Rapport du projet 6

Développer un moteur de recommandations de tags à partir de questions posées sur un site internet

Parcours Data scientist Open Classrooms

Mentor : Mohammed Sedki

Mentoré : Pierre-Emmanuel Beaumale

Date : 29/06/2018

Table des matières

**Introduction3**

**Partie 1 : Extraction et nettoyage des données3**

Commentaires généraux sur nos données3

Synthèse des traitements effectués sur le jeu de données 4

Extraction d’un jeu de test en généralisation 4

**Partie 2 : Exploration de notre jeu de données5**

Horizon temporel des questions sélectionnées5

Traitement des mots contenus dans les données textuelles5

Contenu de la variable « Tags »6

Contenu de la variable « Title »  7

Contenu de la variable « Body ».8

Conclusions de notre exploration de données.8

**Partie 3 : Approche non supervisée9**

Approche par l’algorithme LDA 9

Construction d’une matrice mot-document en utilisant tf9

Recherche du nombre de thèmes maximisant la vraisemblance9

Interprétation et visualisation en utilisant l’outil pyLDAvis10

Génération de 5 tags à partir du modèle LDA10

Approche PLSA 11

Construction d’une matrice de mot-document en utilisant tf-idf11

Réduction dimensionnelle par NMF11

Interprétation des résultats obtenus11

Génération de 5 tags à partir du modèle PLSA12

Conclusion de notre approche non supervisée 12

**Partie 4 : Approche supervisée13**

Modélisation du problème  13

Transformation des données cibles13

Construction d’une matrice mot-document en utilisant tf-idf13

Résultats obtenus par une approche binary relevance 13

Résultats obtenus en utilisant un classifieur multilabel14

**Partie 5 : Création d’une API web**14

Lien vers l’API 14

**Conclusions et perspectives15**

**Introduction**

Notre objectif à travers ce projet sera de bâtir un moteur de recommandations de tags afin d’aider un utilisateur du site Stack Overflow à compléter les tags nécessaires à la question qu’il souhaite poser sur le site. Notre démarche consistera donc à partir de questions préalablement extraites sur Stack exchange à bâtir un moteur de recommandations pertinents relatifs aux questions posées.

Contrainte de notre projet :

* Mettre en œuvre une approche non supervisée.
* Utiliser une approche supervisée ou non pour extraire des tags à partir des résultats précédents.
* Comparer ses résultats à une approche purement supervisée, après avoir appliqué des méthodes d’extraction de features spécifiques des données textuelles.
* Mettre en place une méthode d’évaluation propre, avec une séparation du jeu de données pour l’évaluation. Pour suivre les modifications du code final à déployer, utiliser un logiciel de gestion de versions, par exemple git.

Ce moteur de recommandations sera interrogeable sous la forme d'une API construite à l’aide du micro framework Flask hébergée sur le site pythonanywhere à l’adresse :

<http://peb33100.pythonanywhere.com/tagging>

**Partie 1 : Extraction et nettoyage des données**

* 1. **Commentaires généraux sur le jeu de données :**

A partir de requêtes, 195 855 questions brutes ont été extraites. Afin d’éviter des questions trop atypiques ou peu pertinentes, un filtre a été fait lors de l’extraction afin de n’extraire que des questions dont le score est supérieur à 5. Une fois ces données extraites, nous avons choisi de ne conserver que les features « CreationDate », « Score », « Title » « Body » et « Tags ». L’ensemble des questions avec données manquantes ont été supprimées. Nous avons par la suite choisi de sélectionner uniquement 30 % de ces questions de manière aléatoire afin :

* De permettre le traitement des données dans un temps raisonnable.
* D’éviter de prendre des données dans des périodes trop proches où certains sujets seraient présents de manière concentrés.

Nous avons ainsi sélectionné 49 745 questions qui nous serviront à réaliser notre exploration et notre modélisation.

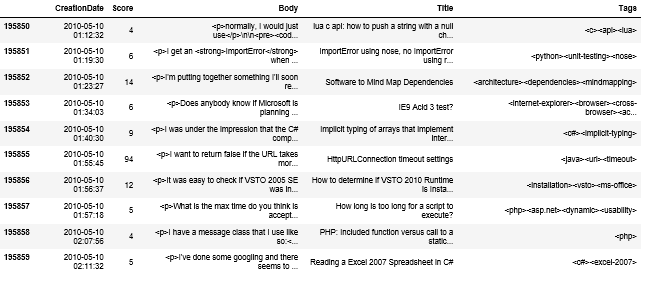


Figure 1 : Aperçu des données brutes sélectionnées.

* 1. **Synthèse des traitements effectués sur le jeu de données :**

Afin d’explorer ces données, nous allons préalablement effectuer un nettoyage de ces données :

* En supprimant les balises html présentes dans la variable « Body » en utilisant la libraire python BeautifulSoup.
* En supprimant les chevrons de la variable « Tags » et en transformant son contenu en liste de « Tags ».

Nous allons également créer une feature « Year\_month» qui contiendra l’année et le mois à laquelle la question a été créée afin de mieux comprendre l’horizon temporelle des questions contenues dans le jeu de données.

* 1. **Extraction d’un jeu de test en généralisation :**

Tout au long de ce projet, nous procéderons de manière classique à une séparation des données en jeu d’entrainement et jeu de données. Cependant afin d’évaluer nos algorithmes sur un même jeu de données, nous avons fait le choix d’extraire un jeu de données indépendant de l’ensemble des jeux de tests et d’entrainements sur lequel tous les modèles seront évalués.

**Partie 2 : Exploration de notre jeu de données**

Nous allons maintenant procéder à l’exploration des données contenues dans ce jeu de données afin de mieux comprendre son contenu.

**2.1 Horizon temporel des questions sélectionnées :**

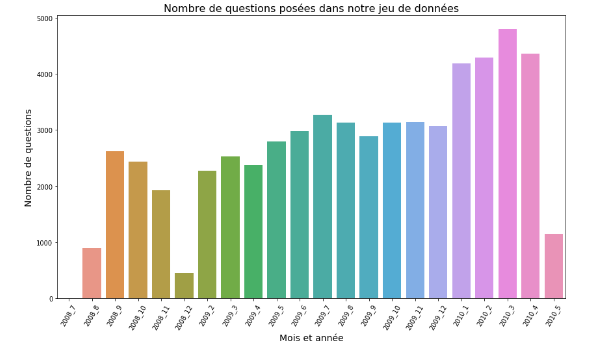
****

Figure 2 : Horizon temporel des questions

On peut voir que les dates de création des questions s’étalent de septembre 2008 à mai 2010 ce qui nous assure un horizon temporel de plus d’un an. On remarque que notre jeu de données contient une importante quantité de questions créée début 2010, période à laquelle les questions posées sur le site semblent plus nombreuses.

**2.2 Traitement des mots contenus dans les données textuelles :**

Afin d’explorer les données textuelles, nous avons d’abord procéder à un traitement des mots afin de limiter au maximum les « stop\_words », mots de langage n’ayant pas de sens sémantique et les doublons potentiellement liés à des formes de conjugaison des mots différentes. Nous avons ainsi développé notre propre « tokenizer », fonction de nettoyage et de traitement des mots permettant d’extraire les features nous intéressant dans le vocabulaire. Ce tokenizer effectue un traitement par étape sur les mots en :

* Supprimant les chiffres et les caractères spéciaux (à l’exception de certains mots clés, ex: c++) en utilisant une extraction des mots suivant une expression régulière.
* En utilisant le Part Of Speech tagging (POS\_tagging) afin de conserver uniquement les mots de type nom, en cas de retour nulle de la fonction, on déclare par défaut un mot en nom et un traitement spécifique est alors effectué.
* La dernière étape consiste ensuite à conserver le lemme du mot (forme canonique) afin de réduire le vocabulaire en supprimant les doublons.

Ce traitement des mots a été utilisé tout au long de ce projet pour manipuler et travailler sur les données textuelles.

**2.3 Exploration du contenu de la variable « Tags » :**

Le contenu de « Tags » contient au total 174 895 mots, dont 10797 mots uniques, ce qui donne une richesse lexicale de 6,17% montrant que nombreux tags sont utilisés de manière redondante.

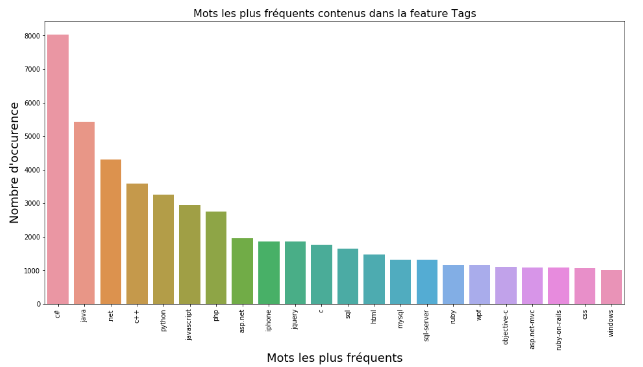


Figure 3 : Représentation des mots les plus fréquents dans la variable « Tags ».

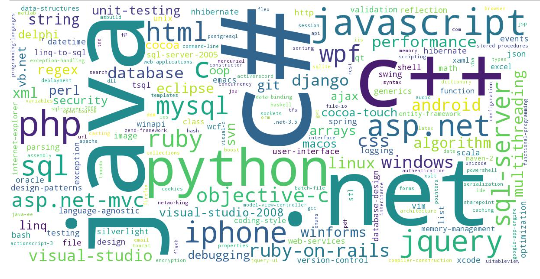


Figure 4: Nuages de mots du contenu de la variable « Tags ».

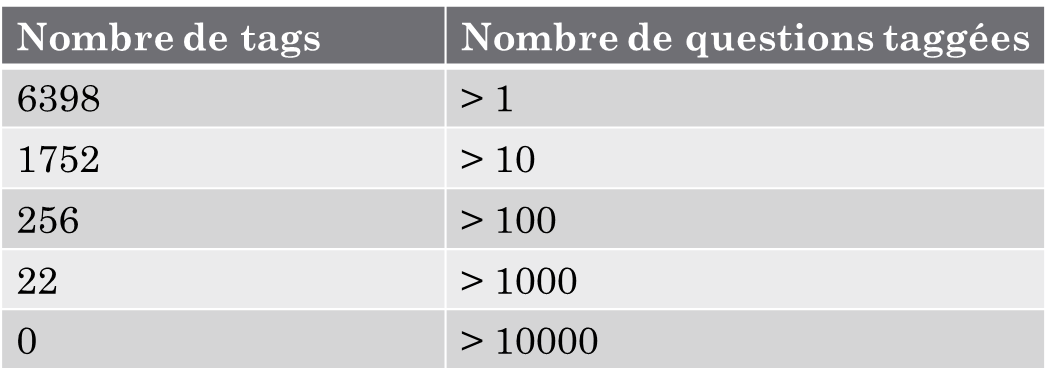


Table 1: Nombre de tags présent dans au moins 1 à 10000 questions.

**2.4 Exploration du contenu de la variable « Title » :**

Le contenu de « Title » contient au total 198 115 mots, dont 11 858 mots uniques, ce qui donne une richesse lexicale de 5,99% montrant que nombreux mots sont souvent répétés.

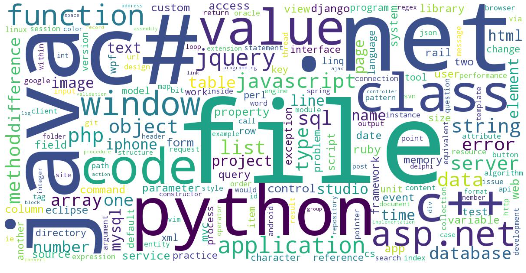


Figure 5: Nuages de mots du contenu de la variable « Title ».

On observe visuellement que les mots les plus fréquents semblent « proches » du contenu de tags. En effet, en comparant les ensembles de mots on s’aperçoit que 40,4% des tags sont présents dans la variable « Title » ce qui représentent au total 36,8 % des mots de « Title » ce que s’interprète tout simplement par le fait qu’un peu plus d’un mot sur trois du titre sont des tags ce qui représente une forte concentration des tags dans les mots du titre.

* 1. **Exploration du contenu de la variable « Body » :**

Le contenu de « Body » contient au total 2 210 739 mots, dont 82 050 mots uniques, ce qui donne une richesse lexicale de 3.71% montrant que les mots du corps de la question ont tendance à être souvent les mêmes (présence de nombreux stop\_words).

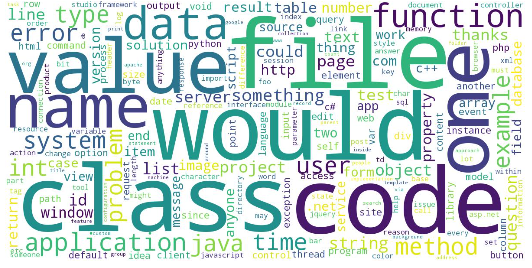


Figure 6 : Nuages de mots du contenu de la variable « Body ».

On observe visuellement que les mots les plus fréquents de la variable « Body » sont différents de ceux de « Tags » et de « Titre ». En comparant les ensembles de mots de « Body » et de « Tags », on observe que 49,85 % des tags sont contenus dans « Body » ce qui représentent seulement 6,6 % du contenu de « Body ». Les tags sont ainsi davantage présents mais plus « diluées » dans le corps de la question.

* 1. **Conclusion de notre exploration**

Nous avons donc vu dans cette exploration que le contenu de tags contient de nombreux mots décrivant seulement 1 à 10 questions, ces tags ne sont pas pertinents dans le cadre de ce projet en raison du faible nombre de données. Notre approche devra donc se focaliser sur un certain nombre de tags plus fréquents qui nous permettront d’identifier des thèmes communs au sein des questions. Nous avons également vu que le contenu des tags se retrouvent dans le titre ainsi que dans le corps de la question, nous allons donc utiliser ces deux entrées afin de construire notre algorithme. Nous fusionnerons donc le contenu de « Title » et de « Body » ce qui nous permettra de contenir 50,5 % des mots contenus dans « Tags » dans la nouvelle variable créée dénommée « BodyTitle ».

**Partie 3 : Approche non supervisée**

Nous allons dans cette partie explorer l’utilisation de deux techniques particulièrement importantes dans le domaine de la modélisation de thèmes contenus dans des corpus de documents textuels : il s’agit de l’algorithme de l’allocation de Dirichlet latente (LDA) développée par Blei, A. Ng, and M. Jordan en 2003 et de l’approche plus ancienne de la probabilistic latent semantic analysis (PLSA).

**3.1 Approche non supervisée par Latent Dirichlet Allocation :**

**3.1.1 Transformation de la variable « BodyTitle » en matrice mot-document :**

Avant d’appliquer l’algorithme LDA, nous allons d’abord représenter notre corpus de documents en une représentation matricielle où chaque document sera représenté par un vecteur de fréquence de mots du vocabulaire défini sur notre corpus de documents (approche tf). Nous exploiterons pour cela les possibilités de la bibliothèque python sklearn et de sa fonction CountVectorizer. Après plusieurs essais, nous avons choisi de restreindre le vocabulaire de représentation de chaque question aux mots présents dans au moins 0.1 % des questions (environ 50 questions) et dans au maximum 0.3 % des questions (environ 15000 questions).

**3.1.2 Recherche du nombre de thèmes par maximisation de la vraisemblance :**

Afin de trouver les paramètres optimaux de notre LDA, nous avons effectué une grille de validation croisée sur les paramètres nombre de thèmes et sur le paramètre learning\_decay. Cette validation croisée nous a permis d’identifier un maximum de vraisemblance obtenue pour 20 thèmes et un paramètre learning\_decay dont la valeur est de 0.75.

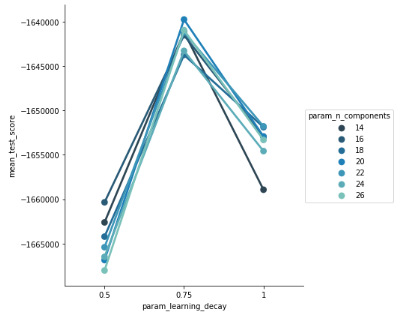
****

Figure 007 : Résultats de validation croisée sur les paramètres n\_components et learning\_decay de l’algorithme LDA.

**3.1.3 Exploration des thèmes identifiés par mots clés :**

Nous allons afficher chacun de mots associés aux thèmes retenus par ordre de fréquence d’apparition.

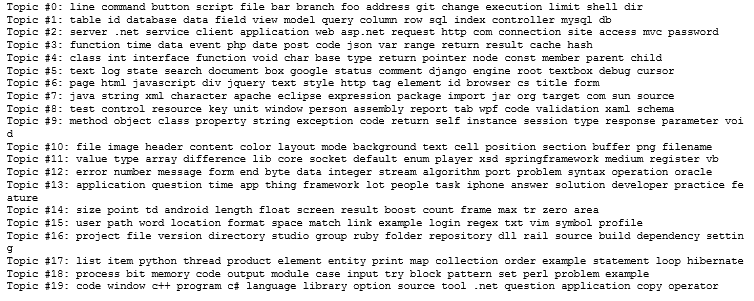
****

Figure 008 : Mots les plus fréquents associés à chaque thème identifié.

L’utilisation de l’outil pyLDAvis permet une visualisation plus efficace des mots contenus dans les thèmes en réorganisant ceux-ci suivant le critère de relevance défini par Sievert et Shirley auteur de l’outil LDAvis. Le contenu de cet outil peut être visualisé dans le fichier joint « lda.html ».

On remarque des thèmes facilement interprétables comme le numéro 1 lié aux bases de données ou le numéro 7 lié au monde java. D’autres thèmes sont plus difficilement interprétables.

**3.1.4 Génération de 5 tags à partir du modèle LDA :**

Cette première approche nous permet alors en séparant le jeu de données en jeu d’entrainement et jeu de test de construire une fonction simple permettant d’identifier le thème d’une question et de prédire les mots les plus « relevant » associés à ce thème comme tags à la question. Cette approche nous permet d’obtenir 5 tags et un score de prédiction des vrais tags de l’ordre de 5,17 % sur le jeu de test et de 4,9 % sur notre jeu de données de tests en généralisation.

**3.2 Approche non supervisée par PLSA :**

L’approche PLSA est équivalente à procéder à une décomposition NMF sur la matrice mot-document dont les coefficients représentent le score du taux de fréquence d’un mot sur l’inverse de sa fréquence dans les différents documents. Nous implémenterons donc cette approche dans cette partie en utilisant le même nombre de thèmes que la partie 3.1.

**3.2.1 Transformation de la variable « BodyTitle » en matrice mot-document en utilisant le tf-idf :**

Nous allons représenter notre corpus de documents en une représentation matricielle mot-document où chaque document sera représenté par un vecteur de fréquence des mots contenus dans le document divisé par la fréquence de ces mots dans l’ensemble des autres documents (approche tf-idf). Nous exploiterons pour cela les possibilités de la bibliothèque python sklearn et de sa fonction TfidfVectorizer. Après plusieurs essais, nous avons choisi de restreindre le vocabulaire de représentation du corps de documents aux mots présents dans au moins 0.1 % des documents (environ 50 documents) et dans au maximum 0.3 % des documents (environ 15000 documents).

**3.2.2 Décomposition NMF de notre matrice mot-document :**

Nous allons effectuer une décomposition matricielle de type NMF afin de projeter notre matrice dans un espace de matrice positif de plus petite dimension en utilisant la norme de Kullkack-Leibler. Les paramètres de l’algorithme NMF seront les valeurs par défaut de l’algorithme sous sklearn.

**3.2.3 Exploration des thèmes identifiés par mots clés :**

Nous allons de même représenter les mots dont les scores tf-idf sont les plus importants associés à chaque thème afin de mieux comprendre le contenu de ses thèmes.

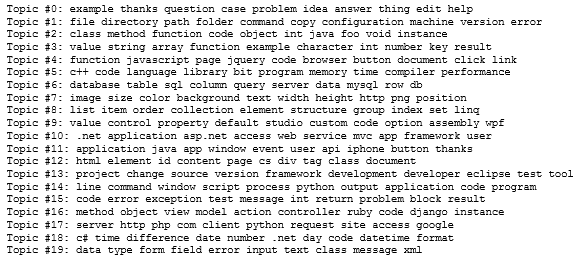
****

Figure 009 : Mots avec les scores tf-idf les plus importants pour chacun des thèmes.

On observe des résultats proches des résultats obtenus par l’algorithme LDA. On notera cependant que le coût de calcul est bien moindre que l’algorithme LDA.

**3.2.4 Génération de 5 tags à partir du modèle PLSA :**

De même, nous allons à partir de cette approche construire un générateur de 5 tags associés au thème prédit par le modèle PLSA. La performance de ce modèle est également médiocre avec des scores de prédiction de 5,89 %.

**3.3 Conclusion de notre approche non supervisée :**

Nous avons donc vu que les scores de prédiction obtenus pour ces deux modèles ne sont pas bons. Ces deux méthodes semblent cependant particulièrement intéressantes afin d’identifier des thèmes sous-jacents (parfois sémantiquement profond), d’identifier l’appartenance d’un ensemble de mots à un thème mais ne semble pas adapter à la génération de tags comme nous le souhaitons dans le cadre de ce projet.

**Partie 4 : Approche supervisée**

Nous allons maintenant passer à la partie supervisée où nous allons entrainer différents modèles à prédire les tags à partir de jeu d’entrainement.

**4.1 Modélisation du problème :**

Afin de construire notre approche supervisée, nous avons choisi de restreindre le nombre de tags sur lesquelles nos algorithmes seront entrainés. Nous allons donc conserver les 80 tags les plus fréquents qui représentent chacun plus de 260 questions dans notre jeu de données. Les autres tags seront supprimés du jeu de données et les questions non liées à ces 80 tags ne seront pas conservées.

**4.1.1 Transformation des données cibles :**

Nous allons utiliser la fonction MultiLabelBinarizer de la bibliothèque sklearn.processing afin de transformer nos listes de tags cibles en vecteurs de la taille du nombre de tags sélectionné où l’appartenance d’une question à un tag est signifiée par la présence d’une valeur 1 pour le tag en question, de 0 dans le cas contraire.

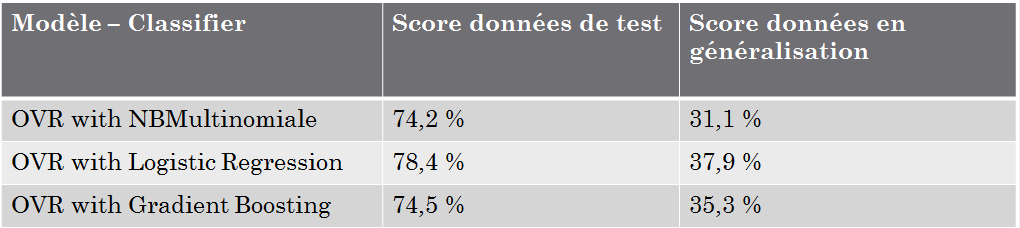
**4.1.2 Construction d’une matrice mot-document à partir de tf-idf :**

De manière similaire à ce qui a été effectué dans l’approche PLSA, nous allons transformer notre corpus de documents en matrice mot-document en utilisant le score tf-idf en coefficient de cette matrice.

**4.2 Résultats obtenus par une approche binary relevance :**

L’approche binary relevance consiste à construire un classifieur binaire par catégories (ici le nombre de tags sélectionné) afin de déterminer une probabilité d’appartenance d’une question à un tag ou aux autres tags (approche One versus Rest OVR). Pour chaque question, nous conserverons les 5 tags les plus probables. Nous avons choisi dans cette approche de comparer 3 classifieurs : NBMultinomial, Logistic Regression et Gradient Boosting.

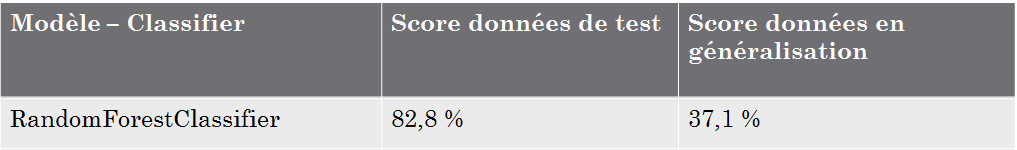
Synthèse des résultats obtenus par cette approche :



**4.3 Résultats obtenus par un classifieur multilabel:**

Nous allons dans cette approche entrainée directement un classifieur intrinsèquement multilabel, notre choix s’est porté sur RandomForestClassifier.

Résultats obtenus par cette approche :



**4.4 Conclusion de notre approche supervisée :**

L’approche supervisée apporte de biens meilleurs résultats en prédiction que l’approche non supervisée. Les meilleurs résultats sont obtenus avec l’approche OVR with Logistic Regression ainsi qu’avec l’utilisation de RandomForestClassifier. Il est probable de pouvoir améliorer les performances des modèles RandomForestClassifier et OVR with Gradient Boosting, cependant cette approche d’optimisation n’a pas pu être mené en raison des temps de calcul très important nécessaire afin d’entrainer ces classifieurs. Nous avons choisi pour notre modèle de retenir le modèle OVR with Logistic Regression qui semble un bon compromis entre performance et temps de calcul.

**Partie 5 : Création d’une API web en utilisant Flask :**

**5.1 Lien vers l’API :**

L’API est disponible à travers le lien : peb33100.pythonanywhere.com/tagging.

**Conclusions et perspectives de ce projet :**

Deux approches ont été développées durant ce travail afin de générer des tags à partir de questions posées sur le site StackOverflow :

* Une approche non supervisée avec l’utilisation de l’algorithme LDA et de l’approche PLSA. Ces deux approches ne sont pas adaptées à la génération de tags et sont plus adaptées à la catégorisation de questions ou à l’exploration des liens sous-jacents entre mots clés.
* Une approche supervisée en modélisant le problème comme un problème de classification multilabel. Deux approches différentes ont été menées. L’approche binary relevance avec un classifieur de type régression logistique semble être un bon compromis entre performance et temps de calcul. Nous avons donc retenu ce modèle pour notre mise en production dans le cadre de notre API.

Il y a de nombreuses possibilités afin de continuer à approfondir ces modèles :

* Le travail sur la sélection du vocabulaire afin de réduire le nombre de stop\_words n’étant pas pertinent est un critère important pouvant nous permettre d’améliorer encore la performance de nos modèles. Cette approche est spécifique à l’application donnée et peut demander un temps conséquent de développement.
* Il serait intéressant également d’explorer la performance de modèle bi-grams en complément des modèles uni-grams développés dans le cadre de ce projet.
* On peut également compléter ce travail en explorant l’utilisation de features liés à la forme des données (nombre de mots dans la question, richesse lexicale…) caractéristiques de la forme des données remplis par l’utilisateur.
* D’autres approches plus sophistiquées (approche ensembliste) peuvent être développées en termes de classification multilabel afin d’améliorer la performance en prédiction (DRAKEL, ORAKEL …), il serait intéressant d’explorer le gain en performance en utilisant de telles méthodes.