Rapport Projet 5

Catégorisez automatiquement des questions

Table des matières

Introduction

Le système de marquage (Tagging) s'avère l'un des meilleurs outils de catégorisations textuelles de nos jours. Cela nous permet de révéler les mots clés ainsi que les informations utiles à partir d'un texte.

Stack Overflow, qui représente l'un des meilleurs forums d'entraide pour toutes les thématiques de développement informatiques de nos jours. Offre la possibilité à ses utilisateurs d'échanger sur des difficultés rencontrées dans la programmation, ce qui permet d'enrichir la base de données de Stack Overflow.

Ces informations peuvent être exploitées pour concevoir des systèmes robustes et automatisés qui créent des tags pour les questions posées. Le marquage automatisée de ces questions permet au moteur de recherché et système de recommandation à extraire des informations sur des questions similaires en fonctions des tags associés.

Stack Overflow propose un outil d'export de données qui est le "**Stackexchange Explorer**", qui recense un grand nombre de données authentiques de la plateforme d'entraide. A l'aide d'une requête SQL on va extraire les données à exploiter. Pour entamer l'étude d'un point de vue métier, on a filtré les inputs d'une façon à n'avoir que les questions pertinentes.

Notre étude comporte trois grandes parties :

- **Pre-processing des questions** : pour la partie prétraitement des données, nous traiterons les données en supprimant les balises html courantes, la ponctuation, les mots alphanumériques et les mots vides. On va rajouter aussi, une étape de Lemmatisation en faisant filtrer les STOPWORDS

- **Extraction des topics** : En faisant recours à la méthode non supervisée LDA du package Gensim, on va essayer d'extraire les topics cachés dans l'ensemble de corpus. Suivi par une analyse exploratoire des résultats

- **Prédiction des tags** : dans cette partie, on découvre la classification en multi-label. Le but serait de prédire l'ensemble des tags pour une question donnée. Avec des méthodes supervisées on va essayer d'optimiser nos métriques d'évaluations

I. Préparation des données

Cette partie sera dédiée à la préparation de notre jeu de données finale :

- Récupération des données : à l'aide d'une requête SQL à partir de Stack-Exchange Explorer

- Nettoyage de l'ensemble des données

- Analyse exploratoire du corpus

Jeu de données

L'explorateur de données Stack-Exchange est un outil permettant d'exécuter des requêtes SQL arbitraires sur les données des différents sites de questions et réponses du réseau Stack-Exchange. Pour pouvoir récupérer des données, il suffit d'écrire la requête adéquate à votre demande

Pour notre cas d'étude, on a fait le choix des questions d'un point de vue métier. Avec des questions récentes, bien notées et qui soulèvent un échange entre les utilisateurs.

On a procéder de la façon suivante dans la récupération des données :

- Title & Tags: toutes les questions ont des titres et des tags renseignés
- CreationDate: les données de l'année précédente
- Body_script: Questions contenants les codes
- Score & AnswerCount: Questions les mieux notées et qui ont plus d'une seule réponse
- N_views & Favorite_Count: Questions les plus vus et choisis comme favoris

On a pu retirer en tout prêt de 12000 questions, qui nous semble suffisant pour notre étude

```
#here the SQL Request we used
'''
SELECT * FROM posts
WHERE Title is not null and Title <> '' AND
      Tags is not null and Tags <> '' AND
      CreationDate>=  '2020-04-01' AND CreationDate < '2021-04-01' AND
      Body LIKE '%<code>%' AND
      Score>1 AND AnswerCount>1 AND
      FavoriteCount>0 AND ViewCount>10
      '''
```

Id	Title	Body	Tags
61193847	How to check if a property has been set in Bla...	<p>My biggest issue here is when I have a prop...	<c#><blazor>
61193896	How does Git parse date string?	<p>I have a date string from Joda DateTime.toS...	<git><date><time><jodatime>
61194243	How to resolve No converter found capable of c...	<pre><code>org.springframework.core.convert.Co...	<spring-boot><jpa><spring-data-jpa><nativequery>
61196175	`movaps` vs. `movups` in GCC: how does it decide?	<p>I recently researched a segfault in a piece...	<c++><gcc><optimization><x86><memory-alignment>
61196282	Working out what was matched by each part of a...	<p>I have a few hundred (fairly simple) regula...	<python><regex><grammar><text-parsing>

Traitement des données

En tant que Data Scientist, on fait face dans 90% des cas à des données non nettoyés et non structurées. Parmi ces cas, on a le traitement des données textuelles (articles, compte rendus, descriptions, avis, réseaux sociaux) qui reste un challenge pour les études « Natural Langage Processing ». Un bon nettoyage de données représente le facteur clé pour une meilleure exploration du corpus.

La préparation des données textuelles pour le NLP comporte 2 choses essentielles,

1. **Nettoyage** : les questions, le Body ou le code doit être, corrigés, transformés en format exploitables. On a un gros travail de nettoyage à faire dans les projets de NLP. Je vais lister les étapes des traitements.

2. **Structuration des données**: Mettre notre série de données dans un format exploitable par les algorithmes de ML. On passe à une représentation matricielle de l'ensemble de corpus avec la méthode de Bag of Words (BOW) ainsi que la vectorisation en TF-IDF.

Fonctions de Nettoyages

Ci-dessous les étapes que j'ai suivies pour le nettoyage de l'ensemble de données :

- Substituer/Extraire le code: A partir du Body, on rajoute une donnée textuelles à notre jeu de données en faisant extraire le code renseigné par l'utilisateur
- Balises HTML : Enlever les tags du html avec la librairie BeautifulSoup
- Supprimer Ponctuation et les caractères spéciaux
- Remettre les textes en minuscules
- Remplacer les abréviations de langages par leurs propres annotations
- Enlever les caractères spéciaux, les chiffres associés avec des mots, les lettres uniques
- Supprimer les Stopwords avec le dictionnaire des mots contenu dans le package nltk, en y rajoutant une liste de mots à savoir (Why, want, get, use, Would, Could …)
- Lemmatisation : en dernière étape on effectue la lemmatisation qui sert à renvoyer un mot à sa racine. On va voir que j'effectue deux types de lemmatisation. Une première méthode en prenant en compte tous les pos_tag possibles (VERB, ADV, ADJ, NOUN), et une deuxième méthode en ne considérant que les noms
- Stemming : on a décidé de ne pas effectuer un stemming à la fois avec la lemmatisation, vu que cela modifie le sens des mots

Pipeline de Nettoyage

Une fois on a terminé l'écriture des fonctions, on passe à bâtir les pipelines de Nettoyage.

On va avoir quatre pipelines différents :

- **Pipeline_Body** : - Substitute code
  - Remove html tags
  - Remove punctuation
  - Lemmatisation (stopwords)

  Pipeline_code : - get_code
  - remove punctuation

- **Pipeline_Title** : - Remove punctuation
  - Remove Stop words
  - Lemmatisation (stopwords)

  Pipeline_tags : -replace (‘≤’,’≥’
  - tokenization

Exemple de nettoyage d’une question:

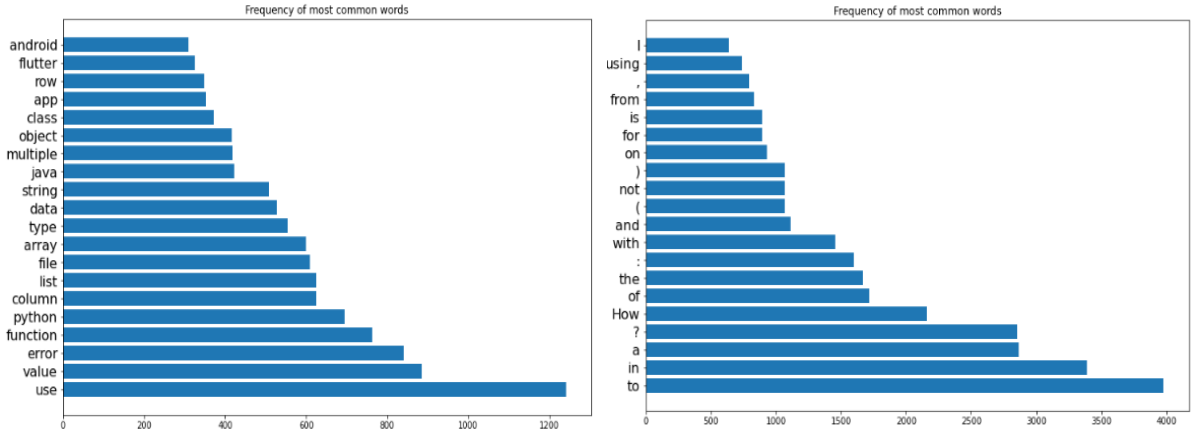
Body originale	I remember reading somewhere that a type like this one can't be <code>Traversable</code>:
	
	<pre><code>data Bar a = Bar a deriving(Show)
	
	instance Functor Bar where fmap f (Bar x) = Bar (f x)
	
	instance Foldable Bar where foldMap f (Bar x) = f x <> f x</code></pre>
	
	 The bit of the explanation I remember is that for <code>foldMap = foldMapDefault</code> to hold, the <code>Traversable</code> instance would have to visit its elements more than once, which a lawful instance can't do. However, I don't remember why a lawful instance can't do that. Consider this one:
	
	<pre><code>instance Traversable Bar where sequenceA (Bar x) = Bar <$ x <*> x</code></pre>
	
	 It looks fine at first glance. What's unlawful about doing that?

Body nettoyé	remember reading somewhere type like one bit explanation remember hold instance visit element lawful instance however remember lawful instance one look fine first glance unlawful
Code extrait	traversabledata bar bar deriving show instance functor bar where fmap bar bar x instance foldable bar where foldmap bar f lt gt x foldmap foldmapdefaulttraversableinstance traversable bar where sequencea bar bar lt lt gt x
Titre originale	Why can't a Traversable visit its elements more than once?
Titre nettoyé	traversable visit element
Tags associés	<haskell><traversable>
Tags nettoyés	haskell traversable

Analyse de corpus

Dans cette partie on va s'intéresser à l'occurrence des mots dans le corpus ainsi que la distribution des tags par questions

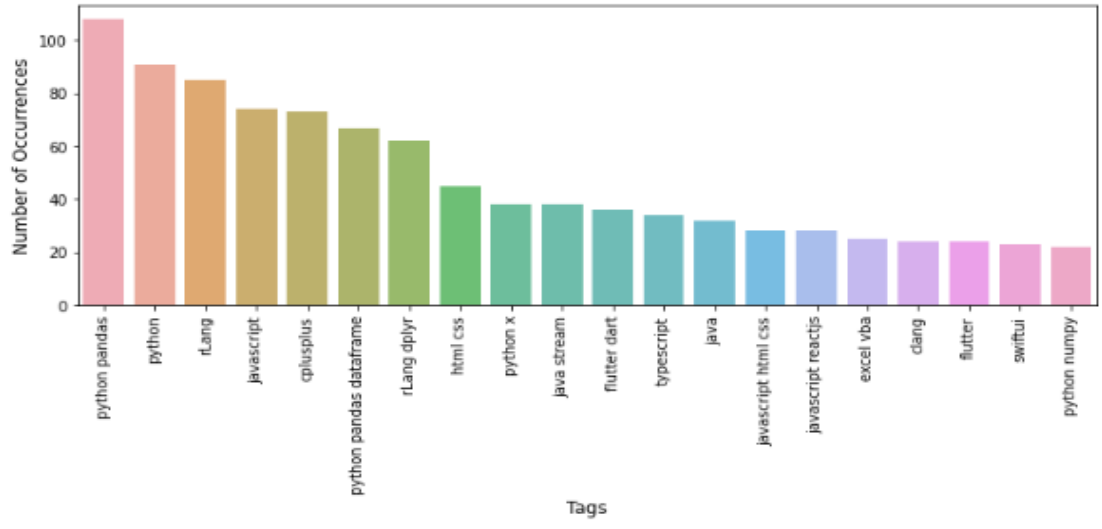
Fréquence des mots



Les deux figures ci-dessus montrent l'importance de supprimer les Stopwords. Des termes qui n'auront aucune influence sur la variance des données vont apparaitre à savoir (to, in, How, and, with …). A gauche on a une liste des mots les plus représentatifs de corpus. Nous remarquons tout d'abord que notre corpus est constitué de 14387 mots différents. On ne pourra garder tous ces mots pour notre BOW ou par la transformation TF-IDF. Dans les prochains chapitres on va expliciter la méthode qu'on a choisie pour garder les mots qui constitue notre corpus.

Les tags les plus utilisés

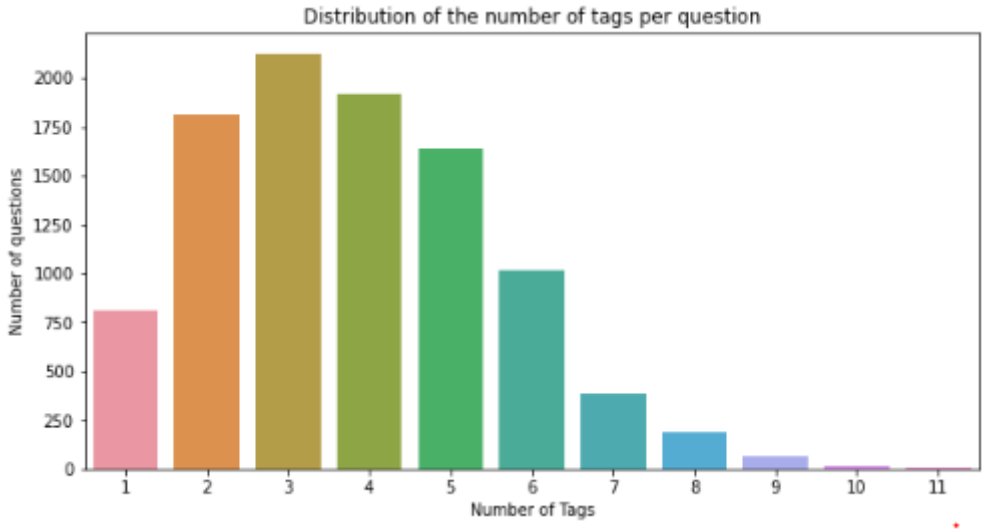
On va regarder de près la distribution des tags avec leur fréquence d'occurrence



- Python, Javascript et le langage R sont les tops sujets abordés par nos utilisateurs
- On remarque que les tags correspondent principalement aux noms des langages de programmation
- L'attribution des tags peut se faire en un, ou plusieurs à la fois

Ensuite, on va voir la distribution des tags par questions

Distribution des tags



On trouve une distribution uniforme des tags :

- La plupart des questions ont un minimum de 2 tags
- On a des questions avec 10 et 11 tags associés !!!

II. Catégorisation des questions : Méthodes non Supervisées

D'une façon générale, le « Topic Modeling » ou le « modèle de sujet » est un modèle probabiliste de l'apprentissage automatique NLP qui permet la détermination des sujets ou thèmes abstraits dans un document.

Considérée comme une méthode non-supervisée, cette technique est capable de numériser un ensemble de documents, analyser l'ensemble de mots et les expressions qu'ils contiennent et de regrouper automatiquement les groupes de mots et les expressions similaires qui caractérisent le mieux un ensemble de documents.

Il existe aujourd'hui, plusieurs algorithmes qu'on peut utiliser pour effectuer la modélisation de sujet. Les plus courants sont :

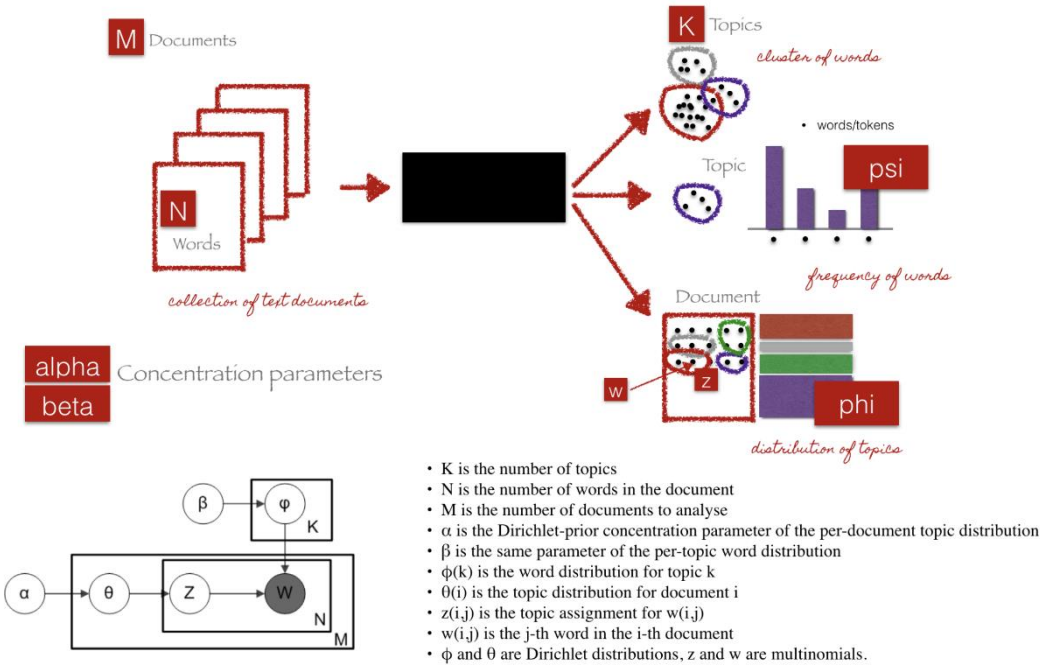
- L'analyse Sémantique Latente (LSA/LSI)
- L'analyse Sémantique Latente probabiliste (pLSA)
- **L'allocation de Dirichlet Latente (LDA)**
- NMF : Factorisation Matricielle Négative

Latent Dirichlet Allocation : LDA

Dans le domaine du NLP, La LDA (de l'anglais Latent Dirichlet Allocation) est un modèle génératif probabiliste permettant d'expliquer des ensembles d'observations, par le moyen de groupes non observés, eux-mêmes définis par des similarités de données.

Cette méthode se repose sur deux hypothèses :

- Les sujets similaires utilisent des mots similaires
- Un document est un ensemble de sujets.



La figure ci-dessus dévoile le fonctionnement de notre méthode.

1. Pour chaque thème J ε {1…, K}, on a la distribution de mots par thèmes avec (β)w
2. Pour chaque document m ε {1…, M} :
    - Tirer la distribution de thèmes θm à partir de (α)K
    - Pour chaque mot d'indice n ε {1…, Nm} dans le document m :
        - Tirer un thème $Z_{m,n}$ à partir du Multinomial $K(\theta_m)$
        - Tirer un mot $W_{m,n}$ à partir du Multinomial (ϕ_{zd})

Modélisation des sujets

Dans cette partie, nous allons utiliser pour l'extraction des sujets le package **Gensim**. Cette bibliothèque est considérée comme la bibliothèque référence pour la modélisation des sujets, l'indexation des documents et la recherche des similitudes avec de grand corpus.

Pour trouver les n sujets les plus importants de nos documents, on a recours à l'algorithme **LDA**. Ces sujets seront décrits par une probabilité de mots clés de nos dictionnaires, avec un poids attribués à chacun. **Gensim** nécessite un format spécifique de données, voici le processus à adopter pour l'obtention des résultats :

- Une liste des documents sous forme de **tokens**, contenue dans une liste
- Entrainer un dictionnaire sur cette liste avec le **modele corpora.Dictionary**
- Préparer le corpus qui consiste à avoir le BOW de chaque ligne dans la liste des documents
- Appliquer la transformation **TF-IDF** à notre corpus, dans le but d'attribuer un poids à chaque mot.

Une fois on a terminé la préparation des données. On passe à la partie d'entrainement de notre corpus avec le modèle qu'on a choisi. Dans notre cas ça serait le **LdaMulticore** qui est une variante plus performante de LDA.

Voici les résultats obtenus avec un **n_topic=5** :

Sujet 1	'0.016*"**file**" + 0.015*"**google**" + 0.015*"**api**" + 0.013*"**git**" + 0.013*"**page**" + 0.012*"**core**" + 0.011*"**aws**" + 0.011*"**container**" + 0.011*"**load**" + 0.011*"**use**"
Sujet 2	'0.026*"**flutter**" +0.014*"**android**" + 0.014*"**app**" + 0.013*"**fail**" + 0.012*"**custom**" + 0.012*"**ios**"+0.012*"**image**" + 0.012*"**laravel**" + 0.011*"**docker**" + 0.011*"**error**"
Sujet 3	'0.033*"**type**"+0.029*"**function**"+0.023*"**class**"+0.017*"**method**" + 0.017*"**error**" + 0.015*"**parameter**" + 0.012*"**module**" + 0.012*"**cplusplus**" + 0.012*"**typescript**"
Sujet 4	'0.030*"**column**" + 0.025*"**value**" + 0.023*"**list**" + 0.020*"**string**" + 0.018*"**row**" +0.017*"**data**"+0.016*"**python**"+0.016*"**rLang**"+0.016*"**dataframe**"+0.016*"**use**"
Sujet 5	'0.017*"**array**" + 0.016*"**java**" + 0.013*"**json**" + 0.013*"**match**" + 0.013*"**package**" +0.012*"**javascript**"+0.012*"**object**"+0.012*"**map**"+0.012*"**result**"0.01*"**swiftui**"

On a choisi la valeur de 5 topic pour pouvoir faire une analyse rapide des sujets extraits, avant de passer à faire un Tuning des hyper-paramètres pour déterminer le nombre de sujets optimaux.

Pour chaque sujet on a les 10 mots les plus probables prédits par le LDA :

- Sujet 1 : web service
- Sujet 2 : pas de sujet spécifique
- Sujet 3 : orienté objet
- Sujet 4 : manipulation des séries de données
- Sujet 5 : pas de sujet spécifique

Tuning des hyper-paramètres

Dans cette partie, on va opter pour une optimisation de nombre de topics. Parmi les hyper-paramètres du modèle LDA, on va s'intéresser à :

- Nombre de sujets demandés
- Alpha : hyper-paramètre de Dirichlet qui correspond à la distribution de thème par document
- Beta : hyper-paramètre de Dirichlet qui nous renseigne sur la distribution de mots par document

On va effectuer une méthode de grid search manuelle, en faisant des itérations sur les différentes combinaisons de notre grille de paramètres.

Avant de passer à l'optimisation, une brève description des critères d'évaluations dans le domaine de modélisation des sujets. On distingue deux métriques importantes pour l'évaluation des sujets :

- **Perplexity** : de façon générale, la perplexité est une mesure de la bonne qualité d'un modèle probabiliste à prédire un échantillon. Dans notre cas, c'est la vraisemblance normalisé pour un test set d'évaluation
- **Conférence score** : la cohérence est une mesure de la similarité sémantique des mots clés dans un sujet. Ces mesures peuvent faire la distinction entre des sujets interprétables et des sujets issus d'un bruit d'inférence statistique.

Pour notre cas d'étude, on va s'intéresser au *coherence_score*.

Voici les valeurs qu'on va prendre en compte :

```
#range of topics
topics=list(range(5,16,2))

#range of alpha
alphas=list(np.arange(0.01,1,0.3))
alphas.append('symmetric')
alphas.append('asymmetric')

#range of beta
betas=list(np.arange(0.01,1,0.3))
betas.append('symmetric')
```

A l'issu de nos itérations, on va choisir nos hyper-paramètres qui maximisent le *coherence_score*. Voici les résultats tirés :

- N_topics : 15
- Alpha : 0.61
- Beta : 0.01
- Coherence_score : 0.56

On remarque que le modèle tend à nous donner le plus de sujets. Mais pour des raisons d'interprétabilité on va se limiter à notre maximum renseigné.
Ci-dessous le tableau qui regroupe les résultats avec les 10 mots clés pour chaque sujet :

Topic N	Topic Keywords	Topic context
1	'0.069*"app"+0.068*"android"+0.047*"map"+0.038*"server"+0.033*"firebase "+0.032*"query"+0.030*"git"+0.029*"aws"+.028*"error"+ 0.027*"application"	Web service
2	' 0.063*"return" + 0.056*"convert" + 0.047*"api" + 0.041*"build" + 0.040*"ios" + 0.037*"laravel" + 0.034*"user" + 0.034*"request" + 0.029*"connect" + 0.025*"state"	Application programming interfaces
3	'0.104*"type" + 0.075*"class" + 0.035*"template" + 0.034*"spring" + 0.031*"typescript" + 0.031*"date" + 0.026*"parameter" + 0.025*"non" + 0.022*"dependency" + 0.022*"generic"	Non définie
4	'0.097*"list" + 0.087*"string" + 0.064*"multiple" + 0.052*"two" + 0.051*"one" + 0.046*"different" + 0.030*"index" + 0.029*"within" + 0.027*"single" + 0.026*"replace"	Manipulation des listes
5	'0.055*"number"+ 0.044*"inside" + 0.044*"json" + 0.037*"get" + 0.034*"array" + 0.034*"possible" + 0.032*"sum" + 0.030*"clang" + 0.029*"operator" + 0.028*"instance"	Non définie
6	'0.084*"data" + 0.055*"rLang" + 0.055*"element" + 0.055*"variable" + 0.042*"group" + 0.035*"line" + 0.031*"specific" + 0.031*"item" + 0.030*"field" + 0.030*"input"	Data analysis with R
7	'0.053*"module" + 0.042*"package" + 0.038*"load" + 0.038*"issue" + 0.036*"html" + 0.036*"unable" + 0.032*"project" + 0.032*"change" + 0.031*"native" + 0.027*"display"	packages
8	'0.060*"find" + 0.057*"table" + 0.038*"model" + 0.036*"stream" + 0.035*"core" + 0.033*"argument" + 0.031*"use" + 0.030*"reference" + 0.028*"duplicate" + 0.028*"exception"	Non définie
9	'0.123*"python" + 0.116*"function" + 0.080*"object" + 0.046*"base" + 0.042*"key" + 0.042*"cplusplus" + 0.031*"order" + 0.028*"condition" + 0.024*"generate" + 0.023*"nest"	Orienté objet Python c++
10	'0.095*"value" + 0.086*"column" + 0.065*"java" + 0.058*"array" + 0.054*"row" + 0.047*"dataframe" + 0.042*"set" + 0.042*"panda" + 0.037*"method" + 0.024*"character"	Manipulation des series de données
11	'0.077*"flutter" + 0.070*"javascript" + 0.052*"loop" + 0.041*"custom" + 0.038*"js" + 0.036*"property" + 0.031*"define" + 0.031*"give" + 0.031*"command" + 0.029*"button"	Flutter app
12	'0.064*"time" + 0.058*"work" + 0.053*"angular" + 0.039*"result" + 0.037*"css" + 0.037*"page" + 0.036*"call" + 0.036*"filter" + 0.032*"script" + 0.031*"write"	Non définie
13	'0.055*"component" + 0.054*"text" + 0.045*"google" + 0.044*"import" + 0.040*"error" + 0.035*"run" + 0.029*"vue" + 0.028*"try" + 0.028*"pattern" + 0.027*"service"	Non défnie
14	'0.061*"image" + 0.052*"swiftui" + 0.037*"version" + 0.035*"window" + 0.030*"view" + 0.029*"sql" + 0.029*"color" + 0.028*"size" + 0.028*"node" + 0.021*"undefined"	Image et visualisation
15	'0.096*"file" + 0.089*"use" + 0.056*"fail" + 0.054*"docker" + 0.041*"std" + 0.035*"match" + 0.032*"VisualStudio" + 0.029*"container" + 0.028*"vector" + 0.026*"message"	Non définie

On retrouve des sujets similaires à l'analyse précédente.

On remarque que les utilisateurs de stackoverflow s'intéressent principalement à la manipulation des séries de données avec python et R, le développement des applications ainsi que la visualisation. Certes, on a beaucoup de sujets non définies, là où on n'a pas pu trouver le topic correspondant.

On a testé l'approche LDA avec nos différentes séries de données (data_nouns et data). Les résultats obtenus et le même pour les deux séries de données

On arrive donc rapidement à la conclusion que cette méthode n'est pas très satisfaisante pour pouvoir taguer nos questions. Cependant, on va analyses les résultats obtenus dans ce qui suit.

Analyse et exploration des résultats

Dans cette partie, on fait une analyse pertinente des résultats extraits de la LDA.

Sujet affecté à chaque document

On s'intéresse dans cette partie au sujet attribué pour chaque question avec notre LDA. Pour chaque document, notre modèle affecte une liste de sujets avec leur poids. On choisit le thème le plus significatif de chaque entrée

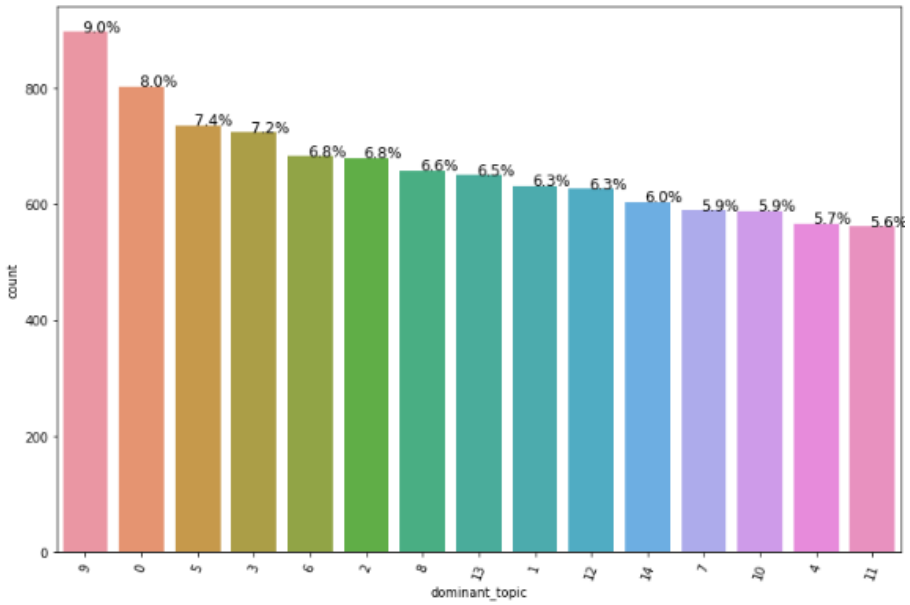
	dominant_topic	topic_contribution	keywords	text
0	9	0.122483	[value, column, array, java, row, dataframe, s...	[initialize, use, constructor]
1	12	0.122350	[component, text, google, import, character, r...	[panda, json, valueerror, protocol, know]
2	0	0.066667	[app, android, map, server, error, firebase, q...	[cumsum, restarts]
3	12	0.120914	[component, text, google, import, character, r...	[pod, give, error, relate, ruby, gem, libffi]
4	6	0.126249	[module, package, load, issue, html, unable, p...	[zwj, sequence, valid]

Dans ce jeu de données, on a stockée l'étiquette attribuée à chaque question. Voir la totalité des résultats dans le Notebook.

Distribution des sujets dans le corpus

On va regarder la distribution des topics dans notre corpus, en utilisant le jeu de données obtenu précédemment.

La figure ci-dessous montre le pourcentage de chaque sujet par ordre décroissant.



On remarque que le sujet « manipulation des séries de données » occupe le plus nos utilisateurs, suivi par « les services du web » et « l'analyse de données avec R »

Les sujets les moins demandés, n'ont pas été renseignés dans notre tableau par manque de cohérence entre les mots clés

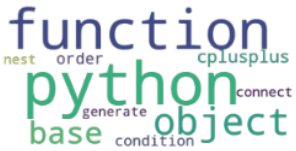
Wordclouds

Sujet 1

Sujet 2

Sujet 8

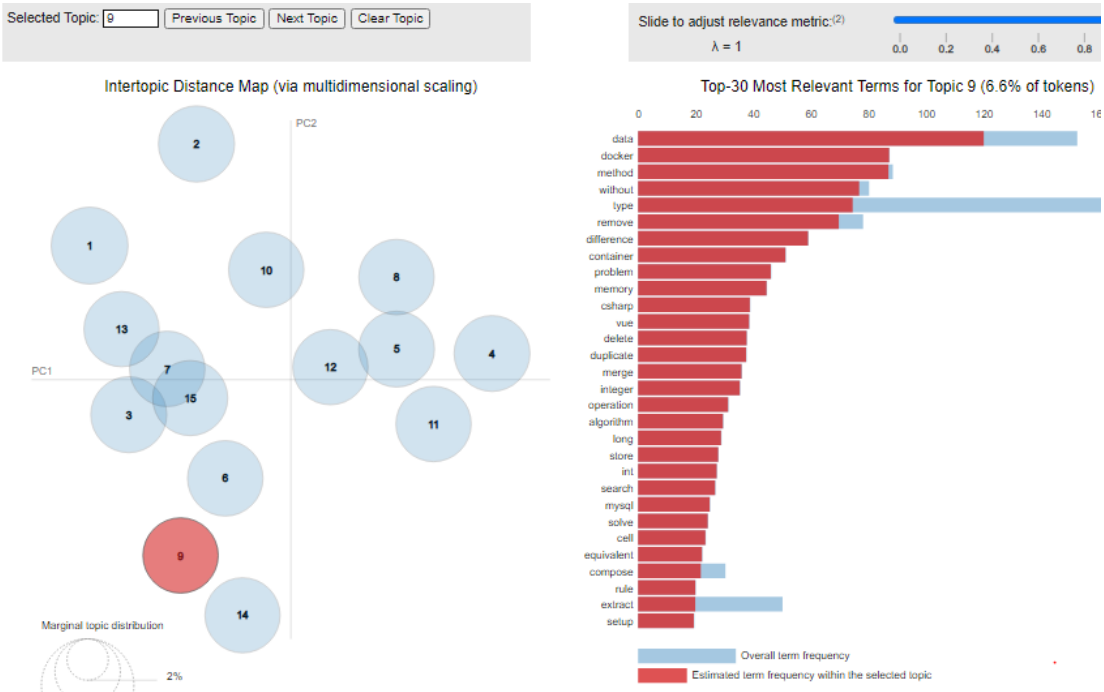
Sujet 9



Les word-clouds sont une parmi les meilleures méthodes de visualisation des sujets extraits. Là où on peut voir quels sont les mots clés qui définissent un certain sujet. Voir annexe pour le reste

pyLDAvis

Finalement, l'outil pyLDAvis est le plus couramment utilisé de nos jours dans l'analyse de LDA. Cet agréable moyen offre la possibilité de visualiser les informations contenues dans un sujet.



III. Catégorisation des questions : Méthodes Supervisées

On passe maintenant à mettre en œuvre une approche supervisée pour l'extraction des tags à partir des questions posées. Notre cas d'étude, représente un problème de classification en Multi-Label. Ce qui insinue qu'une seule entrée peut appartenir simultanément à deux classes ou plus. Cette approche attribue à chaque input une série d'étiquettes. Par exemple, une seule question peut avoir plusieurs tags à la fois (Python, Orienté objet, Pandas…)

Nous allons donc nous servir des tags déjà renseignés pour établir notre algorithme de classification en multi-label.

Pour la résolution de ce type de problématique, on a eu recours à plusieurs approches. D'abord, les algorithmes de transformations à savoir le **BinaryRelevance**, **ChainClassifier** et **LabelPowerset**. Je passe ensuite à explorer les méthodes adaptives comme le **MlKnnClassifier** issu de package **scikit-Multilearn**. Sans oublier l'**OneVsRestClassifier** de sklearn.multiclass

Comme critère d'évaluation, on va s'intéresser à l'**accuracy_score** ainsi que le **Hamming_loss** et le f1_micro pour les **Gridsearch** effectués

Préparation des données Multi-Labels

Filtrer les étiquettes

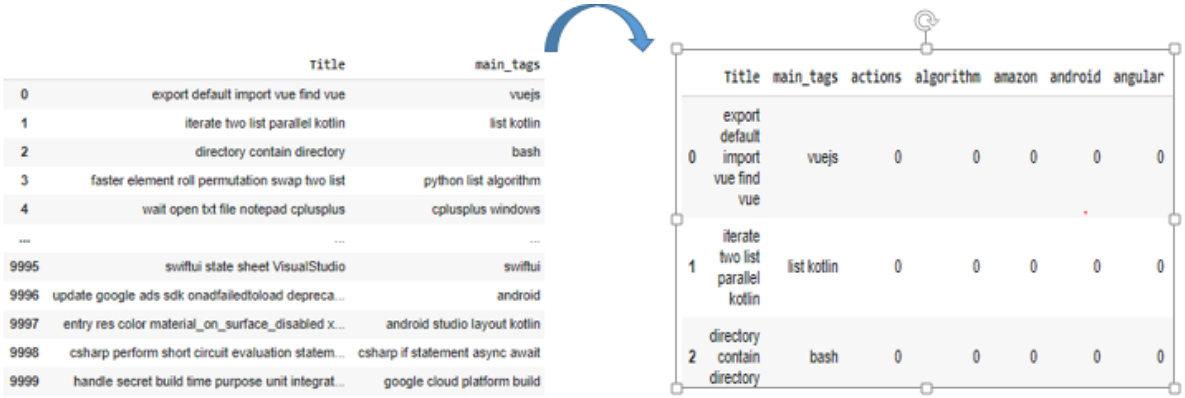
L'analyse des étiquettes, a montré qu'on a peu pré 200 tags présents dans nos étiquettes. Le tableau ci-dessous, présente une classification des étiquettes par ordre décroissant de leurs occurrences dans le jeu de données.

	tag	tag_occurences
0	python	1876
1	javascript	1023
2	java	782
3	rlang	694
4	cplusplus	625
5	android	530
6	reactjs	486
7	pandas	448
8	js	360
9	html	338

Dans le but d'assurer une prédiction pertinente, notre modèle sera entrainé sur un nombre d'étiquettes bien choisi. D'abord un nombre élevé de tags élevés va fausser les prédictions en multi-label et va causer des problèmes d'entrainement. Sans oublier le fait que ça consomme plus de temps de calcul. Avec la fonction top_tags, on fixe notre max_tags.

Générer le jeu de données en Multi-labels

Après avoir fait le nettoyage des étiquettes qu'on va garder. On passe à l'encodage de ces targets en effectuant une vectorisation sous forme de Bag of Words comme le montre la figure ci-dessous



	Title	main_tags
0	export default import vue find vue	vuejs
1	iterate two list parallel kotlin	list kotlin
2	directory contain directory	bash
3	faster element roll permutation swap two list	python list algorithm
4	wait open txt file notepad cplusplus	cplusplus windows
...	...	...
9995	swiftui state sheet VisualStudio	swiftui
9996	update google ads sdk onadfailedtoload depreca...	android
9997	entry res color material_on_surface_disabled x...	android studio layout kotlin
9998	csharp perform short circuit evaluation statem...	csharp if statement async await
9999	handle secret build time purpose unit integrat...	google cloud platform build

	Title	main_tags	actions	algorithm	amazon	android	angular
0	export default import vue find vue	vuejs	0	0	0	0	0
1	iterate two list parallel kotlin	list kotlin	0	0	0	0	0
2	directory contain directory	bash	0	0	0	0	0

Pour notre corpus, on a choisi d'entrainer notre modèle titres des questions.

Avec 14387 différents mots dans notre corpus, le temps de calculs pour l'entrainement de notre modèle seraient énormes. Donc on va procéder pour une réduction de dimensions à l'aide de l'outil de vectorisation TF-IDF. On a pris pour argument :

- **Max_features** : on va garder 3000 features
- **Max_df** : ignorer les tokens d'un document de fréquence en dessus de seuil fixé
- **Min_df** : ignorer les tokens d'un document de fréquence en dessous de seuil fixé

On procède finalement, à la séparation de notre jeu donné en entrainement et test en utilisant le modèle de *train_test_split* de sklearn

```
shape of X_train_tfidf: (6399, 3000)

shape of X_test_tfidf: (2743, 3000)

shape of y_train: (6399, 100)

shape of y_test: (2743, 100)
```

Classification des questions : Algorithmes testés

Principalement, on a trois méthodes pour résoudre les problèmes de classification multi-labels :

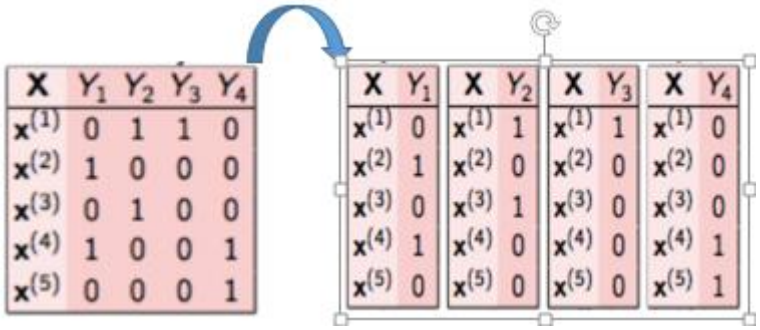
- Les algorithmes de transformations
- Les algorithmes adaptatifs
- Les méthodes ensemblistes

Méthode de transformations des problèmes

Cette méthode tend à transformer un problème de classification multi-labels à une classification binaire simple. On peut mener ces types de transformations avec plusieurs méthodes

Binary Relevance

Considérée comme la méthode la plus simple, l'algorithme BinaryRelevance crée des sous étiquettes dans le but de traiter chaque label comme une classification binaire



X	Y_1	Y_2	Y_3	Y_4
$x^{(1)}$	0	1	1	0
$x^{(2)}$	1	0	0	0
$x^{(3)}$	0	1	0	0
$x^{(4)}$	1	0	0	1
$x^{(5)}$	0	0	0	1

X	Y_1
$x^{(1)}$	0
$x^{(2)}$	1
$x^{(3)}$	0
$x^{(4)}$	1
$x^{(5)}$	0

X	Y_2
$x^{(1)}$	1
$x^{(2)}$	0
$x^{(3)}$	1
$x^{(4)}$	0
$x^{(5)}$	0

X	Y_3
$x^{(1)}$	1
$x^{(2)}$	0
$x^{(3)}$	0
$x^{(4)}$	0
$x^{(5)}$	0

X	Y_4
$x^{(1)}$	0
$x^{(2)}$	0
$x^{(3)}$	0
$x^{(4)}$	1
$x^{(5)}$	1

Dans cet exemple, on peut voir que X est nos données d'entrainement, et les vecteurs Y sont nos multi-labels. Avec cette méthode, le problème est divisé en 4 sous problématiques de classifications
Voir annexe pour l'implémentation

Classifier Chains

Dans cet algorithme, la transformation de problème en une classification binaire se fait rajoutant les étiquettes avec les données d'entrainement.

Pour mieux comprendre le principe, on passe à la figure ci-contre

X	y1
x1	0
x2	1
x3	0

Classifier 1

X	y1	y2
x1	0	1
x2	1	0
x3	0	1

Classifier 2

X	y1	y2	y3
x1	0	1	1
x2	1	0	0
x3	0	1	0

Classifier 3

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0

Classifier 4

Pour les Classifier_chain le problème est converti en 4 classifications binaires, comme le montre le processus ci-dessus. La couleur jaune montre les données d'entrainements utilisées à chaque itération.

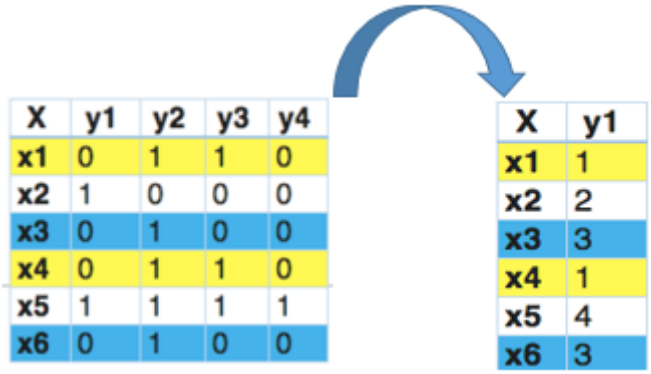
Similaire au binaryRelevance, mais la seule différence c'est qu'on enchaine les classifications binaires pour préserver la corrélation entre étiquettes.

LabelPowerset

Le Label Powerset est considéré un des méthodes les plus appréciées parmi les algorithmes de transformations. Avec cette implémentation, le modèle ne considère pas les étiquettes séparément,

mais plutôt comprend les combinaisons des sous-labels pour chaque entrée et ils les affectent une multi-classification

Voici un exemple



X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0
x4	0	1	1	0
x5	1	1	1	1
x6	0	1	0	0

X	y1
x1	1
x2	2
x3	3
x4	1
x5	4
x6	3

Dans ce cas, on remarque que les entrées x1 et x4 ont les mêmes étiquettes attribués, aussi les entrées x3 et x6. Dans ce cas, la transformation LabelPowerset convertit le problème en un problème de multi-classification simple.

Synthèse des résultats

		Gaussian NB	LogisticRegression	RandomForest	XGBoost
BinaryRelevence	Accuracy_score	0,042	0,084	0,155	
	Hamming_loss	0,026	0,01	0,008	
Classifier_chain	Accuracy_score	0,07	0,13	0,16	
	Hamming_loss	0,028	0,017	0,009	
LabelPowerset	Accuracy_score	0,15	0,14		
	Hamming_loss	0,023	0,012		

Les différentes constatations qu'on peut tirer de cette analyse :

- Pour des limitations de sources, on n'a pas eu de résultats pour le XGBoost Classifier
- Le RandomForest s'avère le meilleur algorithme de prédiction
- Le choix final entre les méthodes de Random Forest s'oriente vers le choix de Binary Relevance, vu que ça présente le plus faible Hamming_loss

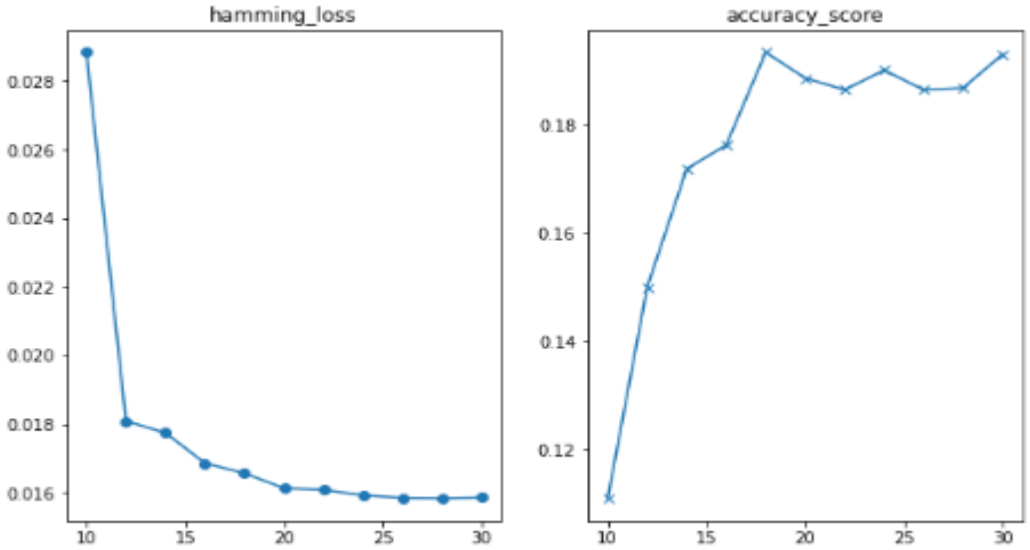
Méthodes des algorithmes adaptives

Comme son nom l'indique, cette méthode adapte l'algorithme pour effectuer directement une classification multi-étiquettes. Plutôt, que de transformer le problème en différents sous-ensemble de problèmes, l'idée c'est d'effectuer le processus de classification en multi-label.

Pour notre cas d'étude, on va avoir recours à la version multi-étiquettes de kNearestNeighbors proposée par le package scikit-multilearn

Tuning des hyper-paramètres

On va essayer d'optimiser le nombre des k voisins de notre algorithme



On a choisi une plage de valeur entre 10 et 30

Le but est de choisir la valeur optimale k, qui maximise l'**accuracy_score** et minimise le **hamming_loss**

Dans notre cas, la valeur optimale de k est 18

Voici les résultats de l'évaluation de notre algorithme de prédiction :

- Accuracy_score : 0.19
- Hamming_loss : 0.016

Méthode : OneVsRestClassifier

Cette méthode issue de la librairie scikit_learn-multiclass est très similaire au Binary Relevance. Partant du même principe, on fait une décomposition de notre classification en différents sous-ensemble de problématique.

Avec cette méthode, on va s'offrir la possibilité de faire un Gridsearch pour optimiser nos hyper-paramètres. On va essayer de combiner les différents algorithmes à savoir (RandomForest, XGBoost, kNearestNeighbors, MultinomialNB) avec l'OneVsRestClassifier pour obtenir le meilleur estimateur.

Multnomial NB	Logistic_regression	kNeighbors	RandomForest	XGBoost
"alpha": [0.01, 0.001]	alpha = [0.0001,0.01,0.1,1,10,100]	leafs = list(range(1,50))	'n_estimators'= [10,100,250,500]	n_estimators':[10,100,500]
	penalty=['l1','l2']	neighbors = list(range(1,30))	min_weight_fraction_leaf'=[0,0.25,0.5]	learning_rate' :[0.01, 0.1, 0.2]
		p_=[1,2]	max_depth': np.arange(1,6)	subsample':(0.2,0.6,0.8)
			min_samples_leaf': np.arange(0.05,0.5,0.05)	max_depth':np.arange(1,6)
			min_samples_split':np.arange(0.05,1.0,0.05)	min_child_weight':np.arange(1,6)

Ci-dessus la grille des paramètres qu'on a utilisés pour optimiser nos hyper-paramètres. On a essayé de tester plusieurs types d'estimateurs pour sélectionner le meilleur.

Voici les résultats obtenus pour l'OneVsRestClassifier

		Multinomial NB	LogisticRegression	RandomForest	XGBoost	kNeighbor
OneVsRestClassifier	Accuracy_score	0,145	0,158	0	0,168	0,046
	Hamming_loss	0,017	0,009	0,25	0,008	0,015

Comme on peut le voir, le meilleur résultat est obtenu avec le XGBoost.

Modèle final

Choix de modèle

On passe maintenant à faire le choix de notre méthode finale. D'après les trois méthodes de classification multi-label qu'on a étudié, on a un compromis à faire entre trois modèles :

Modèle	Paramètres
BinaryRelevance(RandomForest)	Accuracy: 0.155 Hamming_loss: 0.008
Multi kNearestNeighbour	Accuracy: 0.19 Hamming_loss: 0.016
OneVsRestClassifier(XGBoost)	Accuracy: 0.168 Hamming_loss: 0.008

Parmi ces trois modèles, notre choix final sera effectué sur le taux de prédictions des algorithmes.

Chaque modèle de prédiction présente un taux de prédictions manquantes sous forme de « Nans » on va essayer de voir combien de valeurs manquantes existent pour chaque modèle.

Sur le même échantillon d'évaluation (X_test) on a eu les résultats suivants :

- BinaryRelevance(RandomForest) : 862 valeurs manquantes
- MultikNN : 607 valeurs manquantes
- OneVsRestClassifier(XGBoost) : 874 valeurs manquantes

Et voilà! Notre modèle final est celui qui présente le moins des valeurs manquantes. Avec 607 valeurs manquantes des prédictions le kNearestNeighbors de la famille multi-learn sera notre choix définitif. Cependant, le taux des données non prédites est considéré comme très élevé par rapport à notre échantillon

Evaluation sur de nouveaux documents

Finalement, on va essayer de tester notre modèle choisi sur des questions trouvées dans Stackoverflow.

On va utiliser notre modèle pour comparer les tags fournis par stackoverflow et ceux proposé par notre approche

Questions	Tags de stackoverflow	Tags de modèle
'how would firebase be connected to a flask app'	#flask #app #api	#firebase #flutter
'how local transitivity and barrat transitivity are calculated for directed graph in igraph rlang	#graph #R	Pas de prédiction
'faster way to query pandas dataframe for a value based on values in other column'	#query #python #pandas #dataframe	#python #pandas #dataframe
'how to create json file based on pandas dataframe python'	#pandas #python #dataframe #json	#pandas #python #json
'returning a value from within a nested javascript function duplicate'	#Javascript #function	#array #Javascript
'jquery returns an error where it cannot find the property of an objects django'	#Javascript #html #django #json #jquery	#django

Conclusion

Cette étude porte sur l'attribution des tags à des questions posées par des utilisateurs sur la plateforme Stackoverflow. En utilisant le titre de corpus, le modèle est censé prédire une série de tags permettant d'orienter l'utilisateur à trouver la réponse à sa question.

Notre étude s'étale sur trois grandes parties :

- Nettoyage des données : Dans cette section on s'est intéressé à nettoyer nos Features pour pouvoir bien les exploiter avec nos algorithmes de prédictions. Les principales fonctions de nettoyage sont : suppression de la ponctuation, suppressions des balises html, filtrer les StopWords et faire la lemmatisation comme dernière étape

- Approches non supervisées : on a découvert la modélisation des sujets avec la librairie Gensim. On a effectué une série de transformation de notre jeu de données pour le mettre dans un format exploitable. Avec la méthode LDA, on a vu les différents sujets cachés de notre corpus. Ceci dit, l'approche non supervisée est peu précise pour prédire les tags. Nous nous orientons vers une approche supervisées pour voir si on prédit mieux nos étiquettes.

- Approches supervisées : on a testé plusieurs méthodes supervisées pour la classification de nos questions. En passant par les méthodes de transformation aux méthodes de classification en multi-étiquettes, plusieurs algorithmes de classifications ont été testés à savoir le RandomForest, XGBoost et le LogisticRegression.

Après avoir testé quelques algorithmes de classification supervisé à sorties multiples, On a décidé de garder la méthode multi-kNN de la famille scikit multi-learn. Dans le but d'obtenir les meilleures performances de cette méthode, nous avons essayé d'élaborer un Tuning des hyper-paramètres pour améliorer nos résultats.

Cependant, on pourra potentiellement améliorer les résultats en proposant comme:

- Choisir un dictionnaire plus pertinent contenant les abréviations de mots utilisées souvent par les utilisateurs. On a essayé de balayer toutes les propositions mais il reste de travail à effectuer.

- Tester l'augmentation de la taille de nos échantillons d'entrainement et de tests pour savoir si on peut avoir de meilleurs résultats avec un plus gros volume de données. Ici, on s'est limités à 10000 entrées pour des limitations de ressources

- Exploiter encore les méthodes ensemblistes et le boosting avec plus ressources

- On pourra trouver un moyen pour analyser les fausses prédictions en fonction de leur signification. Dans le cas où on prédit « java » au lieu de « javascript » on peut trouver un moyen pour faire en sorte que ce tag soit accepté comme une bonne prédiction

- On pourra bâtir nos modèles sur un corpus de Body et pas seulement les titres de questions. De la même façon on peut combiner les deux textes et voir qu'est ce ça donne en termes de précision

Annexe

Liste complète des wordcloud

Exemple d'implémentation d'une méthode de transformation avec le RandomForest comme un estimateur

```
binary_clf_rand=BinaryRelevance(RandomForestClassifier())
binary_clf_rand.fit(X_train_tfidf,y_train)
pred_forest = binary_clf_rand.predict(X_test_tfidf)
```

```
BinaryRelevance(classifier=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                  class_weight=None,
                                                  criterion='gini',
                                                  max_depth=None,
                                                  max_features='auto',
                                                  max_leaf_nodes=None,
                                                  max_samples=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100, n_jobs=None,
                                                  oob_score=False,
                                                  random_state=None, verbose=0,
                                                  warm_start=False),
                require_dense=[True, True])
```