CentraleSupélec

OPENCLASSROOMS

Recommandation de Tags

Contenu

Introduction

A partir d'une API du site Stack Overflow, l'objectif de ce projet est de mettre en place un modèle de prédiction de tags pour une question posée. Le but est d'aider les membres sur le site Stack Overflow à mieux classifier leurs questions et ainsi avoir des réponses potentiellement plus pertinentes.

Dans un 1er temps, nous allons récupérer des datasets, explorer leur contenu et faire du nettoyage. Par la suite une approche non supervisée sera faite afin de trouver les sujets principaux de la question et ainsi essayer de prédire les tags censés.

Dans un second temps, une approche supervisée sera faite avec un fine tuning du meilleur modèle. L'API vous sera présentée avec des critiques sur la prédiction. Pour finir, des ouvertures à l'amélioration seront proposées

Préparation des données

Récupération des dataset

L'API de Stack Overflow nous permet via une requête SQL de récupérer diverses données publiques (sur les post, utilisateurs, tags, etc.). Dans notre cas, nous sommes intéressés par le titre, le contenu de la question ainsi que les tags. Pour avoir un dataset d'entrainement et de test pour la phase supervisée, on va prendre les questions au hasard. Les 50 000 premiers sujets seront pour l'entrainement et les 50 000 suivant pour les tests. L'Id de la question est aussi téléchargé pour s'assurer qu'il n'y ait pas de doublons. Les requêtes sont donc :

```sql
SELECT Id, Title, Tags, Body
FROM Posts
WHERE PostTypeId = 1
AND Score > 3
ORDER BY RAND()
OFFSET 0 ROWS FETCH NEXT 50000 ROWS
ONLY
```

```sql
SELECT Id, Title, Tags, Body
FROM Posts
WHERE PostTypeId = 1
AND Score > 3
ORDER BY RAND()
OFFSET 50000 ROWS FETCH NEXT 50000
ROWS ONLY
```

PostTypeId est mis à 1 pour n'avoir que les questions. Pour s'assurer de la qualité du dataset, seul les questions avec un score supérieur à 3 est pris. Cela permet de s'assurer que l'auteur a fait des efforts sur le contenu et les tags.

Pré-traitement du dataset

Après avoir vérifié qu'il n'y a pas d'ID en doublons dans les 2 datasets. Une exploration des features a été faite. Concernant le titre, il n'y a de pas besoin de traitements particuliers. Il a juste été fusionné avec le body pour avec un corpus constitué que d'une seul feature.

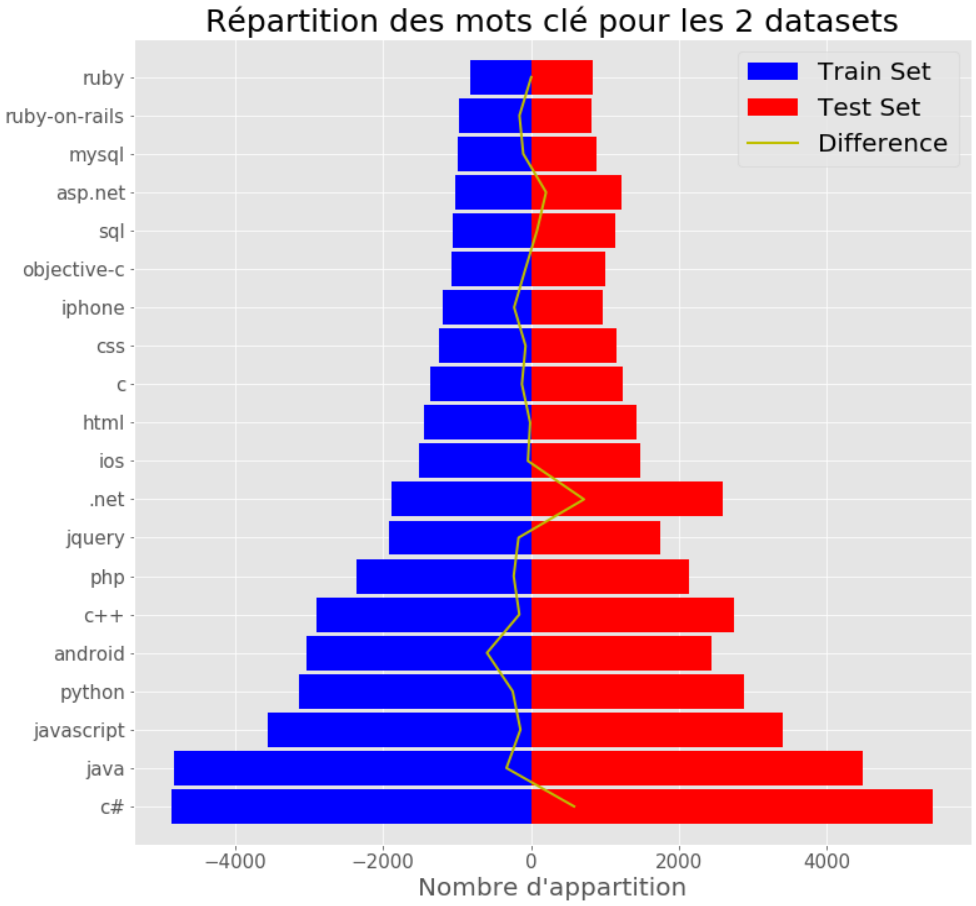
Pour les tags, une regexp a permis de convertir cette feature en une liste de liste. Cette liste a ensuite été stockée avec pickle. Une seconde matrice est aussi générée pour la partie supervisée avec un MultiLabelBinarizer (similaire au One Hot Encoder mais avec 1 pour chaque tags).

Pour le body, le code a été supprimé ainsi que les diverses balises. Cela permet de n'avoir un corpus ne conservant que du "vrai texte". Les balises ne sont là que pour le rendu sur le site. Concernant le code, j'ai pris la décision de le supprimer car la majorité des mots sont uniques (nom des variables) ou communs dans tous les langages (for, while, break, try, return, …). Pour finir, un compte de chaque mot a été fait. Le top 100 par fréquence a été rajouté aux StopWords (English) de NLTK pour supprimer les mots plus courants spécifiques aux Sujets de Stack Overflow.

Exploration

Après la phase de nettoyage, une exploration des tags a été faite. Sur 50 000 posts, plusieurs milliers de tags sont présents. Afin de pouvoir faire un classifieur par la suite, je n'ai gardé que les tags avec plus de 25 apparitions dans les trains set (cela concerne 773 tags). En effet, les tags avec peu d'apparitions dans le train set peuvent être inexistants dans le test set ou dans de plus grandes proportions. De plus, on aura besoin d'avoir assez de fois chaque tag pour entrainer le classifieur.

Par la suite, tous les posts n'ayant plus de tags ont été supprimés. Cela représente que très peu de posts (~1500 sur le train set et ~3000 sur le test set). Afin de s'assurer de la balance des 2 datasets, on peut regarder la fréquence d'apparition des tags avec une pyramide.



On a quelques différences entre les 2 datasets mais hormis avec .net, le dataset est plutôt balance sur les 20 principaux tags.

Préparations des Matrices

La matrice Term Frequency

On a un seul corpus de 48357 posts. Celui-ci été utilisé pour générer la matrice de Term-Frequency. Cette matrice est très sparse avec seulement 1.72 millions d'entiers stockés dans une matrice de 48357 x 91349 (soit un remplissage de 1 élément pour 4000). Une fois générée, elle a été sauvegardée ainsi que le modèle pour l'API. Comparé à la matrice TF-IDF que l'on verra par la suite, aucune lemmatisation a été mise en place car la différence en termes de dimensions est peu importante.

La matrice Term Frequency-Inverse Document Frequency

De la même manière, une matrice TF-IDF a été générée. Celle-ci a beaucoup moins de dimensions car tous les mots ayant des petits scores ont été supprimés. De ce fait, la matrice finale est de 48357 x 2764 remplis avec 1.55 millions de float. Cette matrice est donc aussi moins creuse avec 1 élément sur 86. Celle-ci aussi a été sauvegardée ainsi que le modèle.

Réduction de dimensions – LSA

Pour pouvoir tester l'entrainement du modèle supervisé sur la matrice TF (actuellement impossible avec autant de dimensions). Le Latent Semantic Analysis qui permet de réduire les dimensions en ne gardant que les mots participant le plus à la variance a été testé. En réduisant de 91349 dimensions à 3000, on conserve 88% de la variance.

Malheureusement, le résultat est une matrice et un modèle très lourd (Matrice : 1.3go et modèle : 2Go) et difficilement exploitable en production (du moins sur PythonAnywhere).

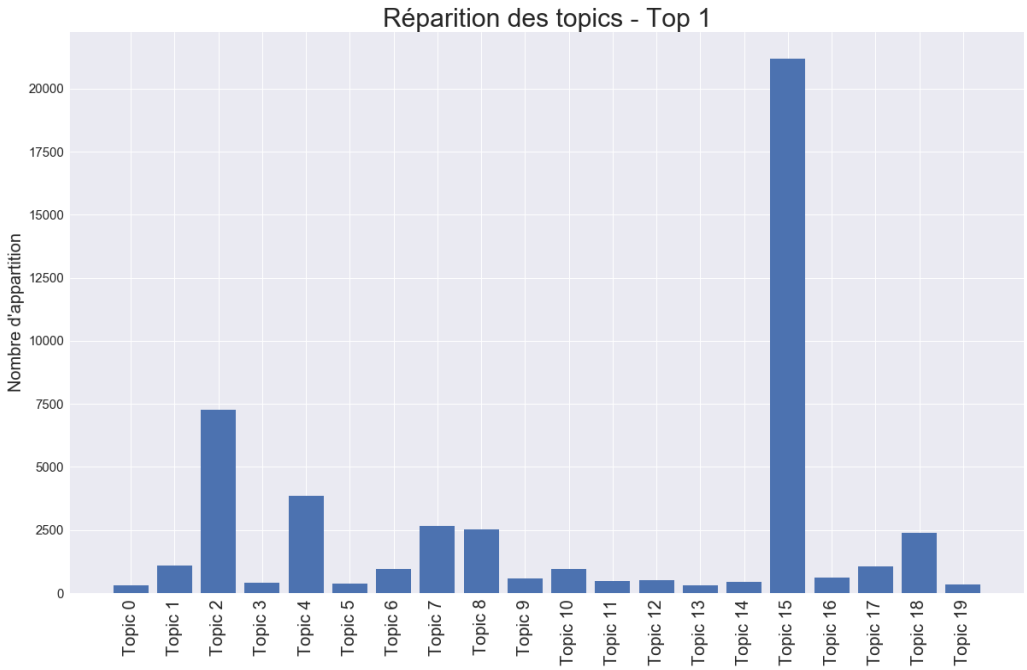
Modèle Non Supervisé

Latent Dirichlet Allocation

Basé sur la matrice TF, le LDA a été entrainé avec plusieurs tailles de choix de Topics. Si l'on met trop de sujets, beaucoup sont identiques et ont les mêmes principaux mots. Si on choisit 20 sujets, on peut voir ci-dessous les mots clés principaux et ainsi faire une analyse du sujet:

Topics	Mots clés	Analyse
1	javascript event js events node tag component tags form control	Gestion de formulaires HTML
2	string value number java memory values list variable two array	Types de données
4	text jquery element css html json button change set click	Mise en page site (CSS, js, forms)
7	project files android build version directory folder git studio eclipse	Gestion de projets/applications
8	table database sql query key mysql field column array id	Base de données
10	image images android size map points video draw plot matlab	Graphiques/images
18	page view net web http asp url controller request mvc	Fonctionnement site web

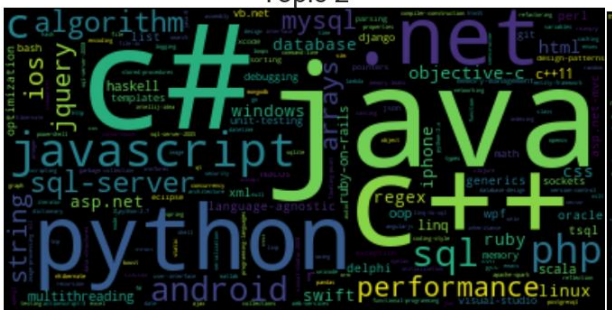
Un des problèmes de ce modèle sur ce type de données c'est que si l'on regarde la répartition du nombre de posts par sujets prépondérants, on a très majoritairement un seul sujet (#15) car celui-ci regroupe des mots très courants (tests, run, try, server, …). Pour améliorer la classification, on pourrait peut-être étendre les StopWords avec les plus fréquents de ce sujet.



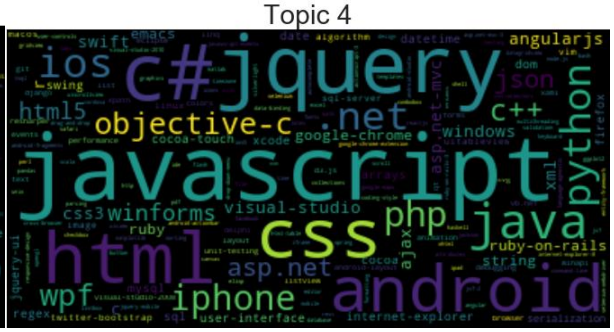
Analyse des tags par Topics

Basé sur le top 3 des topics de chaque sujet (pour éviter le biais du sujet 15), les tags ont été comptés. Si on fait un nuage de Mots basé sur leur nombre d'apparitions, on trouve :

Topic 2

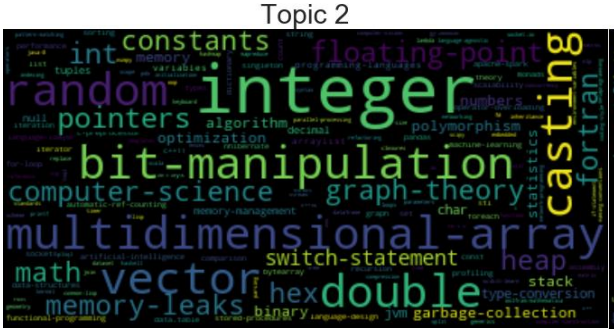


Topic 4

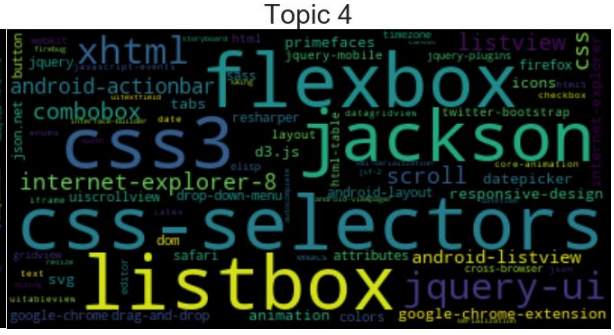


On a majoritairement les langages en tête. Cela s'explique car ce sont les plus utilisés. Si on veut des mots clés un peu moins courants, on peut diviser leur nombre d'apparition par le nombre d'apparition total dans le corpus (on appellera cela la méthode normalisée par la suite) et on trouve :

Topic 2



Topic 4



On a donc maintenant des tags moins fréquents et donc potentiellement plus susceptible d'aider l'utilisateur à tagger ses questions.

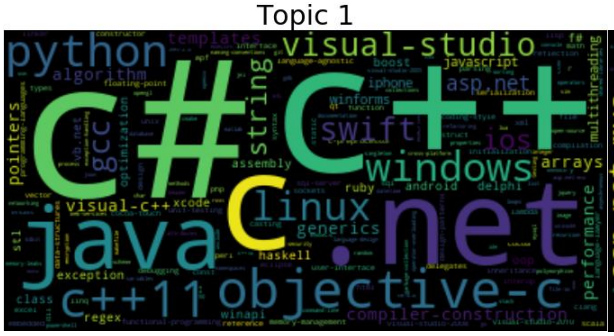
Non-Negative Matrix Factorization

Basé sur la matrice TF-IDF, la même analyse a été faite avec le NMF. Au niveau des sujets on trouve :

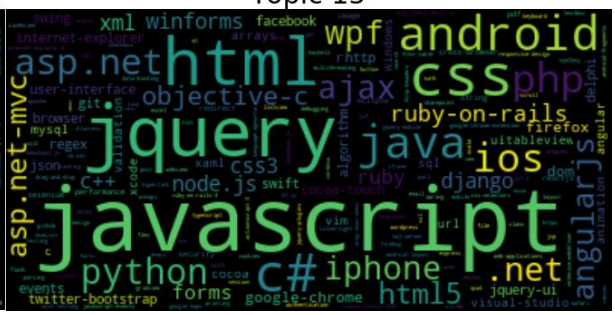
Topics	Mots clés (lemmatisés)	Analyse
1	c compil program librari languag pointer b gcc declar dll	Compilation, librairie et C
2	tabl column queri row sql databas mysql select index field	Base de Données
3	server sql connect client databas servic web request send http	Requete Server
10	array element byte loop index numpi sort pointer size number	Type de données et Structures
13	php script mysql variabl 5 upload session page ini email	PHP
15	page jqueri html element javascript button click text event div	Mise en page web
18	valu return variabl key set null properti default field type	Type de données

Si on regarde du coté des tags sans normalisation on trouve :

Topic 1



Topic 15



Et avec normalisation, on a des tags moins courants mais possiblement hors sujet.

Topic 1

Topic 15

Proposition de Tags par la méthode non supervisée

Dans l'objectif d'améliorer la proposition de tags proposé, il est possible de regarder uniquement les tags des posts les plus similaires (principe du KNN). Le LDA ou NFM donnant une répartition de probabilité, la métrique la plus adaptée est donc la Divergence de Jensen-Shannon. Pour un post donné, les 10 posts les plus similaires basé sur cette métriques ont été extraits et leurs tags comptés. Tout comme on l'a fait précédemment une version normalisée a été faite aussi pour avoir des tags moins courant et on trouve :

Tag du post choisi	Tags non normalisé	Tags normalisé
Php	C#	Event-handling
html	Javascript	Include
apache	.net	android-intent
compression	Winforms	javascript-events
	angular	iframe

Les tags sont bien relatif à de la programmation de pages web mais on ne retrouve pas les sujets liés à la compression ou au serveur apache.

Evaluation de la méthode non supervisée

Le modèle étant non supervisée, l'évaluation n'est pas vraiment évidente. L'objectif du projet étant de fournir des tags relatifs au sujet, une évaluation manuelle a été faite en regardant la prédiction sur 20 posts au hasard. Pour chacun, 5 tags sont prédits avec la méthode normalisé ou non. Si le tag a du sens, il a 1 point sinon 0. On a donc 100 tags évalués et on trouve pour score final 37% de tags cohérents sans normalisation et 42 % avec. Le moins bon score sans normalisation peut s'expliquer par la présence forte des langages de programmation qui sont donc souvent hors sujet.

Modèle Supervisé

Test de Modèles

Pour le modèle supervisé, la matrice TF-IDF a été utilisée car elle possède moins de dimensions et fait ressortir les mots moins courants. Plusieurs modèles ont été testés dont on retrouve un résumé ci-dessous :

Type	Modèles	Résultats
Multioutput	SGDClassifier	Train : 79.3% Test : 71.4%
OVR + Ensemble	AdaBoostClassifier GradientBoostingClassifier	Out of time (Stop après env. 20 min)
Multilabel + Ensemble	ExtraTreesClassifier RandomForestClassifier	Train : 37.5% - Test : 36.1% Train : 47.1% - Test : 45.1%
Multilabel	KNeighborsClassifier	Memory Error (n'utilise pas de matrice Creuse)
Multilabel	RidgeClassifierCV	Memory Error (inversion de trop grosses matrices)
Multilabel	MLPClassifier	Train : 82.0% Test : 68.9% (overfitting malgré Early Stop)

Le score a été mesuré suivant une fonction personnalisée. Au lieu de prédire les tags directement, une prédiction des pourcentages de probabilités ont été faits. Les 5 classes majoritaires ont été extraites. Ensuite, le nombre de classes en commun avec le post était compté. De ce fait, si un post a pour tag Python et que le top 5 regroupe Python, Algorithmes, C++, Integer et Array, il a 100 %. Par contre si le Post a pour tags C++, Pointers, Compiler et que la prédiction est la même alors la réussite n'est qu'a 33% car seul C++ est en commun sur les 3 tags. L'idée étant d'évaluer si, dans une proposition de 5 tags, on va prédire majoritairement ceux qui sont utiles.

Basé sur ces résultats, le Fine Tuning a été fait sur le SGDClassifier. Le MLPClassifieur n'a pas été retenu car étant calculé sur CPU, on ne peut avoir que peu de hidden layers. Or comme on a 2900 dimensions dans la matrice TF-IDF et 773 classes en sortie, il faut garder un nombre de neurones important dans le(s) Hidden(s) Layer(s). Avec un petit nombre, on se retrouve à perdre de l'information. Dans ce cas, celui-ci se comporte plus comme un Auto-Encoder qu'un Classifieur.

Cependant test a été fait sur Keras que vous trouverez dans le Notebook avec des performances légèrement supérieures.

Fine tuning - SGDClassifier

Sur ce modèle, on a un fort overfitting (8% de différence entre train et test set). De ce fait, un grid search a été mis en place sur le SGDClassifier afin de tester des régularisations. Le modèle est entrainé avec 3 cross-validations sur le train set. On trouve comme résultat :

Paramètres	Résultat Train Set	Résultat Test Set
alpha = 1e-6 – penalty = L1	98.8%	64.9%
alpha = 1e-6 – penalty = L2	98.1%	68.0%
alpha = 1e-5 – penalty = L1	82.9%	73.1%
alpha = 1e-5 – penalty = L2	87.7%	72.6%
alpha = 1e-4 – penalty = L1	66.5%	66.0%
alpha = 1e-4 – penalty = L2	63.7%	60.7%
…	…	…
alpha = 1e-2 – penalty = L1	18.4%	18.6%
alpha = 1e-2 – penalty = L2	26.2%	26.2%

On remarque que sans pénalité (ou avec une pénalité très faible), on a beaucoup d'overfitting. Cela s'explique par des poids très importants sur des mots moins lié au sujet. De ce fait, si le mot est utilisé dans un autre contexte, avec ce poids fort, il prédira un mauvais tag. Après régularisation, on supprime cet effet pervers. De ce fait, le modèle avec pénalité est conservé (le rouge au-dessus). Le résultat sur le dataset de test a ensuite été effectué avec pour résultat 66.18%.

Analyse des résultats

Prédictions

Une prédiction a été faite avec chaque modèle pour évaluer la pertinence des résultats. Les résultats sont les suivants (avec les % de prédictions):

Tags du Post	Prédiction du SGDClassifier
	python (45.5%)
python	c++ (21.2%)
iterator	c (3.86%)
iteration	java (3.7%)
	.net (2.32%)

On retrouve malheureusement que des langages utilisés pour ce type d'opérations ce qui n'est pas forcément judicieux car l'utilisateur n'est que sur un langage précis.

Analyse des mots impactant

Sur le SGDClassifier, on peut analyser par tags, les mots principaux qui impactent la décision. Ce sont ceux qui apportent le plus de poids dans le modèle. Si l'on fait ça pour certains tags on trouve:

Tags	Mots clés
pandas	panda, datafram, seri, feedback, feed
dataset	dataset, zoom, feel, feed, featur
python	python, numpi, panda, django, matplotlib
machine-learning	zoom, fatal, feedback, feed, featur
git	git, commit, branch, repositori, repo

On remarque que pour certains sujets, les mots clés sont très logiques comme par exemple Python et git. Par contre pour d'autres c'est moins clair comme Machine-Learning qui ne contient pas de mots très orienté sur ce sujet. On remarque aussi que *zoom, feel, feed* sont très présents pour beaucoup de tags, pour améliorer le modèle, une analyse des mots clés trop courant pourrait aussi être ajouté aux StopWords.

API

Pour l'API, différents modèles ont été sauvegardés. Le LDA, le TFIDFVectorizer, le CountVectorizer et le Classifieur. Quant au site, il contient un champ Titre et Texte. Comme le temps de calcul est assez long, la proposition de tags n'est pas Live et il faut demander les prédictions. La demande est faite via un bouton qui envoie une requête POST au serveur. Celui-ci génère les matrices TF et TF-IDF. La matrice TF est ensuite passée dans le LDA pour avoir les Topics de manière non supervisé et le KNN customisé est fait avec la similarité de Jensen Shannon. Les tags en version normale et normalisé. Quant à la matrice TF-IDF, elle est utilisée avec le classifieur. Celui-ci prédit aussi les probabilités de chaque classe et retourne les 5 principaux tags. Pour un gain de temps lors des essais, 4 Templates ont été mis en place sur des posts récents pris sur Stack Overflow.

Par exemple avec le Template 2, on remarque que l'utilisateur parle de l'utilisation du Depth First Search sur un graphe orienté. Les tags *algorithm*, *graph, sorting* sont logiques. Les tags *géolocalisation*, *gps, gis* sont moyennement logique car il est vrai que généralement le DFS est utilisé pour chercher le plus court chemin dans un graphe. Cependant ils ne correspondent pas vraiment à la question. Quant au reste, c'est plutôt lié aux langages mais on n'a pas d'indice là-dessus. C# est présent 2 fois car c'est surement le langage sur lequel il y a le plus d'utilisation du DFS (pour des raisons de performances).

Pistes d'évolutions

Le modèle fourni certains tags qui sont censé mais beaucoup sont à côté de la question. Peut-être que plus de données aurait permis d'avoir des résultats plus cohérents.

Pour ce qui est du modèle non supervisé, une agrégation des résultats du LDA et NMF pourrait peut-être rendre le résultat plus cohérent bien que les 2 aient environ les mêmes groupements. Augmenter les StopWords avec les mots du Topic 15 pourrait aussi permettre de rétablir une certaines balance entre les sujets. Comme on l'a vu aussi dans la partie supervisée, les mots ayant un poids fort dans plusieurs sujets pourraient aussi être ajoutés au StopWords.

Pour le modèle supervisé, plus de données serait nécessaires. En effet, bien qu'il y ait peu de modèles qui passent en mémoire, on a beaucoup de dimensions par rapport au nombre de posts. Le fléau des dimensions bloque peut-être un peu le learning, notamment sur des modèles non linéaires. Avec plus de données, on n'aurait surement pas autant de différence entre le modèle régularisé ou non.

Au lieu d'avoir un seul modèle qui fait l'ensemble des prédictions, un modèle pourrait être entrainé par langages et un second sur tout ce qui n'est pas un langage. Le 1er classifieur serait donc multi-classe. Le 2nd classifieur serait de type multi-label et serait là pour proposer des tags liés au problème.

La dernière possibilité serait qu'à chaque sélection de tags, le classifieur referait une prédiction en utilisant le contenu mais aussi le tag choisi. Ce modèle serait bien plus complexe mais permettrait d'évoluer en fonction des inputs de l'utilisateur en Live.

Conclusion

Lors de ce projet, nous avons abordé un des secteurs du Data-Scientist qui est l'analyse de données textuelles. Différents modèles ont été mis en place afin de prédire des tags censés à un utilisateur.

Le résultat actuel n'est pas parfait mais permet au moins d'extraire les sujets de la question et ainsi proposer des tags en rapport mais parfois assez éloigné de la question. Au niveau Classification supervisée, les modèles ont des performances supérieures aux modèles non supervisés qui sont plus utiles pour comprendre les sujets abordés.

La 1ère difficulté sur ce projet est que les sujets sont tous assez proches et liés à la programmation et la seconde est sur la longueur des questions. Celle-ci sont parfois très/trop courte pour une bonne classification (on a peu de mots dans la matrice TF-IDF). Plus généralement le LDA est plus utilisé pour séparer des sujets très différents (cuisine, nourriture, habitation, etc…) sur des corpus plus gros (articles de journaux, page web, livre, …).