

Catégorisez automatiquement des questions

Adrian Rodriguez - Ingénieur Machine Learning

Extraction des données	2
Architecture	2
Conditions	2
Requête finale	2
Informations diverses	2
Nettoyage des données textuelles	3
Traitement des tags	3
Exploration	3
Sélection des tags à conserver	3
Tags inclus dans les stop words et ponctuations	3
Traitement des questions	4
Exploration des champs texte	4
Nettoyage du texte	4
Exploration du corpus de mot	5
Modélisation	5
Classification non supervisée	5
Choix du modèle et vectorisation	5
Réduction de dimension	5
Visualisation des données de grandes dimensions	7
Suggestions de tags non supervisés	7
Classification supervisée	8
Vectorisation du corpus et des tags	8
Réduction de dimension	8
Évaluation des classifieurs	8
Tags les moins performants	10
Suggestion de tags supervisés	10
Comparaison entre les modèles	11
Mise en production	11
Annexes	13
Échantillon de la classification non supervisée	13
Échantillon de la classification supervisée	14
Environnement de développement	15

1. Extraction des données

1.1. Architecture

Les données sont disponibles via [StackExchange](https://stackexchange.com/). Plusieurs tables sont disponibles. La table `posts` contient les données relatives aux posts et contient un peu plus de 20 variables.

Dans le cadre de notre étude, nous nous intéresserons aux variables `Title`, `Body` et `Tags`. Je conserve également la variable `Id` des posts.

1.2. Conditions

Le site met à disposition un historique conséquent des posts des utilisateurs finaux. La qualité du modèle final dépendra de la qualité des questions.

Je les filtre en fonction des conditions suivantes :

- Posts ayant une ancienneté de 1 mois minimum : cela afin de laisser un minimum de temps à la communauté pour découvrir le post, et l'agréementer si besoin.
- Posts concernant uniquement des questions : les posts de réponse ne sont pas tagués et ne font pas partie de l'étude.
- Posts ayant un score supérieur ou égal à 20 points : les utilisateurs pouvant voter sur un post ont au minimum une réputation de 15 points.

1.3. Requête finale

```
DECLARE @max_date as DATETIME = DATEADD(MONTH, -1, GETDATE())

SELECT Id, Title, Body, Tags
FROM posts
WHERE CreationDate < @max_date AND PostTypeId = 1 AND Score > 19
ORDER BY CreationDate DESC
```

Le jeu de données contient **50k questions**.

1.4. Informations diverses

A titre d'information :

- Le site Stackoverflow reçoit un post, que ce soit en question ou en réponse aux questions, **toutes les 8.0 secondes** et une nouvelle question toutes les 18.0 secondes
- 38.0 % des questions posées ne trouvent **aucune réponse au bout d'1 mois**.

2.2. Traitement des questions

2.2.1. Exploration des champs texte

En posant moi-même une question sur Stackoverflow, je me rends facilement compte que je peux mettre des informations importantes, soit dans le titre, soit dans le body. Je ne pense pas forcément à les mettre dans les 2 blocs. De ce fait, je dois analyser le champs **Title** et le champs **Body** de manière conjointe.

2.2.2. Nettoyage du texte

J'utilise la technologie spaCy pour le nettoyage de texte. Ce package me permet l'utilisation d'un pipeline de traitement de donnée texte, et inclus par défaut les modules de prédiction tagger, parser et ner.

Composant 1 : J'ai créé un premier composant avec l'objectif de supprimer les blocs préformatés, qui contiennent en général du code, ainsi que des images. J'ai également retiré toutes les balises HTML, les retours ligne \n, les éventuels caractères accentués et j'ai transformé tous les caractères en minuscule s'il ne l'étaient pas.

Composant 2 : J'ai créé un second composant qui me permet de réduire les contractions de la langue anglaise, langue majoritaire sur le site de questions/réponses. J'ai fait cela dans le but de conserver un sens à la phrase.

Certains noms de technologies contiennent de la ponctuation; #, ., +. Nous pourrions conserver juste celles-ci. A ce stade du projet, nous les conservons toutes.

Composants originaux : Le composant "tagger", qui prédit le rôle d'un mot dans une phrase, et le composant "parser", qui prédit les dépendances syntaxiques d'un mot dans une phrase, sont utilisés. Le composant "ner", qui prédit les entités nommées, est désactivé.

Composant 3 : Ce composant me permet de traiter les prédictions faites par les composants originaux. Je conserve les mots ROOT dans une phrase. Ce sont des mots qui sont à la racine d'une question. Ensuite, je conserve les mots taggés nom (NOUN), adjectif (ADJ) et adverbe (ADV). Bien entendu, il faut que ces mots ne fassent pas partis des stops words. Un mot présent dans la liste des tags protégés est conservé quelque soit son taggage. Je rappelle que les stops word ont été retirés de la liste des tags.

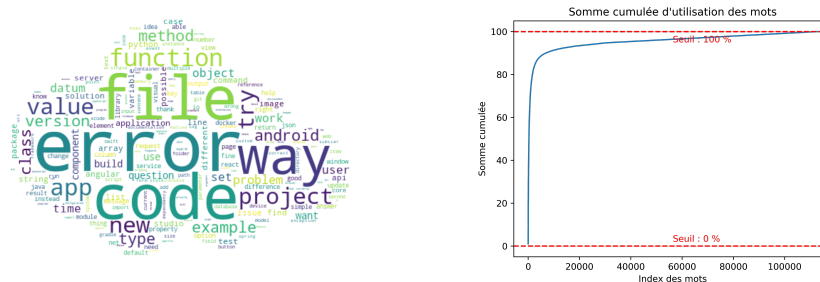
Stemmization : Je ne procède pas à la stemmization des mots dans ce projet. Ce procédé consiste à transformer un mot en sa racine. Par exemple, access peut être un tag désignant la technologie de Microsoft Access. Les racines des mots accessible, accessibility, entres autres mots, ont pour racine access. Si on procède à la stemmization dans ce projet, on se retrouvera avec des faux tags dans les phrases, et cela trompera le modèle final.

Au final, le pipeline de traitement des données texte est le suivant :

```
Pipeline: ['CleanBeforeTagger', 'CleanContractions', 'tagger', 'parser', 'CleanAfterParser']
```

2.2.3. Exploration du corpus de mot

Selon le nuage de mots, on peut constater que certains sont généralistes et sont très fréquents dans le corpus. Il serait intéressant d'y appliquer le principe de pareto comme pour les tags. Ce n'est cependant ici pas très applicable car je vais chercher à conserver les plus intéressants, en enlevant les plus courants et les moins courants. Pour enlever les mots les plus fréquents, j'augmente le seuil inférieur. Pour rejeter les mots les moins fréquents, je baisse le seuil supérieur, selon la problématique métier et les performances du modèles.



Ces options peuvent être manipulées avec les attributs `min_df` et `max_df` des vectorizers, mais je ne sais pas ce que ça enlève concrètement. Cette méthode me permet de le savoir. Pour la suite du projet, je règle les seuils à 0% et à 100%. De ce fait, je ne procède à aucun nettoyage supplémentaire sur les données textuelles. Après nettoyage complet, il reste **49131 questions dans le jeu de données**.

3. Modélisation

3.1. Classification non supervisée

3.1.1. Choix du modèle et vectorisation

2 modèles se sont présentés dans le cadre du projet; LDA, qui est probabiliste, et NMF qui est déterministe. Il est difficile d'évaluer des modèles non supervisés dans le cadre de projet texte et rend les comparaisons peu simple. **Mon choix s'est porté sur le modèle LDA** parce qu'il obtient les faveurs de la communauté en terme de résultats. De plus, il détient un framework de visualisation de données texte, pyLDAvis.

Les données textes sont transformées en données numériques avec `CountVectorizer()`. Les attributs sont laissés dans leur configuration par défaut. Cela consiste à calculer un vecteur de fréquence d'apparition d'un mot dans un corpus de texte. C'est aussi appelé "bag of words".

Le jeu de données devient **une matrice sparse de dimension 49131 * 55570**.

3.1.2. Réduction de dimension

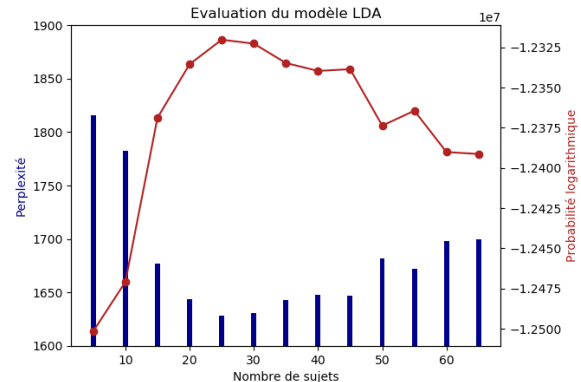
Il est nécessaire de réduire ce jeu de données contenant de nombreuses dimensions. Pour cela, je dois définir un nombre de dimension cible. Cette quantité sera au final un nombre de sujets abordés par le jeu de données.

Pour définir cette cible, j'entraîne des modèles selon différentes valeurs de `n_comp`, qui est le nombre de composants. J'évalue ensuite la perplexité et la probabilité logarithmique de chacun des modèles.

Dans le graphique ci-contre, on va chercher **la moindre perplexité** (en bleu) et **la probabilité logarithmique la plus élevée** (en rouge).

Le nombre de dimension idéal, donc de sujets, est de 25. On peut aussi voir une alternative avec 30 sujets.

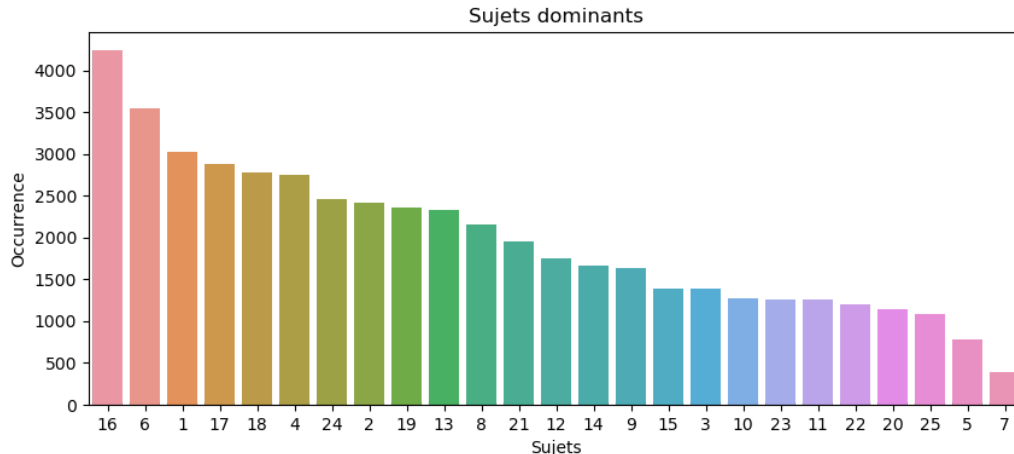
Le modèle de classification non supervisé, LDA, est donc entraîné avec 25 composants.



A partir de ce modèle, j'obtiens deux tableaux croisés :

- le premier me donnant **la probabilité des sujets par question** posée (document), qu'on appellera `doc_topic`.
- le second me donnant **les cinq mots les plus importants par sujet**, que l'on appellera `topic_keywords`. Ce chiffre cinq correspond à la limite de tags sélectionnable pour une question.

A titre d'information, voici les sujets dominants :

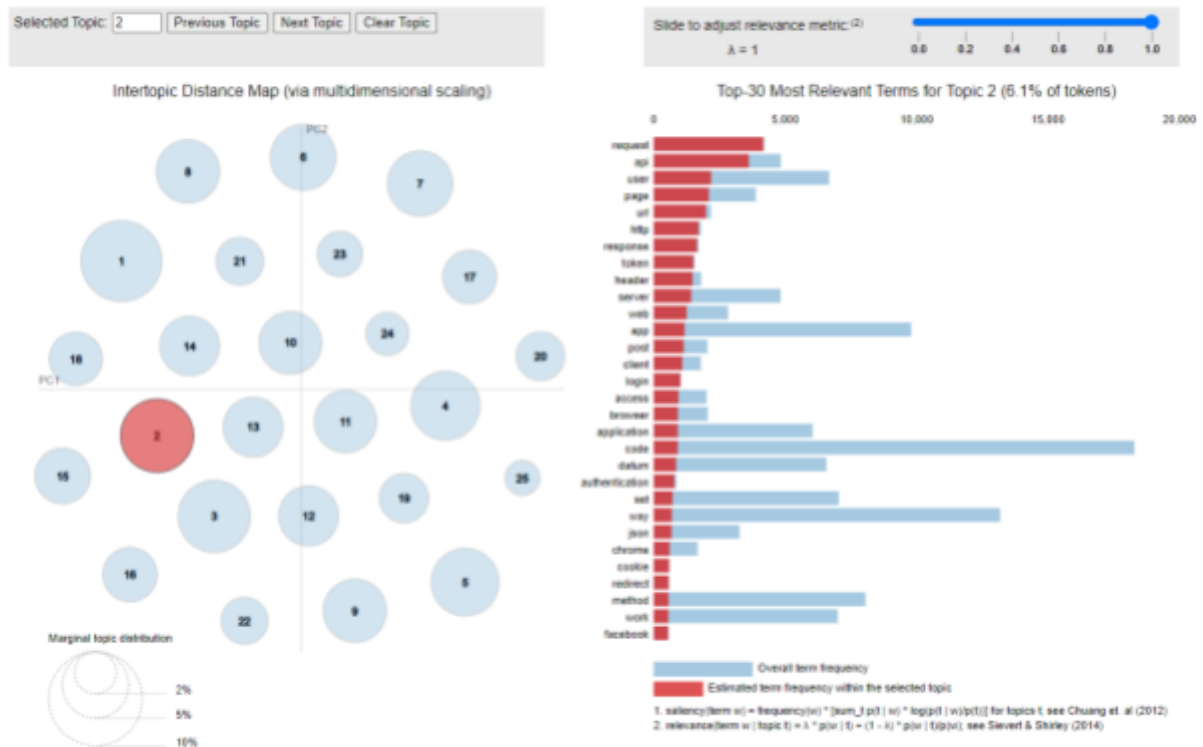


Bien que l'on sache que les sujets 16 et 7 sont respectivement le sujet le plus dominant et le sujet le moins dominant, cela ne nous avance pas plus. A ce stade, nous pourrions chercher à nommer les sujets pour pousser l'analyse. Une première méthode consisterait à définir manuellement les sujets. Sans avoir la connaissance intégrale du domaine, se lancer avec cette méthode serait complètement hasardeux. La seconde méthode consisterait à s'appuyer sur un corpus de texte externe, comme Wikipedia, pour nommer le sujet étudié. Cependant ceci ne fait pas l'objet de cette étude.

3.1.3. Visualisation des données de grandes dimensions

A l'aide du package pyLDAvis, il est possible de visualiser les données texte.

Visualisation des données textes - pyLDAvis



Je peux facilement visualiser les sujets, avec les mots qu'ils contiennent par importance. Dans l'exemple, ci-dessus, nous devinons que le sujet n°2 est dominé par les communications web avec ses principaux mots ; **requests, api, user, page, url**.

3.1.4. Suggestions de tags non supervisés

Je suggère mes tags non supervisés, qui seront limités à 5 unités pour être en phase avec le site de questions/réponses, à l'utilisateur final à partir des 2 matrices obtenues à partir de la réduction de dimension ; doc_topic et topic_keyword.

A partir de chaque question, je pondère le poids de chaque sujet dans un document et en fonction du résultat, je calcule le nombre de mots à choisir dans chaque document.

	Topic 1	Topic 2	Topic 3	Topic 4
Question 1	0.66 (3 mots)	0.33 (2 mots)	0 (0 mot)	0 (0 mot)
Question 2	0 (0 mot)	0.25 (1 mot)	0.6 (2 mots)	0.5 (2 mots)
Question 3	0 (0 mot)	0 (0 mot)	0.2 (5 mots)	0 (0 mot)

Tableau 1 : Exemple de sélection de tags

Les résultats sont visibles en annexe de ce document.

3.2. Classification supervisée

3.2.1. Vectorisation du corpus et des tags

Tout comme pour la classification non supervisée, il est nécessaire de transformer les données textuelles en données numériques.

Je procède à une vectorisation Tfidf avec `TfidfVectorizer()`. Cette technique fait un bag of word et évalue l'importance du mot dans le corpus. Cette importance augmente en fonction de la fréquence d'apparition du mot concerné. Il en sort une matrice sparse de même dimension que la vectorisation pour la classification non supervisée, mais les données à l'intérieur sont différentes. Ce projet nécessite une classification multi label puisque nous pouvons associer jusqu'à 5 tags par question. Je transforme les données tags en données numériques à l'aide de `MultiLabelBinarizer()`.

Je suis en possession des questions et des tags en versions numériques. Je splitte le jeu de données en jeu d'entraînement et jeu de test.

3.2.2. Réduction de dimension

A ce stade, plusieurs réductions de dimensions ont été testées dans le but d'économiser du temps de calcul ; AFC, TruncatedSVD, UMAP. Cependant, aucune ne m'a permis d'obtenir des résultats satisfaisants en phase d'évaluation. Une autre solution aurait été d'utiliser les sujets prédits par LDA, mais nous n'aurions plus été dans une classification purement supervisée, mais dite semi-supervisée. Supprimer les mots ne paraissant qu'une fois dans le corpus de texte réduit de moitié les dimensions, mais ce n'est pas viable car l'évaluation perd fortement en précision.

Aucune réduction de dimension n'a été appliquée pour la classification supervisée.

3.2.3. Évaluation des classifieurs

Les différents classifieurs ont été mesurés à l'aide du **score de Jaccard**. Un article en disponible [en ligne](#) explique très bien son fonctionnement. Il indique que pour 2 textes donnés, il va calculer un score de similarité :

$$\text{Score de Jaccard} = \frac{\text{Nombre de mots communs dans les textes}}{\text{Somme des mots uniques dans les textes}}$$

Si le score de Jaccard est égal à 0, alors il n'y a aucune similarité entre les textes. Si Jaccard est égal à 1, alors les textes sont identiques.

J'ai procédé à une première évaluation des classifieurs avec une méthode par grille afin de tester différents hyperparamètres. J'ai testé d'abord sur un échantillon de 10 000 questions des classifieurs linéaires et des classifieurs ensemblistes.

models	Test : Jaccard	mean_test_score	Prediction duration	mean_fit_time	estimator__C	estimator__alpha	estimator__n_estimators
LogisticRegression	0.363	0.34	00:00:00	148.1	10.0	nan	nan
LinearSVC	0.372	0.36	00:00:00	60.71	10.0	nan	nan
SGD	0.369	0.35	00:00:00	25.26	nan	1e-05	nan
RandomForest	0.151	0.11	00:01:51	2851.55	nan	nan	200.0
GradientBoosting	0.371	0.35	00:00:02	4269.48	nan	nan	200.0

Tableau 2 : Évaluation des classifieurs pour 10 000 questions.

Les méthodes ensemblistes sont clairement hors du coup. Un temps de prédiction moyen de 1mn51s est inenvisageable en production. RandomForest n'est pas adapté. GradientBoosting présente un temps d'entraînement moyen très élevé. Les coûts de maintenance peuvent être élevées.

Les méthodes linéaires présentent de très bons résultats et des temps de d'entraînements et de prédictions très satisfaisants.

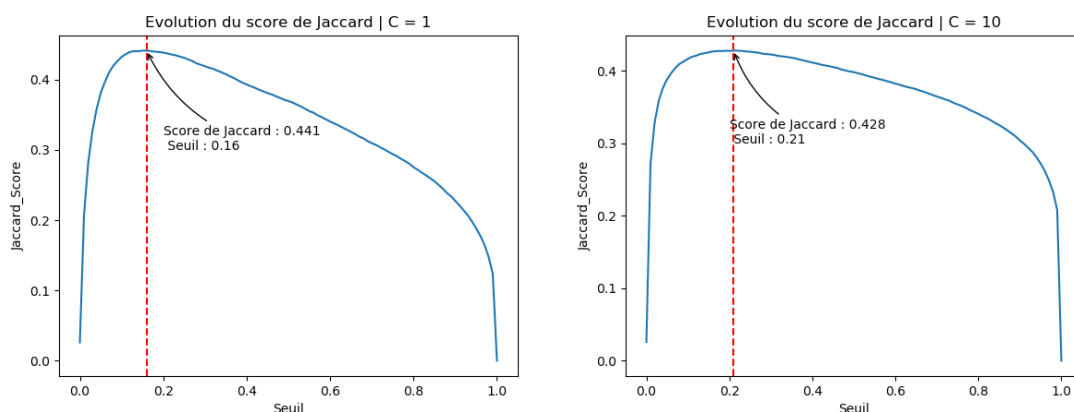
A ce stade, j'ai reproduit l'évaluation des méthodes linéaires sur l'intégralité des questions.

models	Test : Jaccard	mean_test_score	Prediction duration	mean_fit_time	estimator_C	estimator_alpha
LogisticRegression	0.398	0.39	00:00:01	1736.25	10.0	nan
LinearSVC	0.392	0.39	00:00:01	440.42	10.0	nan
SGD	0.41	0.4	00:00:01	158.53	nan	1e-05

Tableau 3 : Évaluation des classifieurs la totalité des questions.

Je présente des résultats légèrement meilleurs avec presque 50 000 questions, que ce soit sur les données de validation et sur les données de test. Le classifieur SGD obtient le meilleur score sur les données de test, mais je vais approfondir la régression logistique.

Cet algorithme détient un option qui permet de régler la probabilité de prédiction. Par défaut le curseur, est réglé à 50 %. Cela signifie que si le score de probabilité du mot courant est inférieur à 0.5, il n'est pas pris en compte dans la prédiction. Si il est supérieur, il est pris en compte. J'ai expérimenté différentes valeurs de ce curseur.

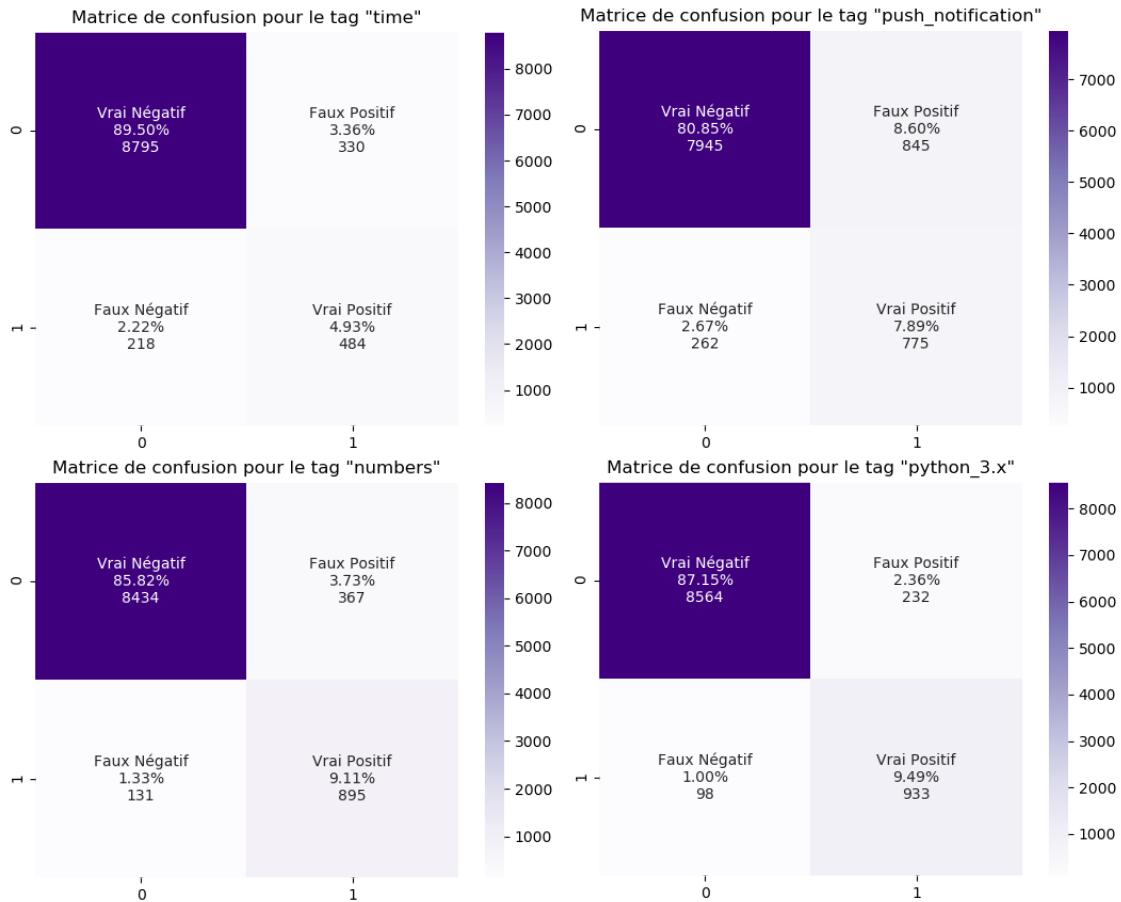


Avec $C = 1$, et un seuil à 16 % au lieu de 50 % sur la régression logistique, j'obtiens un score de Jaccard de 44 %, qui est meilleur que le classifieur SGD.

La régression logistique est retenue pour la modélisation supervisée.

3.2.4. Tags les moins performants

A titre d'information, j'ai trouvé intéressant de mettre en avant les tags les moins performants pour m'aider à trouver des axes d'améliorations.



Parmi les moins performants, nous trouvons les tags ci-dessus. Il s'agit de tags qui ont été attribués à tort ou bien qu'il aurait dû être attribués :

- push_notification a beaucoup de faux positif. C'est peut-être un tag qui est ambigu.
- time et numbers, sont des tags qui peuvent désigner des choses de la vie bien au-delà des technologies représentés.
- python_3.x est peut-être moins utilisé qu'il ne devrait l'être, et que les utilisateurs finaux préfère désigner le tag python.

3.2.5. Suggestion de tags supervisés

Un échantillon de classification supervisée est disponible en annexe.

Certaines questions ne sont pas prédites. Cela représente 7.74 % du jeu de test. On peut considérer que **1 question sur 10, reconnue utile par le communauté, n'aura pas de tags prédits** de manière supervisée. Dans le cadre d'une extension de ce projet, il serait intéressant de comparer ce chiffre avec l'ensemble des questions du site.

3.3. Comparaison entre les modèles

Au final, nous avons donc un modèle de classification multi label non supervisé avec une réduction de dimension avec LDA et un système de suggestion de tags selon la probabilité des sujets, et nous avons un modèle de classification multi label supervisée, avec une régression logistique.

La classification non supervisée reste intéressante mais très généralisée. Cependant elle peut donner des idées de tags auxquels l'utilisateur n'avait pas forcément pensé. Cela peut enrichir le contenu en ajoutant plus de spécificité à ce modèle. On peut faire cela en enlevant les mots les plus courant du corpus de texte, comme envisagé lors du nettoyage des données textuelles.

La classification supervisée donne des tags en phase avec le sujet de l'utilisateur final, malgré quelques imperfections. Presque 1 question sur 10 reconnue utile ne serait pas prédite, dans le cadre des tags sélectionnés.

Il s'avèrent finalement que les 2 types de classifications multi label soient complémentaires.

Axes d'améliorations :

- Prévoir une préparation de données spécifiques pour chaque type de classification, supervisée ou non supervisée,
- Gagner en spécificité sur la classification non supervisée en introduisant des seuils de rejet de mot de corpus. Les `CountVectorizer()` et `TfidfVectorizer()` peuvent gérer cet aspect là, mais je ne sais pas ce qui est rejeté,
- Tester la réduction de dimension LDA avec 30 sujets,
- Comparer les résultats obtenus avec des modèles en provenance du deep learning. J'ai souvent croisé BERT dans mes recherches, mais il existe aussi ELMo, ULMFiT et OpenAITransformer. Source [Medium.com](https://medium.com).

4. Mise en production

Le projet est en production sous forme d'une API Flask déployée sur les [serveurs de Heroku](https://heroku.com).

Lors du déploiement de mes modèles finaux sur les serveurs de Heroku, j'ai été confronté aux limites de mon compte gratuit (Septembre 2020), qui limite l'utilisation de la mémoire à 2 * 512 Mo :

```
heroku[web.1]: Process running mem=1209M(236.1%)
```

```
heroku[web.1]: Error R15 (Memory quota vastly exceeded)
```

Heroku tue mon processus avec `SIGKILL`. Pour dépasser cette limite, la souscription à l'offre "standardx2" aurait été nécessaire. Je n'ai cependant pas fait ce choix-là.

Afin de respecter les contraintes de Heroku, j'ai choisi de ré-entraîner mes modèles avec 10 000 questions, dont le classificateur supervisé donne un score de Jaccard de 40 % (avec un seuil de probabilité à 0.11) et dont la réduction de dimension avec LDA reste à 25 sujets. **Les performances sont donc assez similaires**, et l'échantillonnage du GridSearchCV au chapitre 3.2.3 le confirme également.

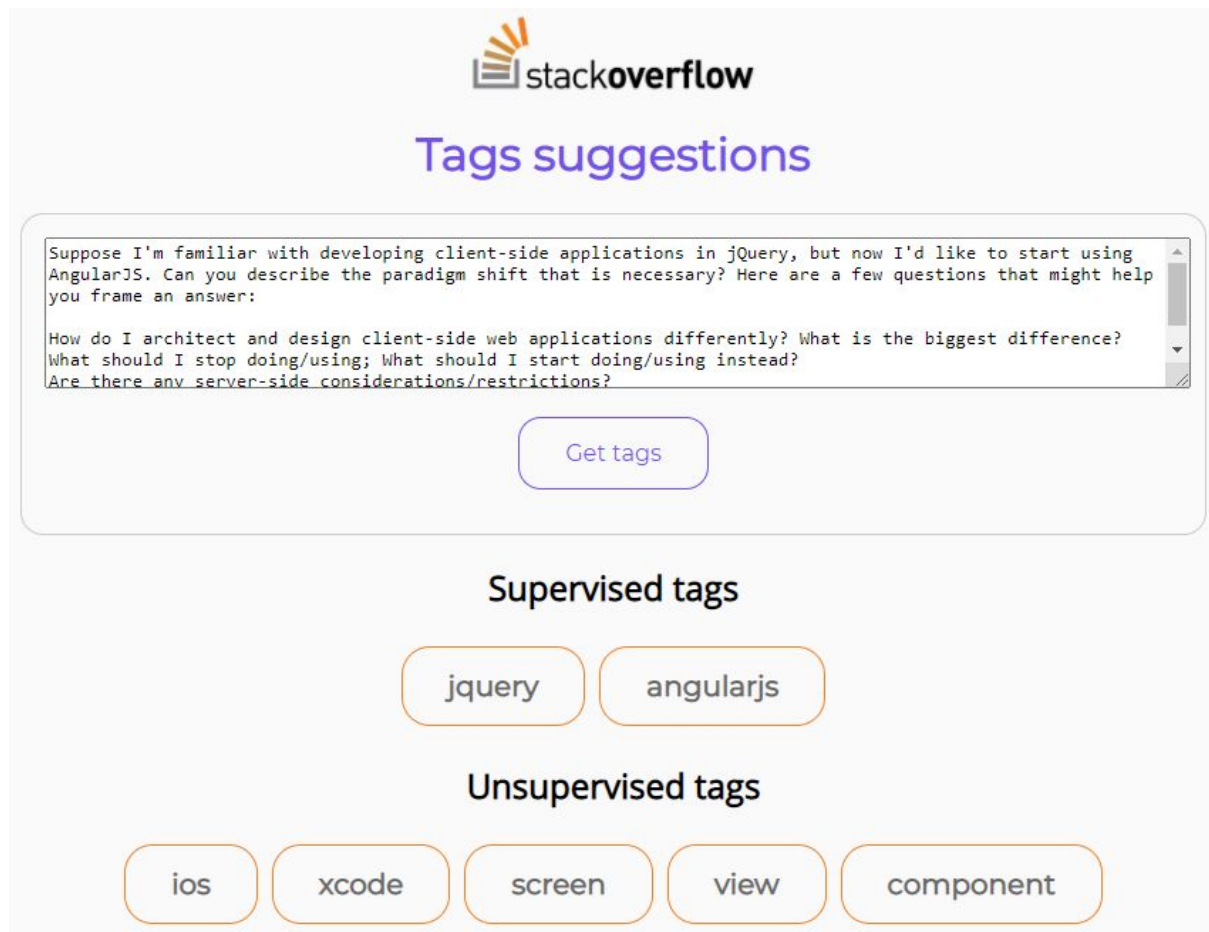
Et puisque les 2 modèles sont finalement assez complémentaires, j'ai décidé de les déployer ensemble en production, à savoir :

- **Classification supervisée entraîné à 10k questions** avec une régression logistique, présentant un score de Jaccard de 40 %,
- **Classification non supervisée entraîné à 10k questions** avec LDA, et suggestions de tags par probabilité.

Est aussi en production tout le traitement préalable aux suggestions de tags, à savoir **les nettoyage de texte avec le pipeline spaCy ainsi que les vectorizers** (countvectorizer, tfidfvectorizer, multilabelbinarizer), **également entraîné à 10k questions**.

Il est possible d'utiliser le modèle à 50 000 questions en l'installant en local. Pour plus d'informations, visiter le projet [GitHub](#).

Note : Le serveur se met en veille au bout d'un temps déterminé. Se rendre sur l'URL le réveille automatiquement et met quelques secondes à se charger lors du démarrage.



stackoverflow

Tags suggestions

Suppose I'm familiar with developing client-side applications in jQuery, but now I'd like to start using AngularJS. Can you describe the paradigm shift that is necessary? Here are a few questions that might help you frame an answer:

How do I architect and design client-side web applications differently? What is the biggest difference? What should I stop doing/using; What should I start doing/using instead? Are there any server-side considerations/restrictions?

Get tags

Supervised tags

jquery angularjs

Unsupervised tags

ios xcode screen view component

5. Annexes

5.1. Échantillon de la classification non supervisée

	Cleaned_Title_Body	Tags	unsupervised_tag
0	change mutable reference field reflect origina...	rust,reference	key,type,object,array,c lass
1	possible new data type javascript possible new...	javascript	datum,type,object,array ,value
2	elastic search index delete frequently elastic...	elasticsearch	document,xcode,ios,erro r,column
3	usage c++17 constexpr try c++17 constexpr cond...	c++,if_statement,templates,c ++17,constexpr	string,function,code,di fference,error
4	java recent require download recent jdk start ...	java,visual_studio_code	xcode,ios,android,studi o,project
5	int128_t fast long long x86 gcc test code test...	c++,performance,x86_64	time,memory,date,differ ence,class
6	discord.py random error TypeError new _ unexpe...	python,python_3.x	swift,error,service,and roid,studio
7	bigint inconsistency powershell c # bigint dat...	c#,powershell,.net_core	error,datum,table,numbe r,column
8	implicit conversion work java java integer lit...	java,casting,type_conversion ,implicit_conversion	string,function,type,ob ject,array
9	inspect element find result yellow usual test ...	google_chrome,google_chrome_ devtools	component,text,list,col or,column

5.2. Échantillon de la classification supervisée

	original_text_cleaned	original_tags	supervised_pred
0	messageall prop unit property component well r...	jasmine,reactjs,unit_test ing	javascript,reactjs,unit_ testing
1	imodelfilter jsonignore webapi2 recommend swas...	asp.net_web_api,c#,swagge r	
2	fishy affected uiswitch kcfrunloopcommonmode u...	xcode11	
3	ngforof company ts property get final componen...	angular,angular2_template	angular
4	github repository android studio project want	android,android_studio,gi t	android,android_studio,a ndroid_studio_3.0,git,.. .
5	when script wrong tell function suggest necess...	node.js	javascript,python
6	kube kubernetes cluster kubect1 kubernete azur...	azure,kubernetes	azure,google_kubernetes_ engine,kubect1,kubernete s
7	tsconfig doc typescript child directory setup ...	typescript	typescript
8	getintentsender cancelall notificationmanagerc...	android,android_notificat ions	android
9	play_arrow sharedinstance inforation sender fa...	crashlytics,firebase,fire base_console,swift	crashlytics,firebase,ios ,swift,xcode

5.3. Environnement de développement

Windows-10-10.0.18362-SP0

Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]

Numpy 1.19.1

Pandas 1.0.3

Seaborn 0.10.0

Matplotlib 3.1.3

requests 2.24.0

BeautifulSoup 4.8.2

re 2.2.1

spacy 2.3.2
