StackOverflow est un site de questions-réponses liées au développement informatique. Aidons-les à améliorer leur plateforme !

Catégoriser automatiquement des questions

Projet 6 : Parcours Data Scientist

OpenClassRooms - Supélec

Adil YADA 2018 - 2019

Table des matières

Introduction

StackOverflow est un site de questions-réponses liées au développement informatique. Pour poser une question sur ce site, il faut entrer plusieurs tags de manière à retrouver facilement la question par la suite.

L'objectif du projet que nous devons réaliser est de prédire les tags associés à une question posée par un utilisateur de la plateforme. Notre travail devra prendre la forme d'un algorithme de Machine Learning qui assignera automatiquement plusieurs tags pertinents à une question.

La première partie de ce rapport sera consacrée à l'exploration des données et à la préparation de la pipeline. La deuxième et la troisième parties seront consacrées aux classifications non supervisées et supervisées. La dernière partie consistera en une étude comparative des différents résultats afin de nous aider à choisir le modèle le plus convaincant.

Environnement technique

Système d'exploitation : Windows 10

Version Python : 3.7

Librairies utilisées : pandas, numpy, scikit-learn nltk, pyLDAvis, mglearn, seaborn, matplotlib, bs4 (beautifulsoup)

Notebook exploité à l'aide de Jupyter Notebook.

Partie I : Exploration des données et préparation de la pipeline

I. Prétraitement des données

1. Chargement des données

Avant de pouvoir charger les données, nous avons cherché tout d'abord à récupérer une base de questions sur **Stackoverflow** à l'aide de commandes SQL.
Un point important de cette étape est toutefois à noter. En effet, il ne suffit pas de récupérer une série de questions classées par ordre chronologique mais plutôt un échantillon varié et représentatif sur plus de 5 ans pour pouvoir entrainer correctement nos données et prédire avec le plus de fidélité les tags associés. Ainsi, une question portant sur une technologie récente aura plus de chance de donner un résultat satisfaisant.
Pour cela, nous avons importé une dizaine de fichiers .csv que nous avons joints pour obtenir notre dataframe de travail que l'on nommé « **df_raw** ».

a) Une première visualisation

Notre Dataframe de travail contient 147000 lignes et 22 colonnes.

Voici les colonnes :

['Id','PostTypeId','AcceptedAnswerId','ParentId','CreationDate','DeletionDate','Score','ViewCount','Body','OwnerUserId','OwnerDisplayName','LastEditorUserId','LastEditorDisplayName','LastEditDate','LastActivityDate','Title','Tags','AnswerCount','CommentCount','FavoriteCount','ClosedDate','CommunityOwnedDate']

b) Sélection des données utiles à la problématique

Notre sujet étant limité aux questions et aux tags, il n'était pas nécessaire de conserver les colonnes autres que : 'Title', 'Tags' et 'Body'
Notre dataframe a donc été donc réduite à 3 colonnes.

c) Nettoyage de la dataframe

- Nous avons traité les TAGS manquants en supprimant tout simplement les lignes qui n'en n'avaient pas.
- Nous avons traité les doublons en supprimant tout simplement les lignes ou l'élément « Body » apparait en double.

2. Traitement du texte

Nous avons appliqué deux types de traitement aux colonnes Body et Title. L'un est simple et nécessite peu de ressources et l'autre avancé est plus gourmand en mémoire mais permet un nettoyage efficace.

a) Traitement simple

Fonction **TEXT_PROCESSING( )**

Nous avons développé une fonction qui permet de :

- Nettoyer le code HTML présent (librairie BeautifuSoup)

- Retirer ce qui n'est pas une lettre de l'alphabet Latin
- Réduire en minuscule le texte

b) Traitement avancé

Fonction **TEXT_PROCESSING_ADVANCED( )**

- Nettoyer le code HTML présent (librairie BeautifuSoup)
- Retirer ce qui n'est pas une lettre de l'alphabet Latin
- Réduire en minuscule le texte
- Convertir en minuscule et découpage du texte
- Charger et retrait des stopwords dans le texte
- Lemmatisation des mots

3. Deuxième traitement du texte (concaténation)

Nous avons décidé de concaténer les colonnes Body et Title pour ne former qu'un unique texte que l'on nommera Body_Title.

4. Vectorization en BOW + traitement TF-IDF

a) TF-IDF Vectorizer (seuil)

Nous avons utilisé le la fonction **TFIDFVECTORIZER( )** afin d'obtenir nos BOW et de les transformer ensuite en utilisant une seule fonction. Nous avons généré une matrice sparse et décidé de supprimer les mots qui sont apparus moins de 20 fois.

b) Dictionnaire

Nous avons limité à 10000 mots la taille de notre dictionnaire.

5. Exemples

Texte brut :

'Making Development Better with Static Analysis I find, when I\'m reviewing VBA code, that most of the bugs are easily classified errors, such as typos, not setting things to <code>Nothing</code>, not closing DB connections, etc. \n\n I know that using "Option Explicit" can solve some of these, but I\'d rather use something more powerful. \n\n Are there any static analysis tools for VBA? \n'

Après text_processing simple :

making development better with static analysis i find when i m reviewing vba code that most of the bugs are easily classified errors such as typos not setting things to nothing not closing db connections etc i know that using option explicit can solve some of these but i d rather use something more powerful are there any static analysis tools for vba'

Après text_processing avancé :

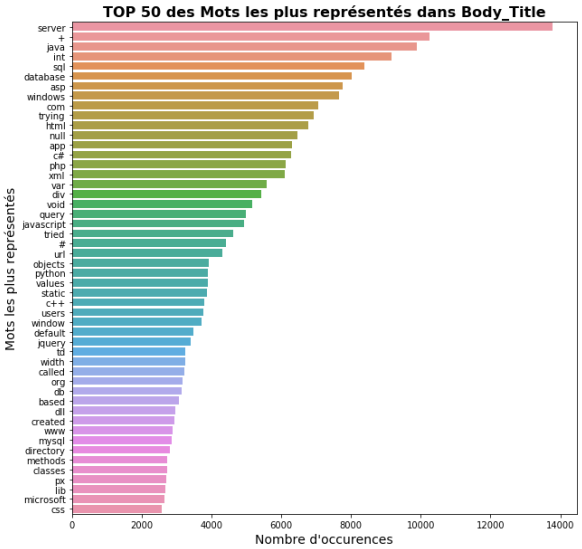
'make static review vba bug classify error typo closing db connection static tool vba'

6. Exploration des données

a) TOP 50 des mots les plus représentés dans notre corpus (avec stopwords)

Nous avons cherché à représenter les mots les plus utilisés dans la feature « Body_Title ».

Pour cela, nous avons retiré les stopwords. Le graphique ci-dessous a été obtenu sur une base text_processing simple.



On remarquera par exemple que certains mots conjugués apparaissent dans ce classement.

b) TOP 50 avec traitement TF IDF (simple et avancé)

De la même façon, nous avons cherché à représenter les mots de notre corpus qui sont les plus représentatifs après leur traitement (TF-IDF) et leur vectorisation.

Principe et utilité

Le TF-IDF permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Ainsi les mots rares (par rapport à l'ensemble des textes) auront un score important relevant ainsi la pertinence de ces mots dans un texte.

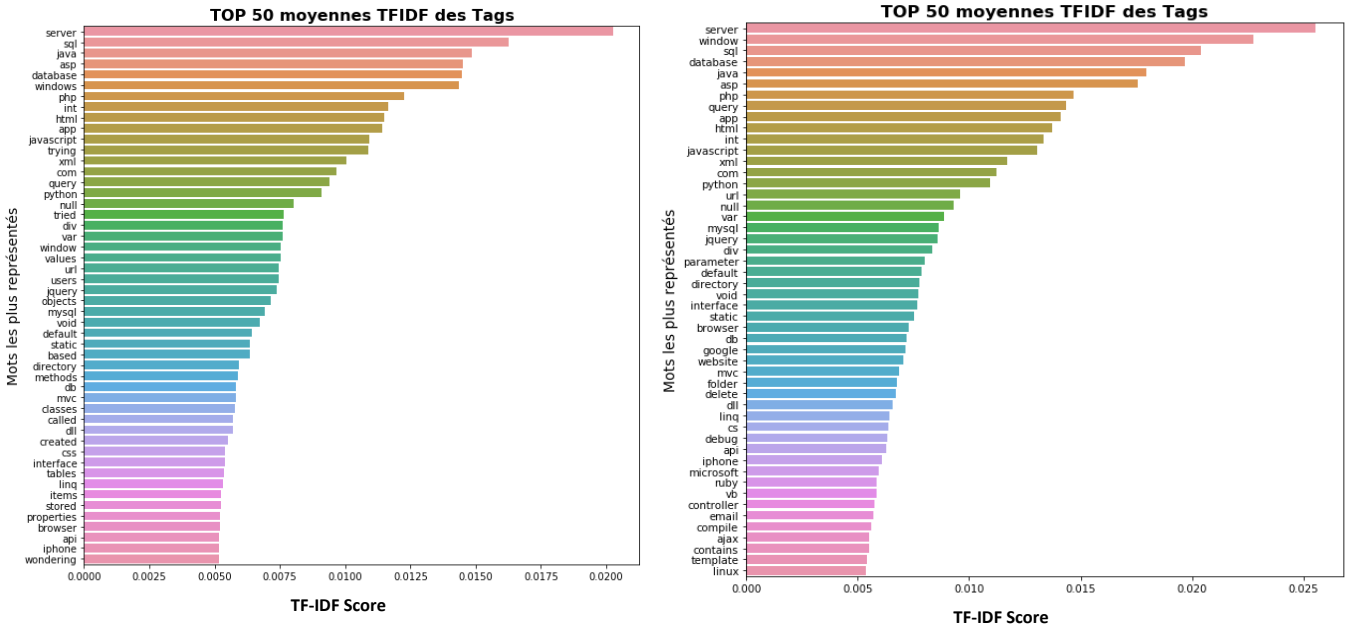


Tableau 1 TOP 50 après traitement par TF IDF (à gauche traitement de texte simple et à droite traitement complexe)

II. Traitement des TAGS

1. Feature engineering

a) Normalisation de l'écriture des TAGS liés aux questions.

Etant regroupés par des caractères « > » et « < », nous avons décidé de remplacer ces caractères par des « / ». Nous avons utilisé pour cela les commandes python **REPLACE( )** et **STRIP( ).** Ensuite, nous avons cherché à récupérer les Tags dans des colonnes afin de pouvoir les exploiter.

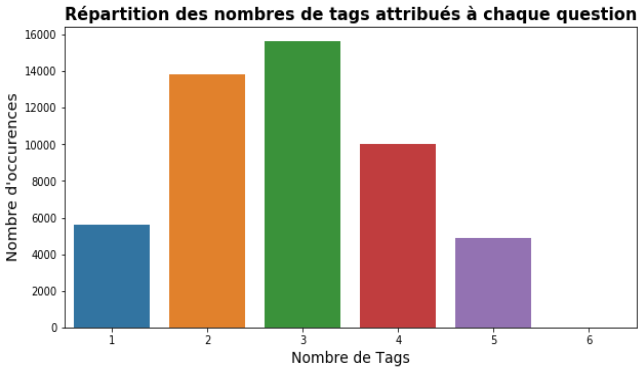
Création de nouvelles features

Nous avons donc :

- ➔ question_tags_featured = ['Tag_1', 'Tag_2', 'Tag_3', 'Tag_4', 'Tag_5']
- ➔ Sur 50000 questions posées nous observons 9098 TAGS différents.

Tableau 2 TOP 10 TAGS

	Tag	Count	Pourcentage (%)
0	c#	7043	4.868153
1	.net	4482	3.097978
2	java	3576	2.471747
3	asp.net	3270	2.260238
4	javascript	2582	1.784690
5	c++	2487	1.719025
6	php	2143	1.481251
7	python	1833	1.266978
8	sql	1702	1.176430
9	sql-server	1638	1.132193



2. Seuil de conservation en % des TAGS

Avec un seuil de 50% en fréquence cumulée croissante, nous ne retenons que les 99 premiers TAGS du classement au lieu des 9098 du départ. Cette troncature nous permet d'effectuer un travail dans de meilleures conditions techniques. Ici le seuil de séparation est la médiane.

III. Préparation de la pipeline

1. Séparation du jeu de données en jeu d'entrainement et jeu de test

Nous avons séparé nos données d'entrainement de nos données de test à l'aide de la commande **TRAIN_TEST_SPLIT().** Nous avons retenu 25% pour le jeu de test et 75% pour les jeux d'entrainement.

a) Feature engineering

Nous souhaitons retirer les tags pour chaque question qui ne sont pas parmi le top 99. Pour cela, nous avons comparé les tags avec ceux retenus. Nous avons obtenu une nouvelle feature « New_tags » qui sera plus ou moins renseignée selon le seuil que l'on aura renseigné avant.

En voici un exemple :

	Tags	New_tags
98052	java/php/html/applet	java/php/html
54118	python/postgresql/performance/dynamic-data/vacuum	python/performance
92683	asp.net/linq	asp.net/linq
143090	silverlight/events/user-interface	silverlight/events/user-interface
57647	exception/exception-handling/outlook/add-in	exception
25144	windows/wmi/admin/ipconfig	windows
33268	php/mysql/version-control	php/mysql/version-control
106414	sql/mysql/database/join	sql/mysql/database

Tableau 3 New Tags/ Old Tags

b) Matrice sparse (bow)

Nous avons appliqué le traitement sur les données d'entrainement pour obtenir la matrice sparse d'entrainement :

X_train, vectorizer = data_preprocessing(data_train)

Nous avons appliqué le vectorizer d'entrainement sur les données test pour obtenir la matrice sparse de test :

X_test = data_preprocessing_test(data_test,vectorizer)

2. Obtention de variables cibles (y_train et y_test)

Une fois que nous avons obtenu les Tags dans notre colonne « New_tags » nous avons pu les placer dans autant de colonnes que nous avons de Tags sélectionnés. Nos variables ou étiquettes sont donc prêtes à être exploitées.
La structure de nos données est à présent adaptée pour de la classification supervisée/non supervisée sur du Multi Label.

3. Dimensions

Nous obtenons les dimensions suivantes pour nos matrices :
PRINT(X_TRAIN.SHAPE) (32271, 2533)
PRINT(X_TEST.SHAPE) (10757, 2533)
PRINT(Y_TRAIN.SHAPE) (32271, 99)
PRINT(Y_TEST.SHAPE) (10757, 99)

4. Sauvegarde des fichiers

Nous avons sauvegardé l'ensemble des données jugées utiles pour la suite et les avons placées dans un répertoire adapté que nous avons nommé « pipeline ».

Conclusion de la Partie I

Nous avons obtenu différentes matrices (training, test…) au bon format pour l'entrainement de nos différents modèles qu'ils soient supervisés ou non supervisés.

Les Fichiers modules.py et pipeline.py sont à exécuter dans votre environnement de travail. Vous pouvez visualiser les différentes étapes de cette partie ainsi que le code associé sur le notebook **« PROJET 6_EXPLORATION ET PIPELINE.IPYNB ».**

Partie II : Modèles prédictifs non supervisés

La classification non supervisée dans le cadre de notre étude sera limitée à la méthode LDA (Latent Dirichlet Allocation).

Notre objectif de classer le plus correctement possible des questions dans la bonne catégorie, sera aidé par l'existence des tags réelles et donc de variables cible. Cette approche aidée aura pour conséquence d'exploiter des métriques connues et d'en implémenter d'autres plus spécifiques.

I. Classification non supervisée – LDA

1. Chargement des données

Nous rechargeons l'ensemble de notre pipeline constitué de nos données d'entrainement, de test ainsi que d'autres s'y rattachant comme les tags sélectionnés ou encore l'objet « vectorizer ».

2. Mise en Place du LDA sur Jeu d'entrainement et visualisation des topics

Nous avons décidé de choisir pour commencer 15 topics. Pour ce faire, nous avons fait appel à scikit-learn ainsi qu'à la matrice X_train obtenue dans notre première partie.

Visualisons les topics et les mots importants qui les caractérisent.

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9
Topic 0	javascript	html	jquery	div	cs	ajax	browser	var	width	php
Topic 1	asp	mvc	controller	url	ii	config	aspx	website	redirect	server
Topic 2	delete	tab	gridview	dataset	queue	grid	datatable	listview	leak	cursor
Topic 3	window	python	dll	directory	folder	debug	sharepoint	eclipse	perl	module
Topic 4	java	int	static	void	interface	char	pointer	byte	constructor	null
Topic 5	unix	bash	jsp	sync	strut	dir	invoke	disadvantage	clr	terminal
Topic 6	iphone	app	asp	animation	cocoa	ui	timer	sender	delegate	nsstring
Topic 7	excel	algorithm	hash	byte	graph	integer	vba	encrypt	math	csv
Topic 8	ruby	flex	dispose	gem	binary	namespace	mx	belongs	typed	obj
Topic 9	cache	nhibernate	wpf	socket	dictionary	ip	refresh	popup	mapping	disable
Topic 10	regex	validation	td	textbox	parse	validate	parser	tr	th	winforms
Topic 11	django	graphic	widget	datagridview	matrix	mock	bitmap	apache	tomcat	pixel
Topic 12	sql	database	server	mysql	php	query	db	oracle	delphi	silverlight
Topic 13	xml	linq	null	query	sql	int	node	datetime	binding	var
Topic 14	server	wcf	svn	email	password	login	com	authentication	repository	subversion

Tableau 4 Mots clés de chaque topic

Nous voyons nettement les tendances et il serait aisé d'attribuer un titre à chacun des topics.

Visualisons maintenant comment se répartissent les premiers textes dans les différents topics.

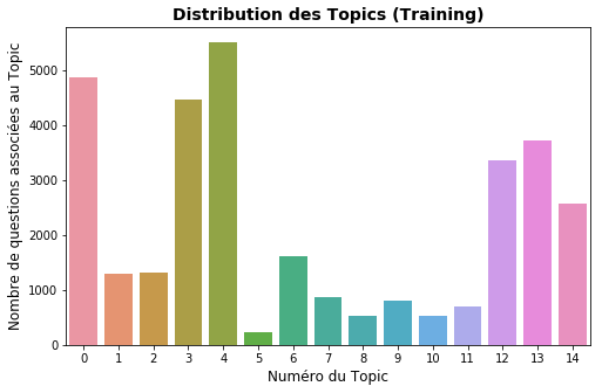
	Topic0	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	Topic10	Topic11	Topic12	Topic13	Topic14	max_proba_topic	dominant_topic
Body Title N°0	**0.174**	0.019	0.019	0.019	**0.116**	0.019	0.019	**0.371**	0.019	0.019	0.019	0.019	0.019	0.019	**0.135**	**0.371**	**7**
Body Title N°1	0.021	0.021	**0.421**	**0.237**	0.021	0.021	0.021	0.096	0.021	0.021	0.021	0.021	0.021	0.021	0.021	**0.421**	**2**
Body Title N°2	0.017	0.017	0.017	0.017	0.017	0.017	0.017	0.017	0.017	0.017	0.017	0.017	0.017	**0.768**	0.017	**0.768**	**13**
Body Title N°3	0.016	0.016	**0.137**	**0.526**	0.016	0.016	0.016	0.016	0.016	0.016	0.016	0.063	**0.101**	0.016	0.016	**0.526**	**3**
Body Title N°4	0.013	0.026	**0.242**	**0.336**	**0.207**	0.013	0.026	0.013	0.025	0.013	0.013	0.013	0.037	0.013	0.013	**0.336**	**3**
Body Title N°5	0.014	0.014	0.014	0.014	**0.442**	0.014	0.014	0.014	0.014	0.014	0.014	0.014	**0.107**	0.014	**0.282**	**0.442**	**4**
Body Title N°6	0.016	0.016	0.071	0.016	0.016	0.016	0.016	0.016	0.016	0.078	0.016	0.016	**0.657**	0.016	0.016	**0.657**	**12**

Tableau 5 Probabilités pour chaque texte d'appartenir à un topic (15 topics)

Nous voyons apparaitre un Topic dominant pour chacun d'entre eux.

3. Distribution des Topics pour l'ensemble du corpus d'entrainement

Observons la répartition des textes dans nos topics à l'aide du tableau suivant :



4. Récupération des Tags associés à chaque topic et les mots associés à chaque topic

Nous obtenons le tableau suivant limité à 5 tags :

	LDA Topic	Tags associés	Topics_group
0	0	[c#, .net, java, c++, iphone]	int/void/static/interface/null
1	1	[c#, .net, windows, c++, javascript]	windows/window/delphi/console/keyboard
2	2	[c#, asp.net, .net, winforms, wpf]	asp/controls/excel/textbox/vb
3	3	[c#, php, c++, .net, windows]	keyword/logging/replication/addresses/curl
4	4	[java, c++, c#, python, c]	java/int/perl/xml/php
5	5	[c#, .net, asp.net, java, php]	server/asp/app/windows/mvc
6	6	[flex, c#, xml, .net, actionscript-3]	xml/mx/serialize/serialization/obj
7	7	[sql, sql-server, c#, mysql, php]	sql/database/query/mysql/server
8	8	[java, python, linux, eclipse, macos]	python/eclipse/java/os/lib
9	9	[c, sockets, c#, java, c++]	socket/decimal/matrix/sockets/extract
10	10	[iphone, c#, jquery, cocoa-touch, javascript]	iphone/timer/cocoa/dialog/xcode
11	11	[c#, c++, .net, winapi, c]	dll/node/nodes/callback/threads
12	12	[javascript, css, jquery, html, asp.net]	javascript/div/jquery/html/css
13	13	[java, flex, flash, c#, email]	flex/email/pdf/swf/actionscript
14	14	[wpf, c#, silverlight, .net, data-binding]	wpf/silverlight/xaml/binding/listview

Tableau 6 Nos deux types de Tags prédits par topic

II. Prédictions sur jeu de test - LDA

Nous avons ensuite cherché à prédire les tags associés à chaque texte.

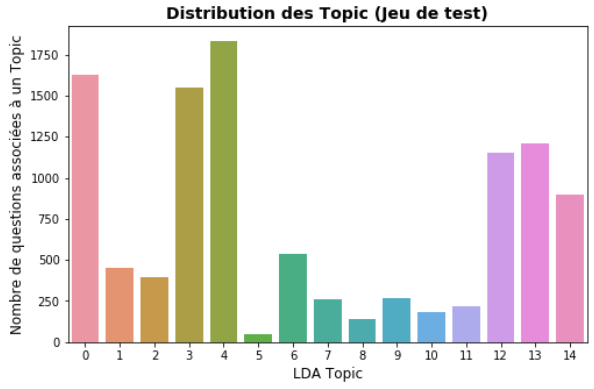
1. Transformation des données

Cette fois-ci nous travaillons sur X_test. Nous avons transformé les données comme suit :

```
LDA_OUTPUT_TEST = LDA.TRANSFORM(X_TEST)
```

2. Prédictions et classifications

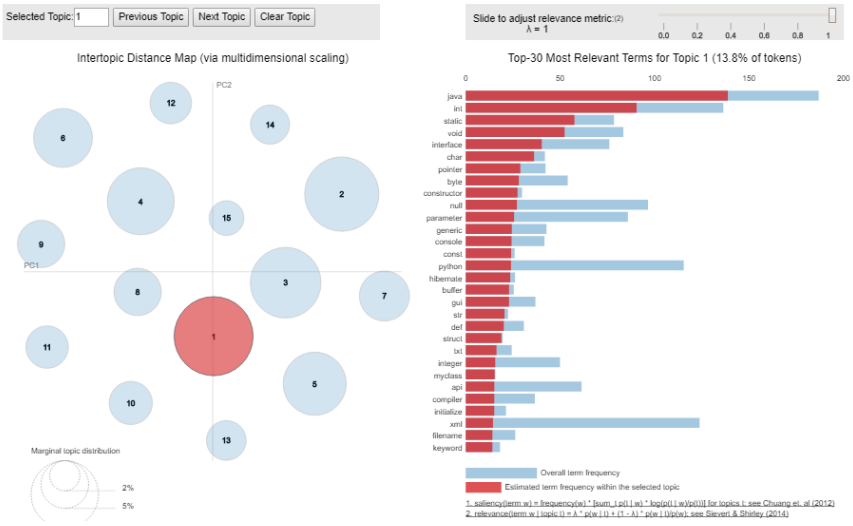
Pour nos données de test nous obtenons la nouvelle répartition suivante :



3. Représentations

Nous avons utilisé la librairie pyLDAavis afin d'améliorer la visualisation des topics et leur compréhension.

Figure 1 Représentation avancée des Topics



4. Nos métriques

Nous avons testé 2 approches chacune pour un type de tag prédit. La première utilisant les tags des utilisateurs et la deuxième créant ses propres tags depuis le texte.

a) Local_Recall et autres métriques

Nous avons souhaité obtenir une métrique qui permet de calculer le rapport du nombre de documents correctement attribués à la classe i par le nombre de documents appartenant à la classe i.

Cette métrique est tout simplement celle que l'on appelle « Recall » ou « « Rappel ».

Première approche

Nous obtenons le tableau et les résultats suivants :

- Local-Recall =0.45, F1-Score=0.197, Précision=0.137

	Predicted_Tags	True_Tags	Recall
2739	window/wpf/gui/dialog/winforms	java/web-services	0.000000
956	sql/database/server/query/asp	ruby-on-rails	0.000000
1725	int/void/static/null/char	.net	0.000000
9863	javascript/html/jquery/div/cs	asp.net/css	0.000000
2289	javascript/html/jquery/div/cs	visual-studio	0.000000
10006	int/void/static/null/char	python/django	0.000000
8934	python/directory/folder/dll/debug	c++/visual-studio/winapi	0.000000
2543	javascript/html/jquery/div/cs	asp.net/jquery/web-services	0.333333
2790	iphone/nhibernate/silverlight/delphi/app	winforms/multithreading/debugging	0.000000
5165	int/void/static/null/char	oracle	0.000000
1080	javascript/html/jquery/div/cs	c++	0.000000

Tableau 7 Tags prédits avec 1ère approche

Deuxième approche

Nous obtenons le tableau et les résultats suivants : Local-Recall =0.15, F1-Score = 0.09,

Précision=0.06

	Predicted_Tags	True_Tags	Local_Score
10639	sql-server/sql/php/mysql/database	php/mysql	1.0
10556	javascript/jquery/html/css/asp.net	c++/c	0.0
5078	iphone/asp.net/c#/cocoa-touch/objective-c	iphone	1.0
6879	javascript/jquery/html/css/asp.net	css	1.0
110	asp.net/c#/svn/.net/wcf	c++	0.0
4261	sql-server/sql/php/mysql/database	sql-server/sql-server-2005	0.5
1049	c#/c++/.net/python/windows	visual-studio/sharepoint	0.0
454	c#/c++/.net/python/windows	visual-studio-2008/visual-studio-2005	0.0
5954	iphone/asp.net/c#/cocoa-touch/objective-c	sharepoint	0.0
5984	asp.net/c#/svn/.net/wcf	php/email	0.0
7655	c#/c++/.net/python/windows	python	1.0
5407	java/c#/c++/.net/c	performance/oop	0.0
10164	c#/excel/.net/java/algorithm	design	0.0
9383	sql-server/sql/php/mysql/database	sql-server	1.0

Tableau 8 Tags prédits avec 2ème approche

5. Remarques

Les résultats sont très moyens pour la première approche mais très insuffisants pour la deuxième approche.

Conclusion de la Partie II

Les résultats sont moyens sur notre première approche mais très médiocres pour la deuxième approche.

Bien entendu, lors de la majorité des cas où la classification non supervisée sera utilisée, il ne sera pas possible de calculer une métrique car nous n'aurons pas d'étiquette à proprement dit. Il faudra faire « confiance » au modèle.

Partie III : Modèles prédictifs supervisés

I. Classification MULTI LABEL

1. Principes

Nous devons pouvoir prédire désormais les tags associés à une question uniquement à partir des mots qui sont contenus dans la question (vectorisés en bag of words avec transformation TF-IDF).

2. Procédé

Nous avons utilisé un modèle linéaire pour chacun des tags sélectionnés de notre liste.
Nous avons entrainé pour cela nos données autant de fois qu'il y avait de tags présents dans notre liste.

Dans notre exemple, nous avons une liste de 99 tags et une fois les entrainements effectués, nous avons obtenu les prédictions des probabilités pour chacun d'eux. Nous avons ensuite procédé par le choix des 5 tags les plus probables.

II. Régression logistique

Nous entrainons nos données sur chaque tag et obtenons les prédictions de probabilités à l'aide de la fonction **PREDICT_PROBA( ).**

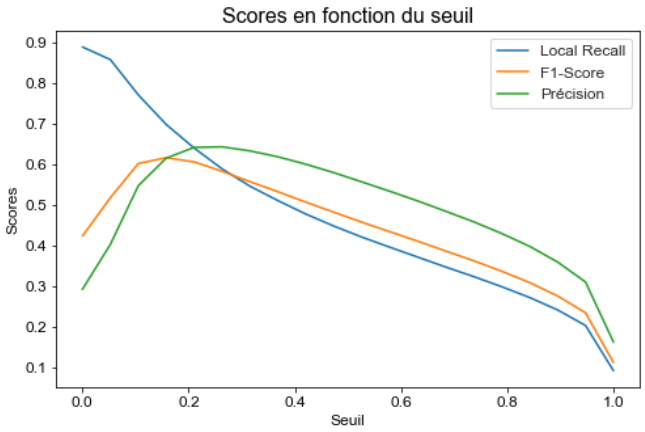
Nous avons obtenu ainsi 99 classifieurs pour la régression logistique qui ont été classés dans le répertoire « classifiers ». Pour éviter d'obtenir des prédictions forcées, nous avons introduit un seuil en dessous duquel les probabilités prédites seront jugées trop faibles pour donner un résultat concluant.

Tag	c#	.net	java	asp.net	javascript	c++	php	python	sql	sql-server	...
0	0.182712	0.125255	0.106742	0.016897	0.006879	0.076015	0.025468	0.039617	0.014156	0.009240	...
1	0.141774	0.092622	0.050384	0.032923	0.025271	0.004880	0.033910	0.024412	0.014780	0.010133	...
2	0.018436	0.009450	0.006205	0.007738	0.430942	0.011832	0.005393	0.003398	0.003574	0.001975	...
3	0.055833	0.039587	0.609938	0.042551	0.043096	0.033340	0.013837	0.008979	0.009474	0.005455	...
4	0.161620	0.052129	0.703448	0.008660	0.005670	0.086124	0.019063	0.018683	0.007572	0.014345	...
5	0.126753	0.046175	0.019528	0.014697	0.012237	0.019959	0.026948	0.018072	0.014827	0.006286	...
6	0.223264	0.084167	0.006702	0.808191	0.030102	0.006829	0.004742	0.004659	0.016736	0.007619	...
7	0.125379	0.055712	0.015203	0.101495	0.006541	0.003561	0.026467	0.003170	0.003070	0.004114	...

Tableau 9 Prédiction des probabilités (Régression logistique)

1. Choix du Seuil

Afin d'obtenir le meilleur seuil possible nous avons maximisé le F1-Score à l'aide du graphe suivant :



Après exploitation, nous avons obtenu les résultats suivants :

	True_Tags	Predicted_Tags	Local_Recall
6975	svn/version-control	svn/version-control	1.000000
1041	java	java	1.000000
10602	flex	flex/actionscript-3	1.000000
4174	javascript/jquery	html/jquery	0.500000
6963	mysql/sql	.net	0.000000
9909	php/ajax	php/javascript	0.500000
7848	javascript/events	javascript	0.500000
10358	php/regex	regex/php	1.000000
1516	ruby-on-rails/ruby	c#/.net	0.000000
6230	asp.net-mvc	asp.net-mvc/asp.net	1.000000
6620	java	java/c#	1.000000
872	c#/.net	html	0.000000
5474	c#/.net/multithreading	multithreading/c#/.net	1.000000
9524	c#	c#/.net	1.000000
4670	java/xml	java/xml/c#	1.000000

Tableau 10 Tags prédits avec Régression logistique

F1-Score le plus élevé = 0.53, seuil retenu = 0.157, recall moyen associée = 0.61, précision associée = 0.54

Les résultats sont assez bons dans l'ensemble.

III. Naive Bayes

Nous avons appliqué la même méthode pour le modèle Naive Bayes et nous avons obtenu :

Recall moyen = 0.57, F1-Score = 0.462, précision = 0.448

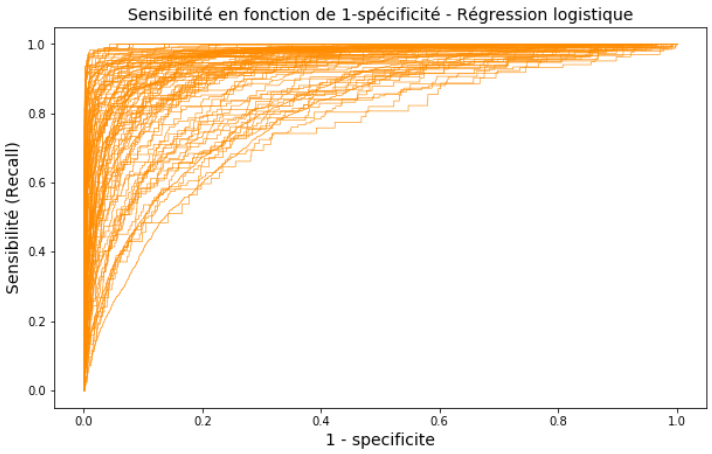
IV. Random Forest

Nous avons appliqué la même méthode que pour le modèle précédent et nous avons obtenu pour 10 arbres :

Recall moyen = 0.42, F1 Score = 0.366, Précision = 0.259

V. Courbes ROC

Nous avons obtenu les courbes ROC pour la régression logistique sur les 99 tags qui constituent nos labels à prédire.



La moyenne **auc** est de 94%.

Conclusion Partie III

Nous avons pu constater que l'on pouvait maximiser le Recall, F1-Score ou la précision afin de trouver le meilleur seuil suivant ce que nous cherchons.

Dans notre cas, le F1-Score est le bon équilibre à privilégier. Nous avons pu constater également que les résultats obtenus avec la Régression Logistique sont meilleurs que ceux obtenus avec les autres modèles étudiés.

Partie IV : Analyse et comparatif des résultats

Cette partie est consacrée à l'analyse et à la comparaison des résultats suivant différents paramètres.

1. 99 Tags retenus (premiers 50%)

a) Classification non supervisée

		Text_processing simple	Text_processing avancé
10 topics + Entrainement sur 50000 questions	**F1_score**	0.187	0.202
	Précision	0.128	0.139
	Recall	0.40	0.43
15 topics + Entrainement sur 50000 questions	**F1_score**	0.198	0.216
	Précision	0.137	0.149
	Recall	0.42	0.46
20 topics + Entrainement sur 50000 questions	**F1_score**	0.197	0.219
	Précision	0.136	0.151
	Recall	0.42	0.47
30 topics + Entrainement sur 50000 questions	**F1_score**	0.198	0.225
	Précision	0.137	0.145
	Recall	0.42	0.48

Tableau 11 résultats Classification non supervisée (LDA) - 99 tags retenus

b) Classification Supervisée (régression logistique avec seuil optimisé)

		Text_processing simple	Text_processing avancé
Entrainement sur 50000 questions	**F1_score**	**0.545**	0.53
	Précision	**0.552**	0.54
	Recall	**0.62**	0.61

Tableau 12 Classification supervisée (Régression logistique) – 99 tags retenus

c) Classification Supervisée (Naive Bayes avec seuil optimisé)

		Text_processing simple	Text_processing avancé
Entrainement sur 50000 questions	**F1_score**	0.437	0.462
	Précision	0.433	0.448
	Recall	0.53	0.57

Tableau 13 Classification supervisée (Naive Bayes) – 99 tags retenus

2. 550 Tags retenus (Q3 - 75%)

a) Classification non supervisée

		Text_processing simple	Text_processing avancé
10 topics + Entrainement sur 50000 questions	**F1_score**	0.157	0.171
	Précision	0.117	0.128
	Recall	0.27	0.30
15 topics + Entrainement sur 50000 questions	**F1_score**	0.157	0.173
	Précision	0.118	0.129
	Recall	0.27	0.30

20 topics + Entrainement sur 50000 questions	**F1_score**	0.154	0.184
	Précision	0.116	0.138
	Recall	0.26	0.32
30 topics + Entrainement sur 50000 questions	**F1_score**	0.156	0.190
	Précision	0.117	0.142
	Recall	0.27	0.33

Tableau 14 Classification non supervisée (LDA) - 550 tags retenus

b) Classification Supervisée (régression logistique avec seuil optimisé)

		Text_processing simple	**Text_processing avancé**
Entrainement sur 50000 questions	**F1_score**	0.472	0.464
	Précision	0.488	0.469
	Recall	0.54	0.54

Tableau 15 Classification supervisée (Régression logistique) - 550 tags retenus

c) Classification Supervisée (Naive Bayes avec seuil optimisé)

		Text_processing simple	**Text_processing avancé**
Entrainement sur 50000 questions	**F1_Score**	0.333	0.352
	Précision	0.234	0.409
	Recall	0.429	0.37

Tableau 16 Classification supervisée (Naive bayes) - 550 Tags retenus

3. Remarques

Le F1-Score est calculé comme suit à partir des précisions et rappels (recall)

$$F = 2 \cdot \frac{(\text{précision} \cdot \text{rappel})}{(\text{précision} + \text{rappel})}$$

- A travers les résultats obtenus, nous pouvons privilégier la **méthode supervisée** de type **régression logistique**.
- Clairement, la méthode **non supervisée** ne donne pas de bons résultats. Ainsi, pour un recall élevé la précision sera assez faible. Le F1-Score sera assez faible quel que soit le nombre de topic choisis. On remarque que cette méthode est plus sensible cependant au text processing avancé qui a plutôt tendance à augmenter les scores obtenus lorsqu'il est appliqué.
- Il est à noter également que le **text processing avancé** ne produit pas nécessairement de meilleurs résultats. Cela montre que le TF-IDF est assez efficace pour retirer les mots qui ne sont pas importants.
- On remarque aussi clairement que si on augmente le nombre de tags sélectionnés les résultats sont moins bons, ce qui est assez logique car si le choix de tags diminue, l'algorithme a moins de chances de se « tromper ». Retenir 99 tags (ceux qui représentent 50% des apparitions) est un bon compromis.

➔ Nous avons maintenant les bons paramètres pour notre site web.

4. Livrables :

Vous trouverez notre site web à l'adresse suivante : http://leftyboy.pythonanywhere.com

Pour visualiser notre travail vous pouvez vous rendre à l'adresse suivante :
https://github.com/leftyboy/Projet-6

Conclusion du rapport

Le modèle choisi pour notre API est le modèle de régression logistique dans le cadre d'une classification supervisée. Nous avons tenté d'être le plus rigoureux possible sur la phase de nettoyage des données car celle-ci nécessite une attention particulière pour l'entrainement des données. Cependant, cette tâche peut prendre un temps important et peut donner l'impression d'être inutile voire tirer les résultats vers le bas au-delà d'un certain point.

Il est à noter que faire appel à la transformation TF-IDF peut faciliter grandement la tâche et réduire le temps de traitement du texte en amont.