Recommandation de Tags

Contenu

[Introduction 2](#_Toc502494503)

[Préparation des données 2](#_Toc502494504)

[Récupération des dataset 2](#_Toc502494505)

[Pré-traitement du dataset 2](#_Toc502494506)

[Exploration 3](#_Toc502494507)

[Préparations des Matrices 4](#_Toc502494508)

[La matrice Term Frequency 4](#_Toc502494509)

[La matrice Term Frequency-Inverse Document Frequency 4](#_Toc502494510)

[Réduction de dimensions – LSA 4](#_Toc502494511)

[Modèle Non Supervisé 5](#_Toc502494512)

[Latent Dirichlet Allocation 5](#_Toc502494513)

[Non-Negative Matrix Factorization 6](#_Toc502494514)

[Proposition de Tags par la méthode non supervisée 7](#_Toc502494515)

[Modèle Supervisé 7](#_Toc502494516)

[Test de Modèles 7](#_Toc502494517)

[Fine tuning 8](#_Toc502494518)

[API 9](#_Toc502494519)

[Pistes d'évolutions 9](#_Toc502494520)

[Conclusion 10](#_Toc502494521)

# 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 la question posée. Cela dans le but d'aider les nouveaux membres sur le site Stack Overflow à mieux classifier leurs questions et ainsi avoir des réponses plus pertinentes.

Dans un 1er temps, nous allons récupérer des datasets, explorer un peu leur contenu et faire un peu de 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. Pour finir, l'API vous sera présentée avec des critiques sur la prédiction. 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 donc aussi téléchargé pour s'assurer qu'il n'y ait pas de doublons. Les requêtes sont donc :

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

#### Titre

Concernant le titre, il n'y a pas besoin de traitements particuliers. Il a juste été fusionné avec le body pour avec un corpus constitué que d'une seul feature

#### Tags

Pour les tags, ils sont comme des balises HTML (<tag>). 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).

#### Body

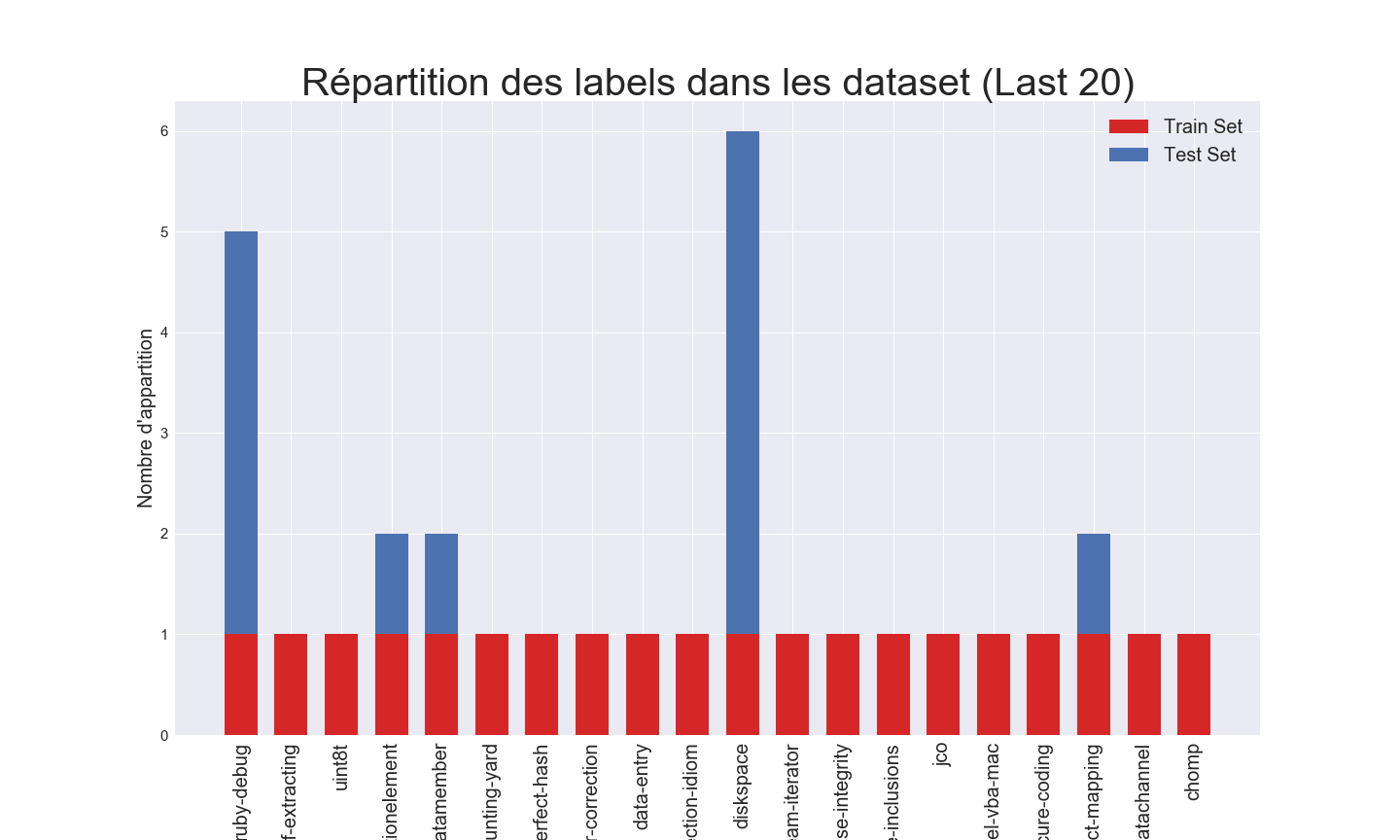
Pour le body, il y a plus de travail. Le texte entre les balises code ont été supprimés, les balises aussi. Cela a permis d'avoir un corpus avec moins de sens mais ne conservant que du "vrai texte". Les balises ne sont là que pour le rendu sur le site. Concernant le code, j'ai pris la décision de la supprimer car la majorité des mots sont uniques car ce sont des variables. On perd certes de l'information sur le langage avec des mots clé. Cependant beaucoup de ces mots clés sont les même dans tous les langages (for, while, break, try, return, …).

Un compte des mots courant a été fait ensuite et le top 100 a été rajouté aux StopWords (English) de nltk plus spécifiques aux Sujets de Stack Overflow.

# Exploration

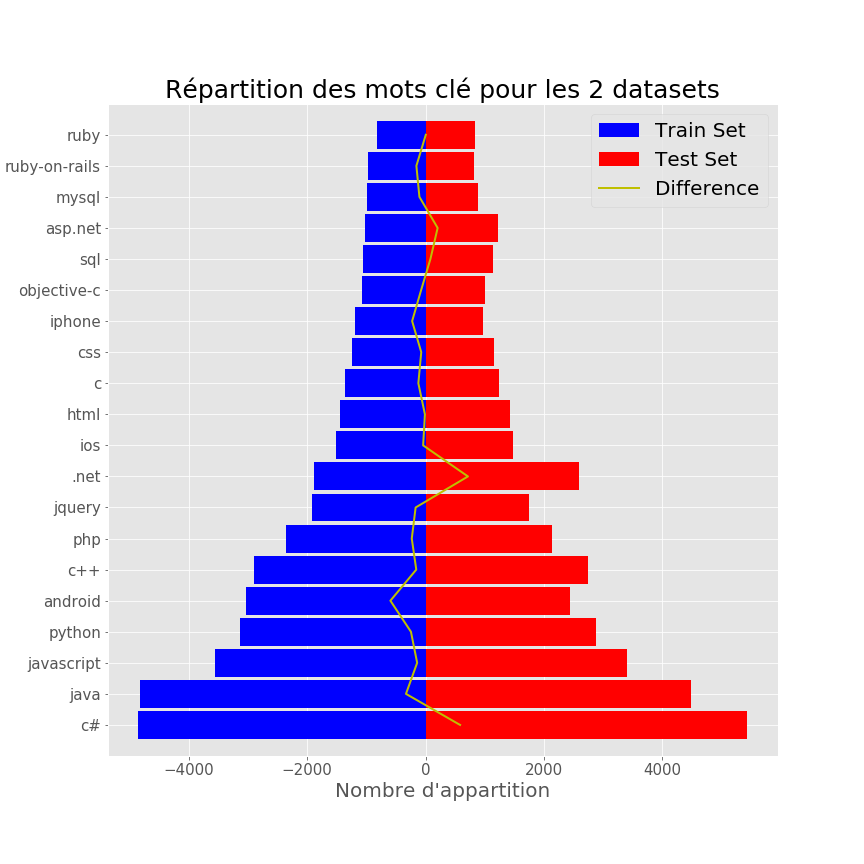
Après la phase de nettoyage, une exploration des tags a été faite. Sur 50 000 posts, plusieurs milliers de tags ont été utilisés. Afin de pouvoir faire un classifieur par la suite, je n'ai gardé que les tags avec plus de 25 apparitions dans les trains set. Tous les autres tags ont été supprimés.

En effet si on regarde les moins courants, ils n'ont qu'une apparition dans le train set et entre 0 et 3 apparitions dans le test set :



Au final on conserve 773 tags. Par la suite, j'ai supprimé tous les posts n'ayant plus de tags. Cela représente que très peu de posts (< 1000 sur le train set et environ 3000 sur le test set).

Afin de s'assurer de la balance des 2 datasets, on peut regarder la différence entre chaque 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

Le titre et la question étant groupés, on a maintenant un seul corpus de presque 50000 posts. Celui-ci été utilisés 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 dimension est peu importante.

## La matrice Term Frequency-Inverse Document Frequency

De la même manière, une matrice TF-IDF a été générée. Celle-ci a bien 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 1.55millions 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 aussi sur la matrice TF (c'est actuellement impossible avec autant de dimensions car il y aura le fléau de la dimension). J'ai voulu tester le Latent Semantic Analysis qui, comme le PCA, permet de réduire les dimensions en ne gardant que les mots participant le plus à la variance. Le résultat est donc similaire au TFIDF car les mots trop fréquents ne participent pas à la variance et seront donc supprimés.

Malheureusement, le résultat est une matrice très lourde (1.3go) et un modèle aussi très important (2go). De plus l'entrainement est très lent (plusieurs minutes). Il devient donc peut intéressant car il faut aussi penser au fait qu'il doit servir en production.

On a donc toutes les matrices nécessaires aux différents modèles.

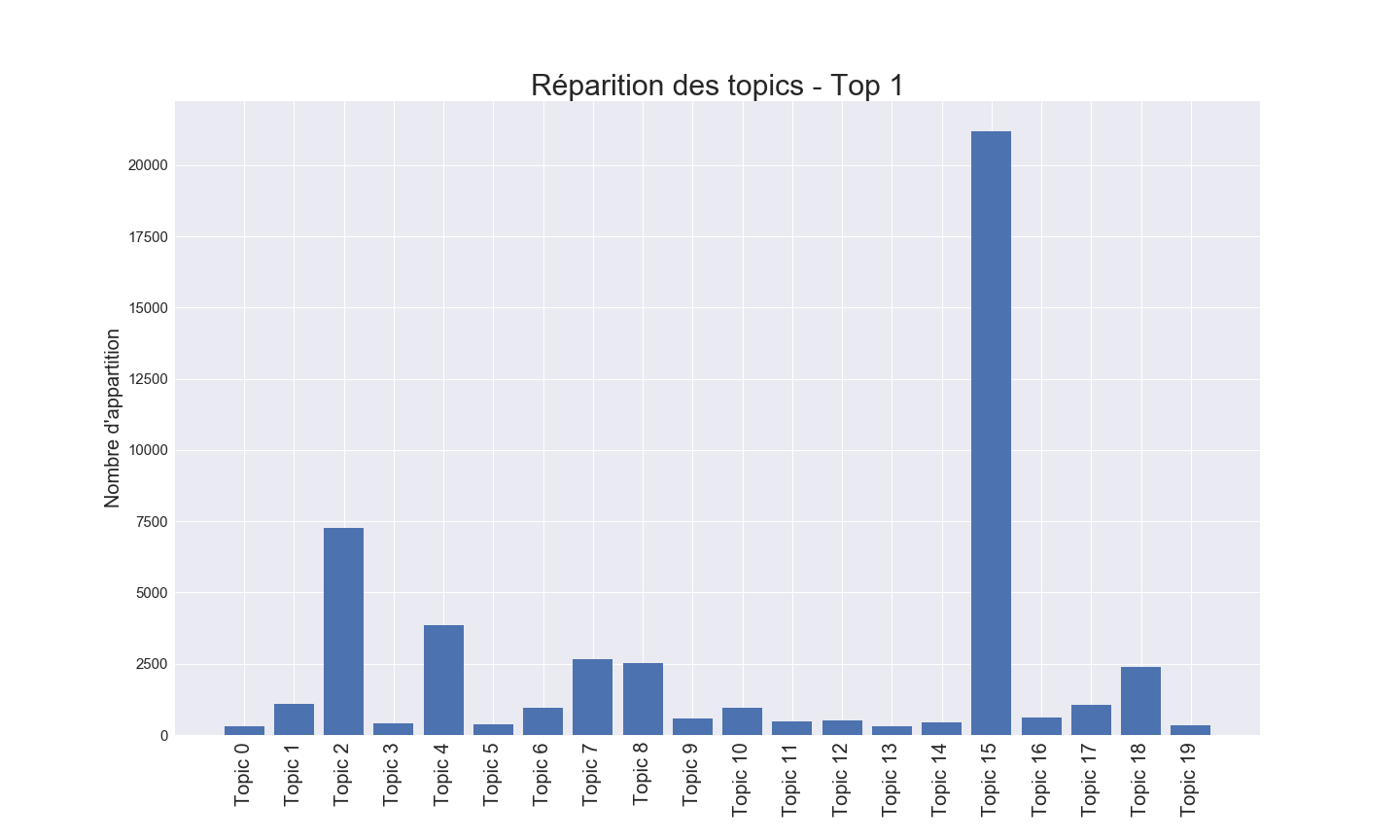
# Modèle Non Supervisé

### 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 topics, beaucoup sont identiques et ont les mêmes principaux mots. Si on choisit 20 topics, on peut voir ci-dessous les mots clé 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 |

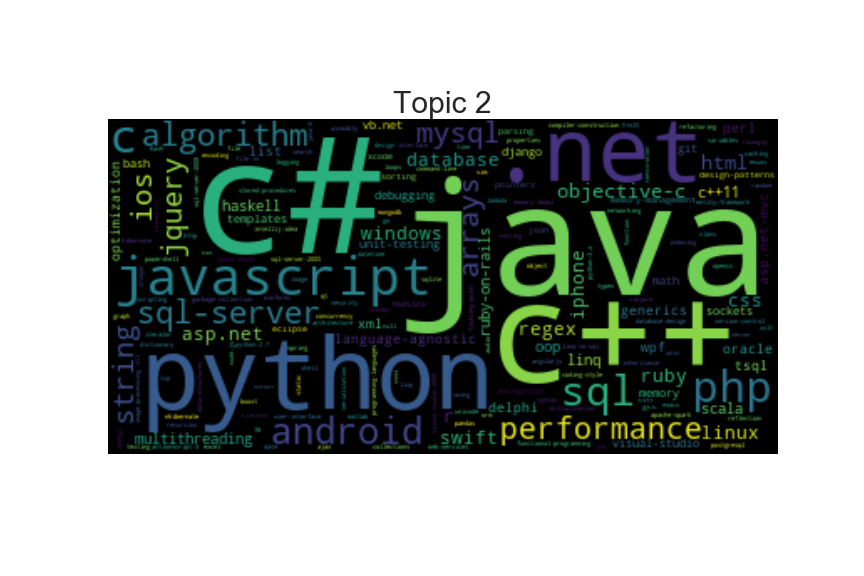
Un des problèmes de ce modèle sur ce type de données c'est que si l'on prendre le topic prépondérant du sujet, on remarque qu'ils sont majoritairement dans un seul topic (15) car celui-ci regroupe des mots très courants (tests, run, try, server, …)



Peut-être que les StopWords étendu lors de la préparation n'ont pas suffi.

#### Analyse des tags par Topics

Basé sur le top 3 des topics de chaque sujet, les mots clé ont été comptés. Si on fait un nuage de Mots sur ceux-ci on trouve :



On a majoritairement les langages en tête. Cela s'explique car ils sont en tête des tags. Si on veut des mots clé un peu moins courants, on peut diviser leur nombre d'apparition par le nombre d'apparition dans le corpus et on trouve :





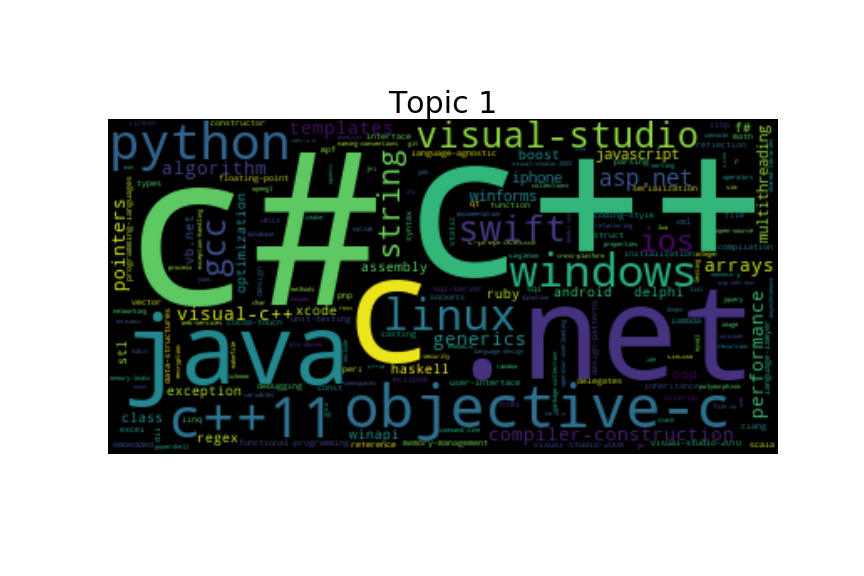
On a donc maintenant des tags plus parlants si on le compare aux topics présenté précédemment.

### Non-Negative Matrix Factorization

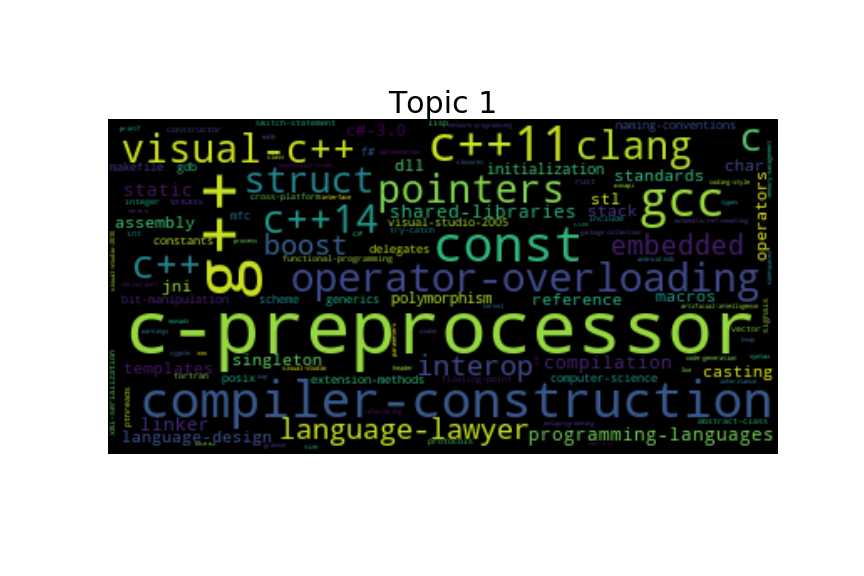
Basé sur la matrice TF-IDF, la même analyse a été faite avec le NMF. Au niveau des topics on trouve :

|  |  |  |
| --- | --- | --- |
| 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 :



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



### 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 ont été extraits et leurs tags compté. 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, html, apache, compression | C#, javascript, .net, winforms, angular | Event-handling, include, android-intent, javascript-events, iframe |

Les tags sont bien relatif a de la programmation de page web mais on ne retrouve pas les topics liés à la compression ou au serveur.

# 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 |
| OVR | SGDClassifier | Train : 62.681%  Test : 60.578% |
| OVR + Ensemble | AdaBoostClassifier  GradientBoostingClassifier | Out of time |
| OVR | GaussianProcessClassifier | Memory Error (n'utilise pas de matrice Creuse) |
| Multilabel + Ensemble | ExtraTreesClassifier  RandomForestClassifier | Prédiction a 0.99 pour toutes les classes |
| Multilabel | KNeighborsClassifier | Memory Error (n'utilise pas de matrice Creuse) |
| Multilabel | RidgeClassifierCV | Memory Error (inversion de trop grosses matrices) |
| Multilabel | MLPClassifier | Train : 82.051%  Test : 68.891% (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 classe 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 topic 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.

Basé sur ces résultats, le Fine Tuning a été fait sur le SGDClassifier et à des fins uniquement de comparaison, le MLPClassifier a été remplacé par Keras afin de pouvoir avoir de plus large Hidden Layers et des performances supérieures.

## Fine tuning

#### SGDClassifier

Un grid search a été mis en place sur le SGDClassifier afin de tester plus d'iterations et des régularisations. On trouve comme résultat :

|  |  |  |
| --- | --- | --- |
| Paramètres | Résultat Train Set | Résultat Test Set |
| max\_iter = 10 – penalty = None | 77.45% | 70.68% |
| max\_iter = 10 – penalty = L1 | 66.61% | 66.09% |
| max\_iter = 10 – penalty = L2 | 63.16% | 61.01% |
| max\_iter = 20 – penalty = None | 77.53% | 70.67% |
| max\_iter = 20 – penalty = L1 | 66.42% | 65.97% |
| max\_iter = 20 – penalty = L2 | 63.08% | 60.93% |

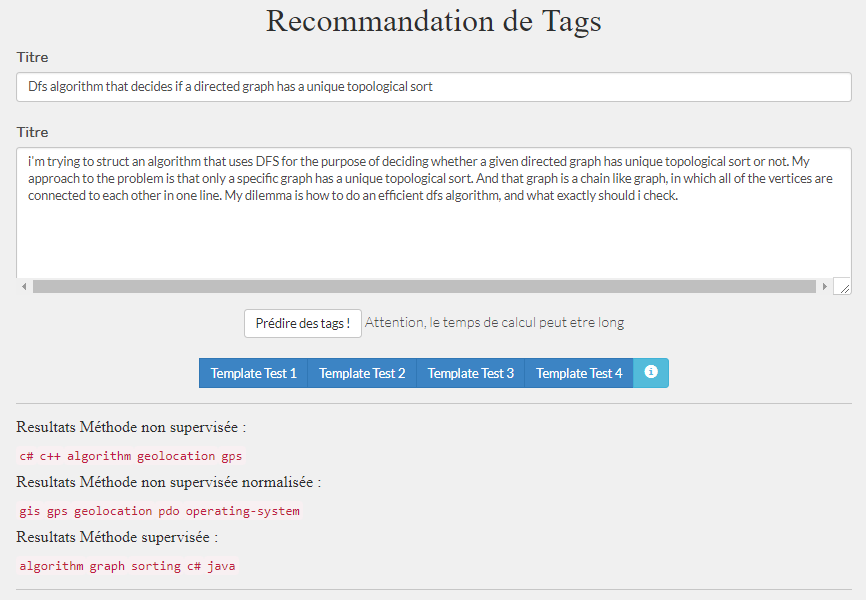
On remarque que sans pénalité on a un peu d'overfitting mais le résultat sur le test set reste le meilleur. Cela s'explique par des poids très importants sur des mots moins lié au sujet. De ce fait, si le mot est utilisé dans un autre contexte, avec ce poids fort, il prédira un mauvais tag. Apres régularisation, on supprime cet effet pervers mais la prédiction est aussi moins bonne à cause de la restriction sur les poids. De ce fait, le modèle sans pénalité est conservé.

#### MLPClassifieur

Comme on a 2900 dimensions dans la matrice TF-IDF et 773 classes en sortie, il faut garder un nombre de neurones dans chaque Hidden Layers. Dans le cas du MLPClassifier, on perd beaucoup d'informations car on réduit fortement ce nombre car les calculs sont faits sur le CPU. Dans cas ces, j'ai mis 1 Hidden Layer de 1500 neurones puis 773 neurone avec une sortie de type Sigmoïde pour avoir les probabilités de chaque classe. Le Loss choisi est donc la BinaryCrossEntropy. Après 5 Epochs, le Early Stop est activé avec 86% de précision en Train Set et 72.1% sur le Test Set. La généralisation n'est encore pas terrible car certains mots prennent trop d'importance et rend le modèle plus difficile à généraliser.

# API

Pour l'API, différents modèles ont été sauvegardés. Le LDA, le TFIDFVectorizer, le CountVectorizer et le Classifieur. Le site contient un champ Titre et Texte, Comme le temps de calcul est assez long, la proposition n'est pas Live et il faut demander les prédictions. La demande est faite via un boutton qui poste le contenu au serveur. Celui-ci généré les différentes matrice TF et TF-IDF. La matrice TF est ensuite passé dans le LDA pour avoir les topics et le KNN avec la similarité de Jensen Shannon est appliqué pour avoir les tags normalisés ou non. Le classifieur prédit aussi les probabilités de chaque classe et retourne les 5 principales. Pour un gain de temps, 4 templates ont été mis en place et cela donne sur le Template 2 :



On remarque l'utilisateur parle de l'utilisation du Depth First Search sur un graphe orienté. Les tags algorithm, graph sont logique. Les tags géolocalisation, gps 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 la dessus.

# Pistes d'évolutions

Le modèle fournit 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 topics.

Pour le modèle supervisé, plus de données serait nécessaire. 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.

Un modèle pourrait aussi être entrainé basé sur un des tags choisi. Par exemple, lié au sujet on propose 5 tags, si l'utilisateur en choisi un, le modèle re-prédit 5 tags en fonction du texte et du tag choisi.

# 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 n'est pas parfait mais permet au moins d'extraire des tags dans des tendances correctes. Au niveau Classification, les modèles ont des performances supérieures au modèle non linéaires qui sont plus utile pour comprendre les sujets abordées.

**+PROBLEME**