Rapport Projet 6 :

« Catégorisez automatiquement des questions »

Bruno PINOS

Table des matières

Introduction

Nous allons dans cette étude proposer de catégoriser des questions issues du site stack Overflow. Stack Overflow est un site d'entraide permettant à ses utilisateurs d'échanger sur des problématiques rencontrées dans le développement informatique. Les questions portent sur les problèmes de programmation des utilisateurs et d'autres utilisateurs peuvent répondre à leurs questions. Un système de votes a également été mise en place de telle sorte que les utilisateurs peuvent se rendre compte des meilleures questions et des meilleurs réponses données sur le site.

Stack Overflow a été créé en 2008 et est rapidement devenu une référence en la matière. Le but de notre étude est de pouvoir proposer un taguage des questions pour des utilisateurs amateurs pour faire en sorte de leur simplifier l'utilisation du site. Le taguage est déjà présent sur le site de telle sorte que l'on pourra s'aider ou non de ce taguage.

Nous allons dans une première partie essayer de créer des variables qui vont nous permettre dans une seconde partie de pouvoir catégoriser le texte. Cette catégorisation se fera de façon supervisée ou non supervisée. Il s'agira ensuite de choisir quelle méthode semble la plus pertinente.

1. Préparation des données

1. Les Données

Pour retirer les données du site, il existe un outil d'export des données présent sur le site stackexchange.explorer. Il s'agit sur ce site d'écrire une requête sql pour pouvoir retirer les questions du site.

Nous avons choisi de ne retirer que les questions (et pas les réponses) avec un score supérieur ou égal à 10, qui ont eu au moins 1 réponse, qui ont été vu plus de 50000 fois et ajoutées au moins 3 fois aux favoris. Nous les avons requêtées par paquet d'approximativement 50 000 questions pour éviter des temps de requêtes trop long. Pour ces questions nous avons retiré le corps, le titre et les tags correspondants.

Voici un exemple de requête :

```sql
SELECT Id, Body, Title, Tags
FROM Posts
WHERE (Score >= 10) AND
(PostTypeId = 1) AND
(AcceptedAnswerId > 0) AND
(ViewCount > 50000) AND
(FavoriteCount > 3) AND
(Id < 10000000)

```

Nous avons extrait en tout, 101366 questions ce qui nous semble suffisant pour l'étude que nous allons mener.

2. Traitement des données

Nous avons ici à faire à des données textuelles. Il s'agit donc de pouvoir « nettoyer » le texte de toutes les informations qui ne nous intéressent pas et qui ne nous aideront pas à catégorisez les questions.

Nous allons computer les opérations suivantes à notre texte en utilisant le module NLTK :

- Enlever les balises HTML
- Enlever les parties de code
- Enlever les caractères qui ne sont pas des lettres
- Convertir tous les caractères en minuscule
- Enlever les accents
- Enlever les mots récurrents du langage (*stopwords* en anglais)
- Prendre la forme canonique de chaque mot (*Lemmatisation*)
- Tokenisation transformer les phrases en une liste de mots

Un exemple :

Texte original	Reading from Excel File using ClosedXML My Excel file is not in tabular data. I am trying to read from an excel file.
	I have sections within my excel file that are tabular.
	
	 I need to loop through rows 3 to 20 which are tabular and read the data.
	
	 Here is party of my code:
	
	<pre><code> string fileName = "C:\\Folder1\\Prev.xlsx";
	  var workbook = new XLWorkbook(fileName);
	  var ws1 = workbook.Worksheet(1);
	</code></pre>
	
	 How do I loop through rows 3 to 20 and read columns 3,4, 6, 7, 8?
	Also if a row is empty, how do I determine that so I can skip over it without reading that each column has a value for a given row.

Texte « nettoyé »	['read', 'excel', 'file', 'use', 'closedxml', 'excel', 'file', 'tabular', 'datum', 'try', 'read', 'excel', 'file', 'section', 'within', 'excel', 'file', 'tabular', 'need', 'loop', 'row', 'tabular', 'read', 'data', 'party', 'code', 'loop', 'row', 'read', 'column', 'row', 'empty', 'determine', 'skip', 'without', 'read', 'column', 'value', 'give', 'row']

3. Analyse du corpus

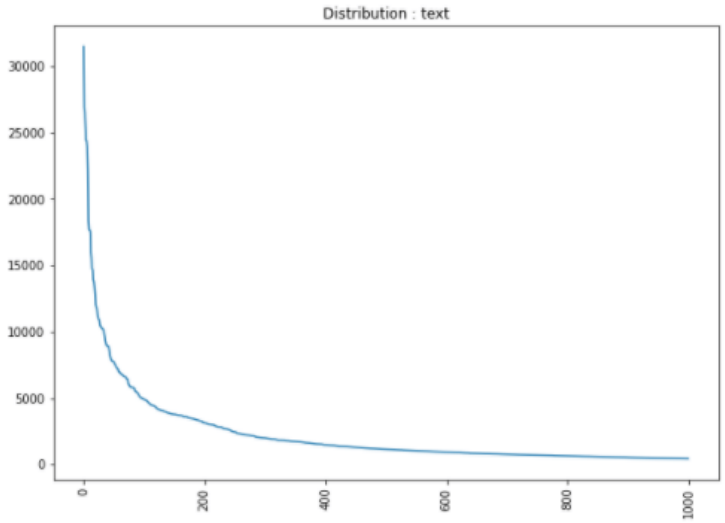
Après n'avoir gardé que les mots qui nous intéressaient, nous décidons de faire une petite analyse des mots de notre corpus.

Nous remarquons tout d'abord que notre corpus est constitué de **63093 mots différents**. On ne pourra garder tous ces mots pour notre catégorisation sous peine d'engendrer des calculs beaucoup trop longs.

Nous allons d'abord éliminer les mots qui n'apparaissent que très peu il faut supprimer suffisamment de mots pour que les calculs puissent être rapides. Pour cela on élimine tous les mots qui apparaissent dans moins de 50 documents. Il nous reste 3627 mots.

Nous allons maintenant nous intéresser aux mots qui reviennent le plus fréquemment. En effet ces mots très fréquemment utilisé sont souvent des mots neutres complètement inutiles pour prédire un tag.

J'ai commencé par observer la distribution des 1000 mots les plus fréquents :



Courbe de distribution des mots dans le corpus de questions

La méthode classique est d'enlever tout les mots à gauche du coude qui sont donc les mots les plus fréquents. Ici je dois enlever approximativement les 70 premiers mots. Je décide d'analyser les 100 mots les plus fréquents afin de prendre ma décision finale:

1	get	31488	26	add	10948	51	datum	7697	76	result	6009
2	like	26885	27	good	10887	52	think	7621	77	case	5968
3	try	26472	28	value	10517	53	line	7537	78	check	5852
4	want	25482	29	time	10402	54	type	7484	79	web	5828
5	code	24441	30	method	10381	55	java	7345	80	version	5824
6	work	24406	31	new	10251	56	edit	7282	81	html	5817
7	way	23965	32	could	10224	57	return	7224	82	build	5810
8	would	22237	33	change	10203	58	solution	7202	83	answer	5809
9	file	18380	34	thank	10168	59	list	6989	84	number	5769
10	find	17682	35	application	9908	60	app	6985	85	window	5744
11	one	17660	36	call	9596	61	simple	6913	86	not	5540
12	know	17561	37	function	9317	62	show	6893	87	test	5510
13	need	16064	38	give	9134	63	string	6815	88	first	5509
14	follow	15568	39	user	8957	64	name	6805	89	http	5435
15	error	14754	40	write	8951	65	able	6780	90	take	5412
16	make	14669	41	class	8916	66	read	6733	91	issue	5320
17	example	13907	42	possible	8914	67	thing	6651	92	table	5263
18	create	13809	43	help	8716	68	update	6641	93	text	5193
19	run	13379	44	server	8445	69	python	6636	94	database	5095
20	look	12828	45	object	8163	70	start	6628	95	go	5086
21	problem	12042	46	without	7990	71	different	6508	96	understand	5028
22	see	11855	47	net	7844	72	page	6484	97	access	5013
23	seem	11728	48	project	7810	73	command	6470	98	javascript	5007
24	set	11288	49	say	7808	74	idea	6417	99	base	4961
25	question	11102	50	two	7782	75	may	6083	100	variable	4923

Je décide d'enlever les 70 premiers mots des mots très important figure dans cette liste je décide donc de les garder malgré tout. ('java', 'list', 'net', 'app', 'c', 'server','class', 'application').

Nous avons donc gardées après ces quelques observations uniquement 3565 mots pour construire notre dictionnaire.

Pour achevé notre traitement de texte nous transformons nos données en Bag of Word comme ceci :

```
[(27, 1), (72, 1), (206, 2), (329, 4), (451, 1), (509, 2), (654, 4), (1010, 1), (1070, 1), (1123, 1), (1573, 1)]
```

Cette représentation permet à nos algorithmes d'interpréter nos données.

2. Catégorisation des questions

Nous allons dans cette partie tester des méthodes de taguages de nos questions Stack OverFlow. Nous allons tout d'abord tester des méthodes de taguages non supervisées c'est-à-dire sans entrainer d'algorithmes sur le taguage déjà existant. Ensuite nous évaluerons les méthodes de taguage supervisées

Il nous restera à choisir la méthode de taguage qui nous semblera la plus pertinente.

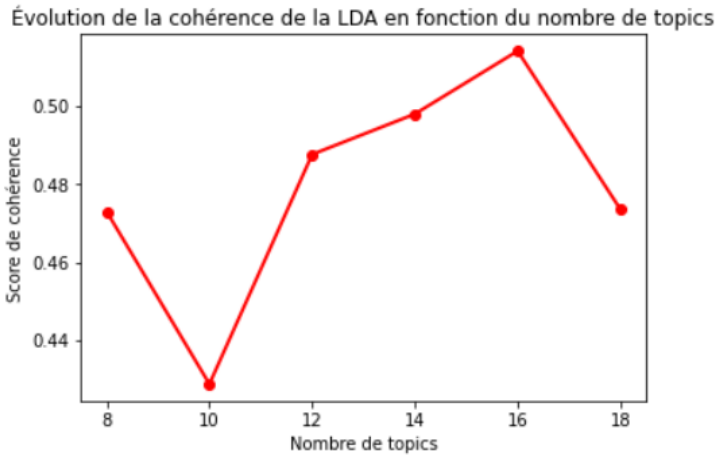
1. Apprentissage non supervisée

Pour l'apprentissage non supervisée nous allons réutiliser la représentation Bag of Word créé dans la partie 1.

a. Allocation Latente de Dirichlet

Nous allons utiliser l'algorithme appelé Allocation Latente de Dirichlet (LDA en anglais). Cet algorithme va nous permettre de trouver les n sujets les plus importants dans nos documents. Ces sujets sont décrits par la probabilité des mots de nos dictionnaires pour chacun des sujets.

Pour déterminer le nombre de sujets nous allons utiliser le score de cohérence. Nous allons calculer ce score pour un nombre de sujet allant de 8 à 20 et nous choisirons le nombre sujet avec le score de cohérence le plus élevé.



Score de cohérence en fonction du nombre de sujet

Comme nous cherchons le maximum, la courbe ci-dessus nous indique que le nombre optimal est de 16 sujets.

Nous allons d'abord observer les 16 sujets prédits par cet algorithme en donnant pour chacun les 10 mots les plus probables.

```
Topic # 1:
         page, php, html, javascript, browser, css, http, ie, content, com
Topic # 2:
         memory, size, large, performance, x, point, take, number, small, fast
Topic # 3:
         image, view, text, display, color, style, background, tab, pdf, template
Topic # 4:
         net, build, app, application, version, framework, tool, system, library, assembly
Topic # 5:
         table, database, sql, query, server, store, mysql, key, linq, field
Topic # 6:
         java, test, parameter, xml, eclipse, unit, pass, variable, argument, jar
Topic # 7:
         class, property, c, interface, implement, define, instance, reference, static, implementation
Topic # 8:
         git, folder, repository, delete, svn, copy, commit, quot, header, directory
Topic # 9:
         command, script, directory, path, output, window, expression, variable, text, match
Topic #10:
         python, print, module, library, json, package, parse, byte, encode, character
Topic #11:
         server, net, service, web, asp, request, client, application, access, http
Topic #12:
         form, jquery, event, click, button, control, select, wpf, window, input
Topic #13:
         application, thread, exception, visual, process, studio, log, message, debug, system
Topic #14:
         list, array, column, row, index, key, loop, element, item, sort
Topic #15:
         c, program, difference, language, ruby, linux, library, email, learn, window
Topic #16:
         number, convert, date, format, algorithm, character, c, integer, day, int
```

On peut aussi regarder la contribution des sujets à chaque document:

Index	Dominant Topic	Contribution
10000005	14	0.7369
10000083	7	0.4020
10000335	15	0.2511
10003270	14	0.3365
10003562	3	0.4234
...	...	...
1213074	13	0.3550
1213127	13	0.6187
1213137	3	0.7079
1213144	5	0.6893
1213217	4	0.4073

101366 rows × 2 columns

On peut aussi regarder pour chaque sujet le nombre de document associer :

Topic #	Keywords	Document count
7	class, property, c, interface, implement, define, instance, reference, static, implementation	10175
11	server, net, service, web, asp, request, client, application, access, http	9074
4	net, build, app, application, version, framework, tool, system, library, assembly	7640
5	table, database, sql, query, server, store, mysql, key, linq, field	7641
1	page, php, html, javascript, browser, css, http, ie, content, com	7533
14	list, array, column, row, index, key, loop, element, item, sort	7112
9	command, script, directory, path, output, window, expression, variable, text, match	6898
13	application, thread, exception, visual, process, studio, log, message, debug, system	6584
15	c, program, difference, language, ruby, linux, library, email, learn, window	5810
12	form, jquery, event, click, button, control, select, wpf, window, input	5802
16	number, convert, date, format, algorithm, character, c, integer, day, int	5675
3	image, view, text, display, color, style, background, tab, pdf, template	5579
6	java, test, parameter, xml, eclipse, unit, pass, variable, argument, jar	4416
2	memory, size, large, performance, x, point, take, number, small, fast	4197
8	git, folder, repository, delete, svn, copy, commit, quot, header, directory	3719
10	python, print, module, library, json, package, parse, byte, encode, character	3511

On voit que les documents sont divisés entre les sujets de façon relativement équilibré de 3500 à 10175 documents par sujet.

Pour prédire les tags on décide d'abord d'assigner à la question le sujet qui lui correspond le plus puis d'extraire les mots de la question qui sont des mots clé du sujet qu'on lui a assigné.

Exemple :

Texte original	Reading from **Excel** File using ClosedXML My Excel file is not in tabular data.
	I am trying to read from an **excel** file.
	I have sections within my **excel** file that are tabular.
	
	 I need to **loop** through rows 3 to 20 which are tabular and read the data.
	
	 Here is party of my code:
	
	<pre><code> string fileName = "C:\\Folder1\\Prev.xlsx";
	var workbook = new XLWorkbook(fileName);
	var ws1 = workbook.Worksheet(1);
	</code></pre>
	
	 How do I **loop** through rows 3 to 20 and read columns 3,4, 6, 7, 8?
	Also if a row is empty, how do I determine that so I can skip over it without reading that each **column** has a value for a given **row**.

Topic assigné	Topic #14: List, array, **column**, **row**, index, key, **loop**, element, item, sort…
Tags proposés	{'column', 'excel', 'loop', 'row'}

Les tags semblent à peu près correspondre à la question mais sans vouloir préciser à quoi correspondent tous les sujets, on se rend compte que nos sujets restent très vagues. On va donc avoir du mal à décrire nos questions avec la base de ses sujets quand bien même tous les sujets seraient correctement attribués aux questions. Nous n'allons donc pas retenir cette méthode.

2. Apprentissage supervisée

Nous allons donc nous servir des tags mais au préalable nous allons cleaner et étudier ces données qu'on a laissé de côté jusqu'à présent.

a. Préparation des tags

Voici des exemples de tags que nous renvoie le site stackexchange :

```
Id
10000005                            <php><arrays><sorting>
10000083    <javascript><events><event-handling><handler>
10000335                          <linux><gdb><libc><ldd>
10003270                        <c><arrays><gcc><struct>
10003562           <objective-c><ios><storyboard><segue>
                                ...
1213074          <wpf><winforms><interop><drag-and-drop>
1213127        <java><mysql><database-connection><sleep>
1213137                           <iphone><uitableview>
1213144                    <sql-server-ce><data-paging>
1213217             <perl><dependencies><module><legacy>
Name: Tags, Length: 101366, dtype: object
```

Voici les mêmes tags après cleaning :

```
Id
10000005                            [php, arrays, sorting]
10000083    [javascript, events, event-handling, handler]
10000335                           [linux, gdb, libc, ldd]
10003270                          [c, arrays, gcc, struct]
10003562           [objective-c, ios, storyboard, segue]
                                  ...
1213074          [wpf, winforms, interop, drag-and-drop]
1213127        [java, mysql, database-connection, sleep]
1213137                             [iphone, uitableview]
1213144                      [sql-server-ce, data-paging]
1213217             [perl, dependencies, module, legacy]
Name: Tags, Length: 101366, dtype: object
```

Après avoir cleaner ces tags on s'intéresse aux nombres totaux de tags différents : il y en a 14413.

On ne va pas s'intéresser à tous les tags d'abord parce que un trop grand nombre de tags poserait des problèmes à nos algorithmes lors de l'apprentissage : il nous faudrait un volume de données d'apprentissage beaucoup plus grand que nos 101366 questions et les temps de calculs seraient énormes mais également parce que notre but ici est de proposer un taguage pour des utilisateurs novices : qui ne sont pas intéressés pas un taguage de leurs questions trop précis. Ces utilisateurs sont potentiellement uniquement intéressés par un taguage assez général de leurs questions. Il s'agit de montrer la marche à suivre et non pas de faire le travail à leurs places.

Nous avons donc décidé de ne retenir que les 40 tags les plus communs

b. Représentation Tf-idf

Je récupère le Bag of Word :

```
Id
10000005    [(0, 6), (1, 1), (2, 1), (3, 1), (4, 1), (5, 2...
10000083    [(14, 1), (15, 2), (16, 1), (17, 1), (18, 3), ...
10003683    [(22, 2), (73, 1), (94, 1), (95, 1), (96, 1), ...
10006459    [(101, 2), (102, 1), (103, 2), (104, 1), (105,...
10006529    [(41, 1), (110, 2), (111, 4), (112, 2), (113, ...
                                    ...
1212838     [(8, 1), (50, 1), (95, 1), (101, 1), (103, 1),...
1212882     [(0, 1), (49, 1), (74, 1), (88, 1), (118, 1), ...
1212914     [(3, 1), (9, 1), (67, 1), (92, 1), (100, 1), (...
1212978     [(32, 1), (45, 1), (86, 1), (98, 1), (102, 7),...
1213127     [(19, 3), (46, 2), (61, 2), (100, 1), (167, 2)...
Name: Text, Length: 53721, dtype: object
```

Je le transforme en sparse matrix :

```
(1, 18)       3.0
(1, 19)       3.0
(1, 20)       1.0
(1, 21)       1.0
(1, 22)       1.0
(1, 23)       1.0
(1, 24)       1.0
:     :
(53720, 359)  2.0
(53720, 384)  1.0
(53720, 418)  1.0
```

Puis j'applique la transformation tf-idf :

```
(1, 24)       0.18654217900443262
(1, 23)       0.15154644676972484
(1, 22)       0.11537939789970482
(1, 21)       0.18673577902014998
(1, 20)       0.1432187039124216
(1, 19)       0.43743533447485367
(1, 18)       0.38473731881132844
:     :
(53720, 1474) 0.10544378818639528
(53720, 1459) 0.16051571558128963
(53720, 918)  0.19180985657631353
```

Le **TF-IDF** (de l'anglais term frequency-inverse document frequency) est une méthode de pondération souvent utilisée en recherche d'information et en particulier dans la fouille de textes. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Le poids augmente proportionnellement au nombre d'occurrences du mot dans le document. Il varie également en fonction de la fréquence du mot dans le corpus.

c. Les algorithmes testés

Après avoir fais un état de l'art des techniques de classification utilisées dans la fouille de texte, j'ai décidé de tester les 5 algorithmes qui me semblent les plus pertinents :

Naïve Bayésien :

La classification naïve bayésienne est un type de classification bayésienne probabiliste simple basée sur le théorème de Bayes avec une forte indépendance (dite naïve) des hypothèses. Elle met en œuvre un classifieur bayésien naïf, ou classifieur naïf de Bayes, appartenant à la famille des classifieurs linéaires.

SVM :

Les machines à vecteurs de support ou séparateurs à vaste marge (en anglais support vector machine, SVM) sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression. Les SVM sont une généralisation des classifieurs linéaires.

MLPclassifier :

Un perceptron multicouche (MLP) est une classe de réseau neuronal artificiel à réaction (ANN). Le terme MLP est utilisé de manière ambiguë, parfois vaguement à tout ANN anticipatif, parfois strictement pour désigner des réseaux composés de plusieurs couches de perceptrons (avec activation de seuil. Les perceptrons multicouches sont parfois communément appelés réseaux de neurones «vanille», en particulier lorsqu'ils ont une seule couche cachée.

Un MLP se compose d'au moins trois couches de nœuds: une couche d'entrée, une couche cachée et une couche de sortie . À l'exception des nœuds d'entrée, chaque nœud est un neurone qui utilise une fonction d'activation non linéaire. MLP utilise une technique d'apprentissage supervisé appelée rétro propagation pour la formation. Ses multiples couches et son activation non linéaire distinguent MLP d'un perceptron linéaire. Il peut distinguer les données qui ne sont pas linéairement séparables.

RandomForest :

Les forêts d'arbres décisionnels1 (ou forêts aléatoires de l'anglais random forest classifier) ont été premièrement proposées par Ho en 19952 et ont été formellement proposées en 2001 par Leo Breiman3 et Adèle Cutler4. Elles font partie des techniques d'apprentissage automatique. Cet algorithme combine les concepts de sous-espaces aléatoires et de bagging. L'algorithme des forêts d'arbres décisionnels effectue un apprentissage sur de multiples arbres de décision entraînés sur des sous-ensembles de données légèrement différents

Xgboost :

L'amplification de gradient est une technique d'apprentissage automatique pour les problèmes de régression et de classification, qui produit un modèle de prédiction sous la forme d'un ensemble de modèles de prédiction faibles, généralement des arbres de décision. Lorsqu'un arbre de décision est l'apprenant faible, l'algorithme qui en résulte est appelé arbres boostés par gradient, qui surpassent généralement la forêt aléatoire. Il construit le modèle par étapes comme le font les autres méthodes d'amplification, et il les généralise en permettant l'optimisation d'une fonction de perte différentiable arbitraire.

Il faut comprendre ici que notre sortie (les tags) est une sortie multiple : il peut y avoir plusieurs tags pour une seule question et il n'y a pas de priorité entre les tags. J'ai donc appliqué à toutes les méthodes citées précédemment l'une des fonctions suivante :

- **Multioutputclassifier**
- **OneVsRestClassifier**
- **OneVsOneClassifier**

Ces méthodes permettent d'utiliser nos algorithmes pour faire de la classification multi label.

d. Résultats des algorithmes

Pour évaluer les résultats nous allons utiliser plusieurs mesures qui permettent d'évaluer la qualité de la prédiction.

Precision et Rappel :

Dans les domaines de la reconnaissance de formes, de la recherche d'information et de la classification automatique, la précision (ou valeur prédictive positive) est la proportion des items pertinents parmi l'ensemble des items proposés ; le rappel (ou sensibilité) est la proportion des items pertinents proposés parmi l'ensemble des items pertinents. Ces deux notions correspondent ainsi à une conception et à une mesure de la pertinence.

Jaccard :

L'indice de Jaccard (ou coefficient de Jaccard, appelé "coefficient de communauté" dans la publication d'origine1) est le rapport entre le cardinal (la taille) de l'intersection des ensembles considérés et le cardinal de l'union des ensembles. Il permet d'évaluer la similarité entre les ensembles. Soit deux ensembles A et B, l'indice est :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

F1_score :

Dans l'analyse statistique de la classification binaire, le score F ou la mesure F est une mesure de la précision d'un test. Il est calculé à partir de la précision et du rappel du test, où la précision est le nombre de vrais résultats positifs divisé par le nombre de tous les résultats positifs, y compris ceux qui ne sont pas correctement identifiés, et le rappel est le nombre de vrais résultats positifs divisé par le nombre de résultats positifs. La précision est également connue sous le nom de valeur prédictive positive et le rappel est également appelé sensibilité dans la classification binaire diagnostique. Le score F1 est la moyenne harmonique de la précision et du rappel, 1,0, indiquant une précision et un rappel parfaits, et la valeur la plus basse possible est 0.

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}$$

Nous étudions des modèle multilabel il nous faut donc moyenner nos indice de mesure soit par échantillons soit par colonne. (_s : Par échantillon / _m : Par colonne)

Modèle	Recall	Precision	Jaccard_s	Jaccard_m	f1_m	f1_s	Time_ms
Naive_bayesian	0.468	0.843	0.478	0.437	0.59	0.493	35
LinearSVC	0.711	0.855	0.723	0.649	0.774	0.747	12
MLP_classifier	0.739	0.785	0.731	0.629	0.761	0.761	49
XGBoost	0.335	0.899	0.358	0.328	0.478	0.368	168
RandomForest	0.576	0.917	0.635	0.556	0.691	0.65	978

Score des différents algorithmes selon les différentes métrics

e. Amélioration et optimisation

Au vue des résultats j'ai choisi d'utiliser le model basé sur le LinearSVC, ces résultats sont à peu près équivalent au MLPClassifier et très largement supérieur à tous les autres modèles. En plus de cela c'est aussi le modèle le plus rapide.

Afin d'optimiser mon algorithme je procède à une phase de tunning, en jouant sur les différents paramètres :

- **Penalty** : Spécifie la norme utilisée dans la pénalisation. La pénalité «l2» est la norme utilisée dans SVC. Le «l1» conduit à des coef vecteurs clairsemés.

- **Loss :** Spécifie la fonction de perte. 'hinge' est la perte SVM standard (utilisée par exemple par la classe SVC) tandis que 'squared_hinge' est le carré de la perte de charnière.

- **Tol :** Tolérance pour les critères d'arrêt

- **C :** Paramètre de régularisation. La force de la régularisation est inversement proportionnelle à C. Doit être strictement positive.

J'ai ensuite conservé les paramètres qui donnent le meilleur score :
Penalty = L2 / Loss = Hinge / Tol = 1e-4 / C = 10

Conclusion

Nous avons dans cette étude essayé de prédire des tags à partir d'un corpus de questions stackoverflows avec leurs titres. Nous avons tout d'abord testé une méthode non supervisé qui s'est avérés être une méthode peu précise pour prédire des tags. Nous nous sommes alors tourner vers un taguage supervisé qui lui apporte des résultats bien meilleurs.

Après avoir procédé à une phase de benchmarking pour comparer les algorithmes de classification supervisés multilabels grâce à différente metric le score Jaccard et le score F1. Nous avons sélectionné le modèle le plus intéressant LinearSVC. Nous avons ensuite procédé à une phase d'optimisation ou nous avons joué de façon plus poussé sur les paramètres de l'algorithme mais aussi sur le nombre de tag possible ainsi que sur la base d'apprentissage (le traitement du corpus).

Nous obtenons un résultat satisfaisant: recall 0.711 et precision 0.855. Il reste cependant de nombreuses améliorations sur lesquelles on pourra potentiellement travailler pour améliorer les résultats comme:

- 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 (on avait ici près de 100 000 questions)
- on pourrait également s'intéresser aux tags prédits faux: on a fait ici que compter les tags qui sont faux mais on pourrait essayer d'évaluer à quel point la prédiction est fausse: le tag "C" est une erreur moins importante si le tag à prédire était "C++" que si on avait à prédire le tag "database".
- Il est également intéressant d'observer à l'apport de l'information du titre. En effet, il y a certainement plus d'informations à récupérer dans le titre que dans le corps de la question. Peut-être qu'en donnant plus d'importances aux mots du titre qu'aux mots de la question, on arriverait à un résultat amélioré.

Ces améliorations n'ont pas été mise en place mais nous avons tout de même essayé d'apporter une solution satisfaisante à ce problème de taguage des questions stack overflows.