to Python, but want to figure out how to take a string and swap pairs of characters around. Le
t's say we have the string \n'HELLO__WORLD' and want to switch the HE in HELLO with __. So that the string
now looks like '__LLOHEWORLD' How could that be done? Does it have anything to do with .pop and .append? O
r perhaps if, elif, else functions could be used? Maybe to begin with, .index is needed to find the charac
ters that the user specifies need to be swapped?</p>\n\n<p>I honestly have no idea really where to begin.
</p>\n"
```

Nous ne voulons garder que du texte, sans les tags html ni leur contenu. La librairie Beautiful soup permet de faire cela facilement. Nous allons donc utiliser son parser HTML et la méthode get_text()

```
"I'm new to Python, but want to figure out how to take a string and swap pairs of characters around. Let's
say we have the string \n'HELLO__WORLD' and want to switch the HE in HELLO with __. So that the string now
looks like '__LLOHEWORLD' How could that be done? Does it have anything to do with .pop and .append? Or pe
rhaps if, elif, else functions could be used? Maybe to begin with, .index is needed to find the characters
that the user specifies need to be swapped?\nI honestly have no idea really where to begin. \n"
```

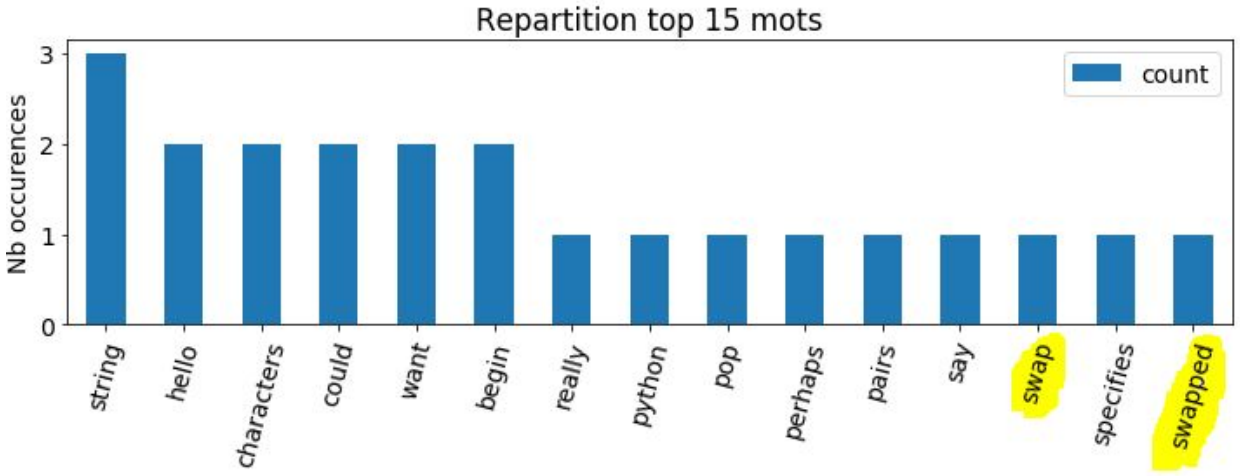
Il faut également retirer la ponctuation et autre caractères spéciaux, et transformer les majuscules en minuscules. Nous transformons ensuite notre texte en liste de mots. De manière à pouvoir traiter chaque mot séparément.

Quelques statistiques sur nos top 15 mots les plus présents dans notre exemple



Comme l'on pourrait s'y attendre, les mots les plus communs sont des prépositions telles que 'to', 'the', 'and'. N'étant pas pertinente pour déduire le contenu du texte, nous allons les retirer grâce à la librairie NLTK. Il suffit de télécharger la liste anglaise de stopwords. Puis de retirer ces mots de notre liste.

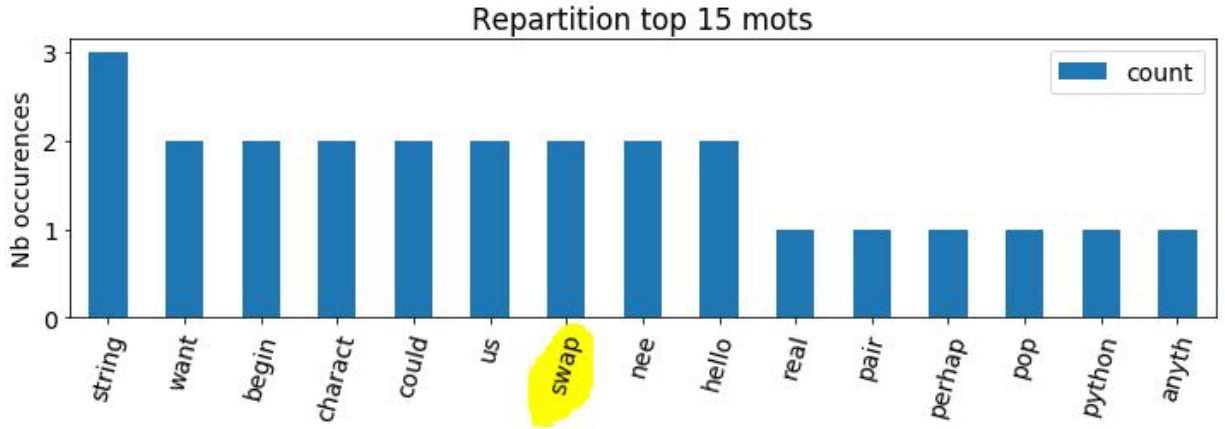
```
nltk.download('stopwords')
from nltk.corpus import stopwords
df_words = df_words.loc[~df_words.index.isin(stopwords.words("english"))]
```



Notre mot le plus utilisé est désormais 'string', et les mots restants sont globalement plus intéressants. Cependant l'on remarque que les mots swap et swapped sont comptés séparéments alors qu'ils véhiculent la même idée.

Nous allons donc utiliser un stemmer pour ne garder que la racine des mots, et ainsi associer entre eux les mots ayant une racine commune. Ce qui réduira notre nombre de dimensions final. Après avoir testé les Lancaster & Porter Stemmer, et WorldNet Lemmatizer, nous nous sommes arrêté sur le Lancaster Stemmer, avec lequel nous avions les meilleurs résultats.

Voici les résultats après avoir appliqué l'algorithme.



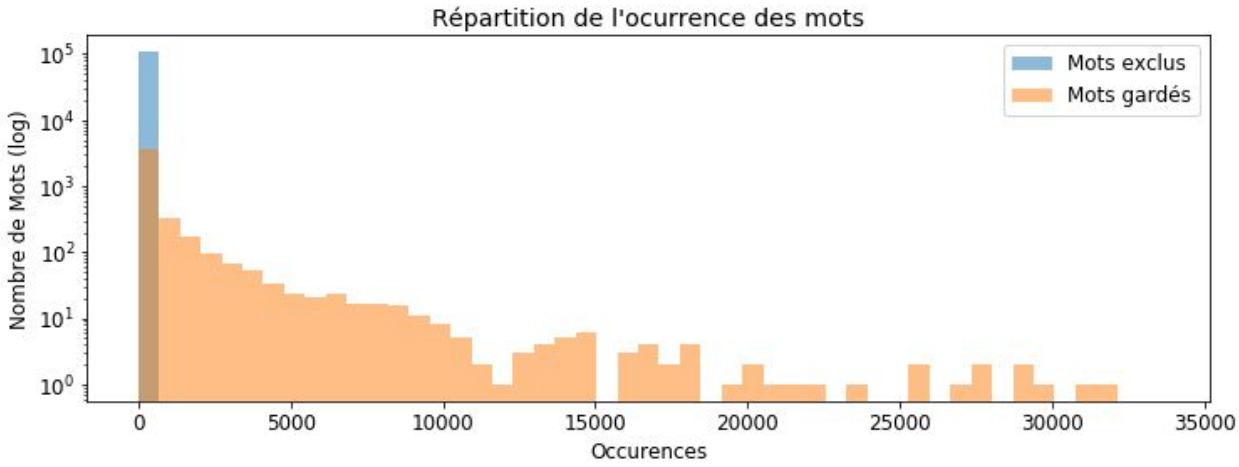
b. Fréquences de mots

Sur nos 43K posts nous avons un corpus de 4.6 millions de mots en unifiant titre + corps. Ce qui nous donne une moyenne de 108 mots par post. **115K mots** sont uniques (après Lancaster stemmer), dont 80% apparaissent moins de 5 fois ! Il y a donc encore trop de dimensions dans l'état.

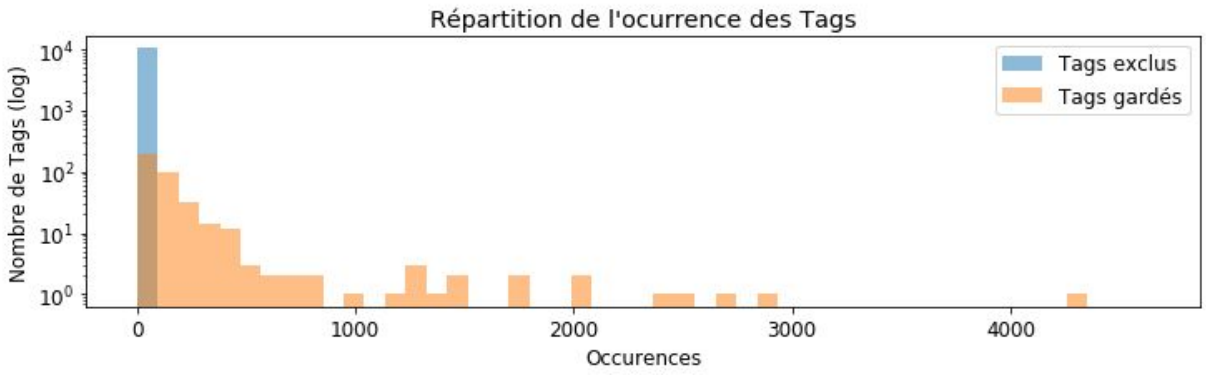
Pour réduire nos dimensions, nous ne sélectionnons que les mots apparaissant au moins dans **1 post sur 1000** (soit environ 43 fois sur notre dataset). Cela devrait éliminer de nombreux mots inexistants ou très rares, qui seront inexploitables.

Et en effet, on exclut ainsi 95.8% des mots uniques. Il y a **4708 mots uniques** restants.

Mais cela ne représente qu'une perte de 9% sur nos 4.6M de mots du corpus. Soit 4.1M de mots restants.



Même démarche pour les tags. Il y a 133K tags sur notre corpus, divisible en **11k tags uniques**. En ne gardant que les tags présents dans **1 post sur 1000** minimum , nous écartons 96.5% des tags uniques, en gardant 68.3% du corpus total de tags. Soit 91K tags restants dont **379 distincts**.



3. Réduction de dimensions

a. Association de tags

Malgré les étapes de pré traitements, nous avons toujours **379 tags distincts** à prédire, ce qui semble encore beaucoup. En observant plus en détail nos tags restant, on s'aperçoit rapidement que de nombreux tags reflètent des idées similaires.

Par exemple :

'asp.net',
'asp.net-core',
'asp.net-mvc',
'asp.net-mvc-3',
'asp.net-mvc-4',
'asp.net-web-api',

Ou encore :

'visual-studio',
'visual-studio-2008',
'visual-studio-2010',
'visual-studio-2012',
'visual-studio-2013',
'visual-studio-2015',

L'idéal serait d'**unifier les tags** les plus corrélés, de manière à ne garder que le tag le plus commun. Cela afin de réduire le nombre de tags distincts à prédire, sans pour autant supprimer plus de tags.

Notre cible arbitraire est d'arriver en dessous des 100 tags. Nous réalisons donc une **matrice de corrélation** entre tous nos tags. Puis les classons par score pour obtenir une liste des plus corrélés. Nous appliquons également un malus selon la fréquence de leur apparition, de manière à prioriser la réduction des tags les moins utilisés, au profit de ceux les plus utilisés.

Il nous suffit ensuite de réaliser des associations parent-enfants, où le parent est toujours le tag étant le plus fréquemment utilisé des deux. Grâce à cela nous réussissons à associer 284 enfants à 72 parents, plus 23 orphelins. Ce qui nous laisse **95 tags** distincts à prédire. Chaque enfant étant remplacé par son proche parent.

Exemple ci dessous de remplacement :

	replace_with
visual-studio	NaN
visual-studio-2008	visual-studio
visual-studio-2010	NaN
visual-studio-2012	visual-studio
visual-studio-2013	visual-studio
visual-studio-2015	visual-studio

	replace_with
asp.net	NaN
asp.net-core	visual-studio
asp.net-mvc	NaN
asp.net-mvc-3	asp.net-mvc
asp.net-mvc-4	asp.net-mvc
asp.net-web-api	asp.net-mvc

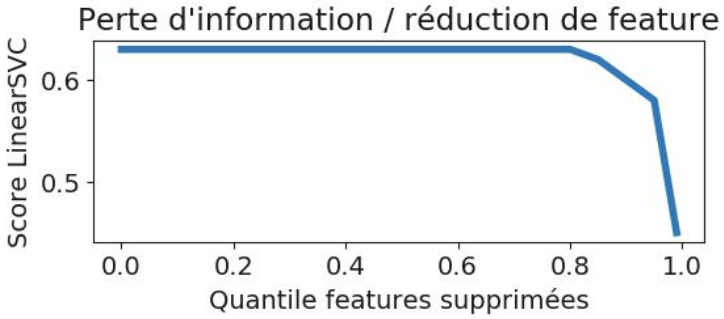
b. Réduction du nombre de features

Après l'étape de pré-traitement, il nous reste **4.7K mots uniques** (features) dans notre corpus. Bien que cela soit exploitable, il serait intéressant de réduire ce nombre de features, pour améliorer le temps de calcul en éliminant les mots "polluants". On peut penser par exemple aux mots courants, qui pour la plupart ne vont pas nous aider à définir nos catégories, contrairement aux mots techniques.

Nous avons avons d'abord tenté d'entraîner notre algorithme sur les données telles qu'elles, puis choisi le meilleur algorithme (LinearSVC dans notre cas). Nous notons le f1-score obtenu comme base de comparaison. LinearSVC calcul un **coefficient d'importance** pour chaque feature, et pour chaque tag. Nous allons nous baser sur ce coefficient pour sélectionner nos mots les plus pertinents.

Nous avons au total 95 tags. Nous appliquons donc un modèle de LinearSVC pour chacun de ces tags, et récupérons la liste des features ayant le meilleur coefficient absolu (les plus utiles pour prédire ce tag). On note également le score de précision f1 obtenu pour ce tag, afin de savoir si la prédiction est plus ou bonne pour ce tag. (les features d'un tag imprévisible vont moins nous intéresser).

A partir de cette liste, nous pouvons sélectionner plus ou moins de features. Après quelques tests, l'on constate qu'en **réduisant de 80%** nos features, il n'y a pas de perte de précision sur notre algorithme final ! (score f1). Il y aurait une baisse de 0.01 à 85%, -0.03 à 90% et -0.005 à 95%.



Bien que l'on pourrait supprimer 90% des features, nous préférons garder le meilleur score possible, et donc de réduire nos features de 80% seulement, soit **942 features restantes**, ce qui est déjà peu !

A droite, un extrait des 10 mots jugés les plus utiles pour catégoriser nos posts.

	NB tags influencés	Score moyen
python	14	5.683
php	7	5.102
android	12	4.977
c#	11	4.783
cout	1	4.700
c++	4	4.580
typescrib	3	4.560
def	12	4.384
jav	12	4.358
perl	4	3.916

Passons maintenant à la catégorisation.

4. Catégorisation des posts

Nous avons désormais un dataset bien épuré de 43K posts, composés de seulement 942 mots différents. Et catégorisés en 95 tags (plusieurs tags pouvant être associés à un même post). Nous allons entraîner un modèle supervisé, puis un modèle non supervisé, et comparer nos résultats.

Pour transformer nos mots en features, nous utilisons un **TFIDF Vectorizer**. Comme nous avons un titre et un corps pour chaque post, nous devons unifier les deux. Le titre étant généralement plus important pour définir le contexte, nous appliquons un poids de 2 pour les mots du titre, contre 1 pour le corps du post.

Pour les tags, pas d'action spéciale, nous nous contentons de les transformer en targets de manière binaire.

a. Mise en place d'un modèle supervisé

Nous divisons notre dataset en un set d'entraînement et un set de test, en les séparant sur des proportions 25/75.

Notre classification étant un problème de **type multilabels**, nous devons utiliser une méthode adaptée afin d'entraîner notre modèle. Nous testerons les transformations suivante :

- **Binary Relevance** : Qui va entraîner un modèle à partir de nos features pour chaque target, de manière indépendante.
- **Chain Classifier** : Va également prédire une feature à la fois, mais en incluant les targets déjà prédites en tant que features. Cela peut être utile pour trouver des corrélations entre nos targets.

Nous n'utiliserons pas la transformation de Label Powerset, car trop coûteuse pour 95 targets.

Pour le choix de l'algorithme de classification supervisé, nous avons testé KNeighborsClassifier, RandomForests, GaussianNB, LinearDiscriminantAnalysis, et LinearSVC. **LinearSVC** est l'algorithme ayant obtenu les meilleurs résultats, suivi par LDA.

La comparaison entre Binary Relevance et Classifier Chain montre des résultats très similaires, mais légèrement inférieurs pour le Classifier Chain. Il n'y a probablement pas suffisamment de corrélation entre nos targets pour que la méthode soit performante. A noter que nous n'avons pas tenté d'optimiser l'ordre des targets.

b. Résultats

Après avoir entraîné notre modèle de LinearSVC (C=1) sur 32K posts, puis prédit les tags des 10K posts restants, nous calculons les scores obtenus.

Rapport de classification :

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1008110
1	0.81	0.52	0.63	19220

Sur les 19220 tags à prédire de notre set de test, environ **1 tag sur 2** ont été prédits correctements, avec une **précision de 0.81**, ce qui montre que notre algorithme se trompe assez peu, mais omet tout le même la moitié des tags.

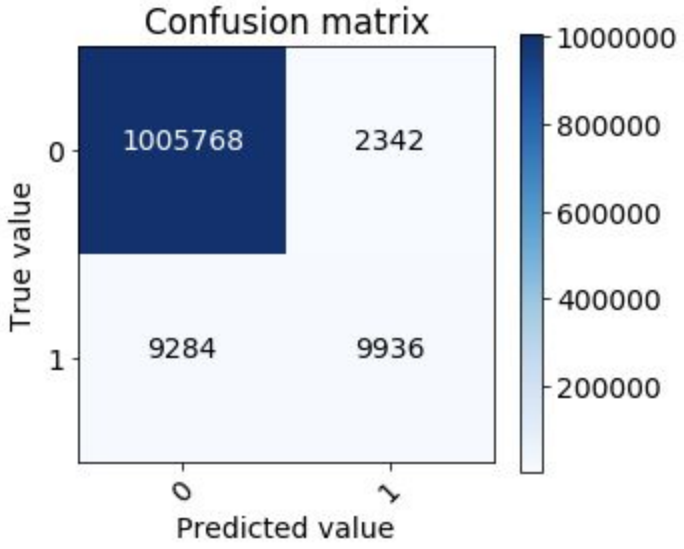
Voir la matrice de confusion ci contre

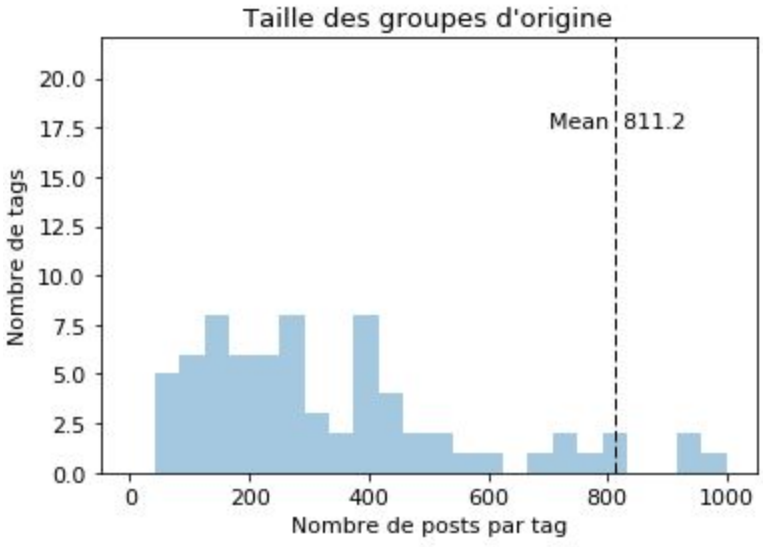
Autre métriques :

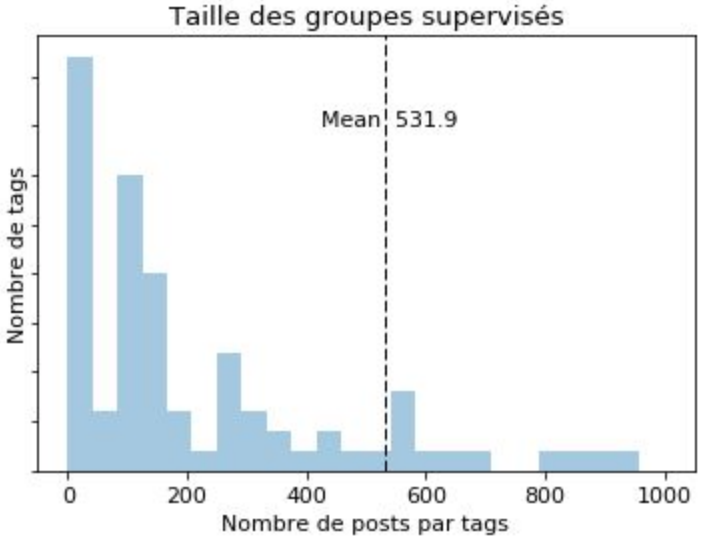
- Score de similarité Jaccard : 0.54
- Adjusted Rand Score : 0.62

Pour les tags mal prédits, ils sont souvent discutables puisque les tags indiqués par l'utilisateur ne sont pas toujours exhaustifs, ni forcément les plus pertinents.

En tentant de regarder de plus près les tags omis, on s'aperçoit que beaucoup de catégories de petite taille ne sont jamais prédites (0 membres). Voir répartition ci dessous.







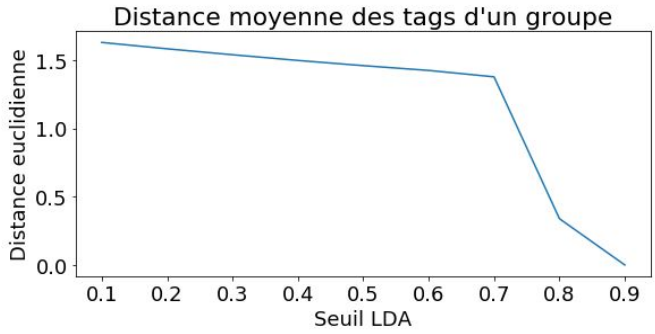
c. Modèle non supervisé

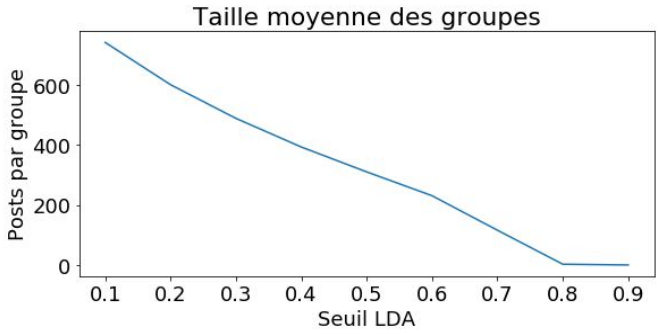
Pour notre modèle non supervisé, nous utiliserons **LDA**, (LatentDirichletAllocation)

Le nombre de composants sera le même que notre nombre de tags distincts, soit 95.

Nous obtenons donc **95 groupes**, pour lesquels chaque post possède une **probabilité d'appartenance**. En faisant varier ce seuil, nous pouvons donc inclure plus ou moins de posts à nos groupes.

Plus l'on augmente le seuil, plus les posts groupés seront proches, mais moins il y aura de posts par groupe. Voici ci dessous les courbes montrant les effet de variation du seuil de 0.1 à 0.9





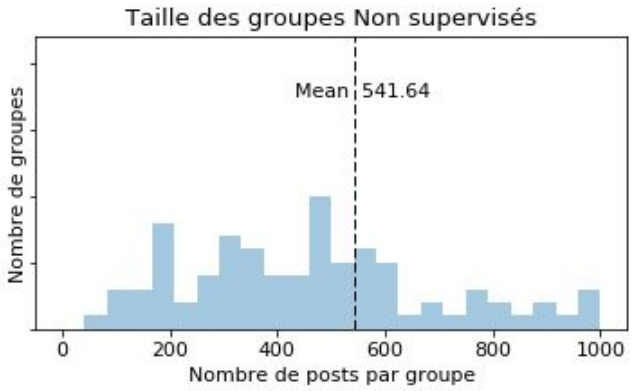
A gauche, nous avons un calcul de distance moyenne euclidienne inter-cluster, basée sur nos tags d'origine. A droite, la taille moyenne des groupes. Il est intéressant de voir que nous pouvons facilement adapter les groupes en fonction du besoin (davantage d'associations ou bien davantage de précision).

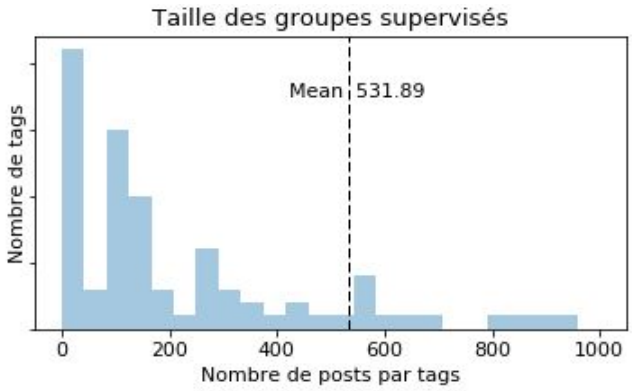
Dans notre cas, nous tentons de nous rapprocher des associations par tags déjà existantes. Le nombre moyen de post par tag est de 811 pour les tags réels. Et de seulement 531 pour les tags prédits avec notre méthode supervisée LinearSVC présentée plus haut.

Nous tentons donc de nous rapprocher de la moyenne prédite par LinearSVC, en choisissant un **seuil de 0.25**, avec lequel nous obtenons **541 posts par groupe** en moyenne. La distance moyenne obtenue dans nos groupe est de **1.56**

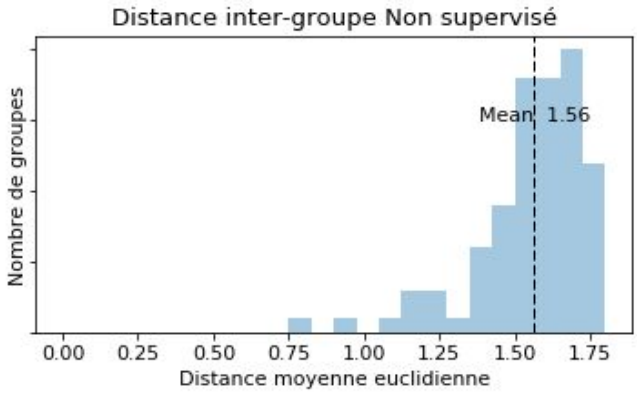
d. Comparaison

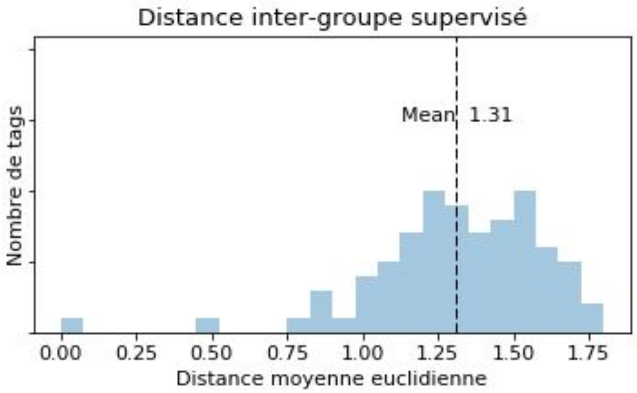
Nous tentons à présent de comparer nos modèles supervisé et non supervisé.





Pour ce qui est de répartition des posts dans les groupes, le modèle non supervisé répartit de manière plus équilibré. On peut voir que pour une même taille moyenne, la méthode supervisée a beaucoup de tags vides! Ce qui peut être gênant dans certains cas de figure.





Note : La distance moyenne 'maximum' est de 1.8, ce qui représente la distance moyenne de posts répartis aléatoirements.

En terme de distance moyenne (nous avons écarté les groupes vides pour la méthode supervisé), le LDA garde un nette désavantage, comme on pouvait s'y attendre puisqu'il n'était pas optimisé pour cette tâche. Cependant pour certains groupes, on peut voir que les posts sont tout de même très proches et donc très similaires à des tags existants.

La méthode supervisée créé des groupes plus unifiés encore qu'ils ne le sont en réalité, puisque nos vrai tags ont une distance moyenne réelle de 1.41. Cela s'explique par des associations de tags plus aléatoires dans la réalité que dans notre prédiction.

5. Conclusion

Les résultats obtenus sur ce projets sont assez bons. La méthode supervisée est efficace pour prédire une partie des tags et pourrait être utilisée dans un contexte réel. Soit pour des tags passé manquant, soit pour assister l'utilisateur en temps réel. A condition qu'il puisse toujours ajouter ses propres tags. Pour rappel nous obtenions une précision de 0.81 pour un recall de 0.52.

La méthode non supervisée quand à elle donne des associations plutôts réalistes. Bien qu'ils soient naturellement moins proches des catégories réelles, ils recréés tout de même des associations pertinentes. Le fait d'avoir gardé une majorité de mots techniques a probablement pu aider dans ce sens.

Le traitement de texte et réduction des dimensions ont été particulièrement efficaces et primordiaux sur ce projet. Ils ont permis de réduire considérablement les temps de traitement, sans perdre de précision sur nos résultats.

Améliorations

Pour la suite, il serait sûrement pertinent d'améliorer le système de prédictions en le couplant à un modèle non binaire. L'on pourrait ainsi proposer à l'utilisateur un top 10 des tags les plus probables. De manière à lui apporter des idées tout en le laissant décider de la pertinence de chaque tag. Cela devrait améliorer à terme la cohérence entre les tags utilisateurs, et ainsi renforcer la précision de nouveaux modèles de prédictions.

Nous pourrions également ajouter un historique des tags déjà utilisés par l'utilisateur. Puisqu'une même personne a davantage de chance de reposter sur les mêmes catégories. Cela devrait améliorer la précision pour les utilisateurs vétérans.

Liens

Lien GitHub du projet

https://github.com/xmontamat/OCR_Project6_StackOverFlow

Lien de l'API en ligne

http://xmontamat.pythonanywhere.com/