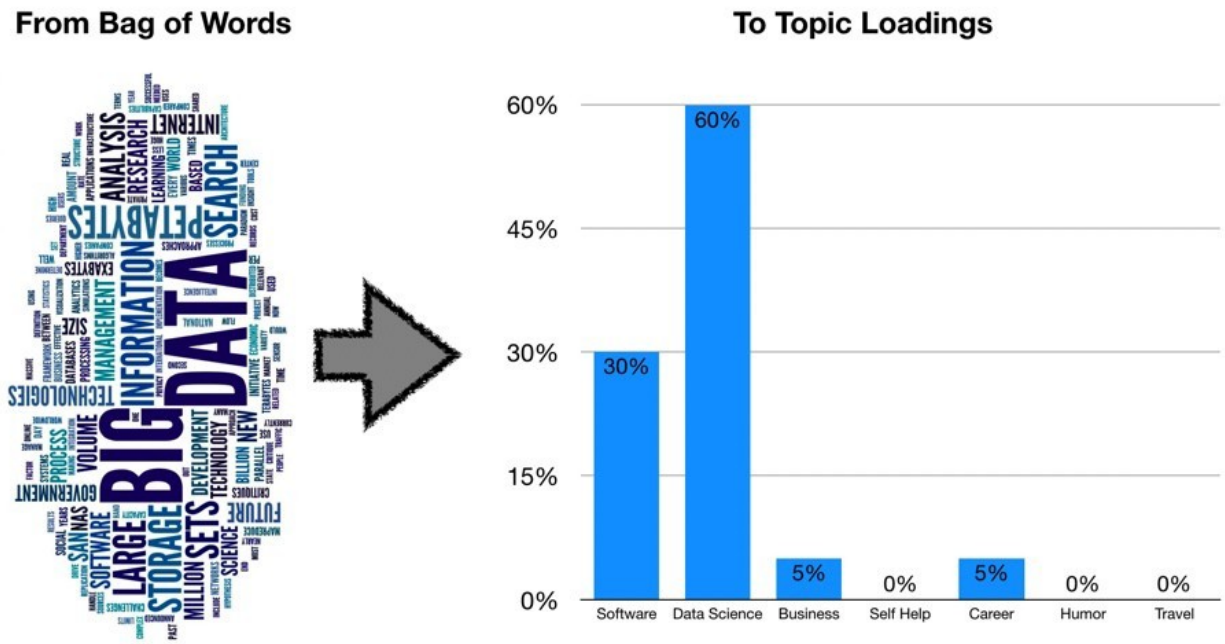
Catégorisation automatique des questions posées sur Stackoverflow



Marco Berta

Table des Matières

1. Introduction

Dans cette étude j'ai appliqué le traitement automatique du langage naturel, 'natural langage processing' (NLP) en anglais, au texte html de 'Stackoverflow', le plus grand site de questions-réponses sur Internet consacré à la programmation et au développement. Le but du projet était de développer une application pour classer automatiquement chaque question posée sur le site et lui assigner des mot clés (tags). Pour cette tâche j'ai traité un jeux de données exporté par le site[1]

Sur ce site j'ai téléchargé un tableau de questions et les mots clé respectifs avec une requête SQL. J'ai sélectionné les 50000 questions les plus vues, considérées comme favorites par les utilisateurs (Favoritecount) et pertinentes (Score), avec plus d'une réponse. Le code est le suivant:

```
SELECT
        TOP 50000 ViewCount,
   CreationDate,
   Title,
   Body,
   Tags,
   Score,
   CommentCount,
   AnswerCount,
   FavoriteCount

FROM Posts
WHERE
   FavoriteCount > 10
   AND AnswerCount > 1
   AND Score > 100
ORDER BY Score DESC
```

J'ai obtenu un fichier csv de 50MB, prêt pour être importé avec Python comme Pandas dataframe (Tableau 1).

```
[4]: print(df_raw.shape)
     df_raw.head()
```

```
(37175, 9)
```

[4]:

	ViewCount	CreationDate	Title	Body	Tags	Score	CommentCount	AnswerCount	FavoriteCount
0	1483128	2012-06-27 13:51:36	Why is processing a sorted array faster than p...	<p>Here is a piece of C++ code that shows some...	<java><c++><performance><optimization><branch-...	24320	22	26	10983
1	8547399	2009-05-29 18:09:14	How do I undo the most recent local commits in...	<p>I accidentally committed the wrong files to...	<git><version-control><git-commit><undo><pre-c...	20895	13	83	6776
2	8115583	2010-01-05 01:12:15	How do I delete a Git branch locally and remot...	<p>I want to delete a branch both locally and ...	<git><version-control><git-branch><git-push><g...	16826	6	40	5357
3	2782271	2008-11-15 09:51:09	What is the difference between 'git pull' and ...	<p>What are the differences between <code>git ...	<git><version-control><git-pull><git-fetch>	11833	9	35	2333
4	2783219	2009-01-25 15:25:19	What is the correct JSON content type?	<p>I've been messing around with <a href="http...	<json><http-headers><content-type>	10204	0	36	1446

Tableau 1. Données importées dans Pandas à partir du fichier obtenu par la requête SQL.

Ce document est structuré comme suit. Dans la section 2 on trouve le nettoyage et le pré-traitement des documents pour obtenir un corpus prêt à l'analyse et une liste des mots clé . La section 3 présente une analyse exploratoire des données. La section 4 traite la représentation vectorielle du

1 'Query Stack Overflow - Stack Exchange Data Explorer' <https://data.stackexchange.com/stackoverflow/query/new> [accessed 5 June 2020].

texte. La modélisation non-supervisée est présentée dans la section 5 et la modélisation supervisée dans la section 6. La section 7 présente le choix du modèle pour le traitement automatique et les conclusions.

2. Pré-traitement des données

L'objectif de cette étape est de standardiser le texte pour rendre son traitement facile par les algorithmes d'apprentissage automatique. Dans la colonne des mots clé, Tags, (Tableau 2) il suffit d'enlever les parenthèses.

Title	Body	Tags
Why is processing a sorted array faster than p...	Here is a piece of C++ code that shows some...	<java><c++><performance><optimization><branch-...
How do I undo the most recent local commits in...	I accidentally committed the wrong files to...	<git><version-control><git-commit><undo><pre-c...
How do I delete a Git branch locally and remot...	I want to delete a branch both locally and ...	<git><version-control><git-branch><git-push><g...
What is the difference between 'git pull' and ...	What are the differences between <code>git ...	<git><version-control><git-pull><git-fetch>
What is the correct JSON content type?	I've been messing around with <a href="http...	<json><http-headers><content-type>

Tableau 2. Les colonnes 'Body' et 'Tags' dans la forme originelle des fichiers html.

Par contre, le pré-traitement du corpus du texte (colonne 'Body') est plus complexe. Le texte de chaque question est dans la forme du langage html, par exemple dans la première ligne on lit:

'' Here is a piece of C++ code that shows some very peculiar behavior. For some strange reason, sorting the data miraculously makes the code almost six times faster: \n\n<pre class="lang-cpp prettyprint-override"><code>#include <algorithm>\n#include <ctime>\n#include <iostream>\n\nint main()\n{\n // Generate data\n const unsigned arraySize = 32768;\n ''

Enlever les caractères html à la main serait très long, et pour ça j'ai utilisé le paquet ''Beautiful soup'' . Le texte traité par ''Beautiful soup'' devient:

'' Here is a piece of C++ code that shows some very peculiar behavior. For some strange reason, sorting the data miraculously makes the code almost six times faster:\n#include <algorithm>\ n#include <ctime>\n#include <iostream>\n\nint main()\n{\n // Generate data\n const unsigned arraySize = 32768;\n ''

Mieux, mais pas encore prêt au traitement automatique. Après la concaténation avec les mots du titre, plusieurs étapes de nettoyage étaient nécessaires.

- traitement des contractions plus couramment utilisées en anglais, ex. changer "it's" en "it is".
- enlever les chiffres
- enlever les caractères spéciaux comme @, #,%, <
- enlever la ponctuation

- enlever les mots pas utiles (stopwords)

finalement j'ai séparé chaque phrase en mots ou ''tokens'' avec l'outil de segmentation ''word-tokenizer'' du paquet python ''nltk'' (natural language toolkit). Pour chacun d'eux on peut appliquer la racinisation (stemming) que donne une forme tronquée du mot, commune à toutes les variantes morphologiques
Ex : *cheval, chevaux, chevalier, chevalerie, chevaucher*⇒''*cheva*''

mais pour cette étude il est plus intéressant d'obtenir la forme canonique (le *lemme*) à partir du mot

Pour un verbe: sa forme à l'infinitif

Pour un nom, adjectif, article, ...: sa forme au masculin singulier

avec une fonction de lemmatisation. Pour ça le paquet ''textblob'' donne une meilleure performance que ''nltk'' .

Voici un exemple de question (post) avant:

```
move the most recent commit(s) to a new branch with git i would like to move the
last several commits i have committed to master to a new branch and take master
back to before those commits were made. unfortunately, my git-fu is not strong
enough yet, any help? i.e. how can i go from this master a - b - c - d - e to
this? newbranch c - d - e / master a – b
```

et après les dernières étapes de nettoyage (tokens, lemmatisation...)

```
['recent', 'commit', 'new', 'branch', 'git', 'like', 'commits', 'commit',
'master', 'new', 'branch', 'master', 'commits', 'unfortunately', 'git-fu',
'strong', 'help', 'ie', 'master', 'c', 'd', 'e', 'newbranch', 'c', 'd', 'e',
'master']
```

3. Analyse exploratoire

Les mots plus fréquents dans le corpus nettoyé sont représentés dans le ''nouage de mot'' en Figure 1.

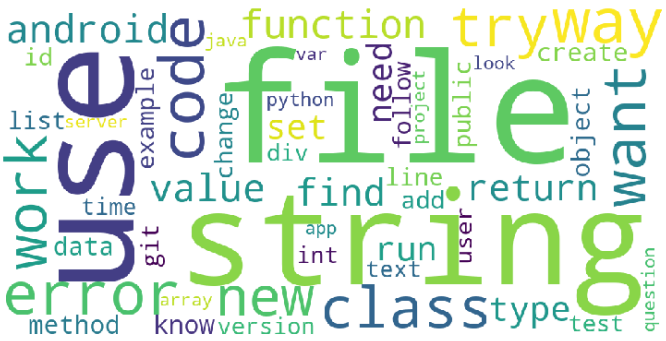


Figure 1. Les mots les plus utilisés dans les corpus traités.

Ce sont des mots que l'on trouve suivant dans les post de Stackoverflow. Avec la fonction python ''value_counts'' appliquée à la colonne ''tags'' on retrouve les mots clés les plus fréquents (tableau 3).

tag	count	frequency (%)
javascript	3919	3.47796
python	3576	3.173561
java	3434	3.047541
c#	2742	2.433418
android	2450	2.17428

Tableau 3. Les 5 mots-clés (‘‘tags’’) plus fréquentes.

Avec un groupement des données textuelles par un mots clé spécifique on peut trouver les mots du corpus associés (exemple en Figure 2) et chercher des régularités distributionnelles qui permettraient une prédiction du mot clé.

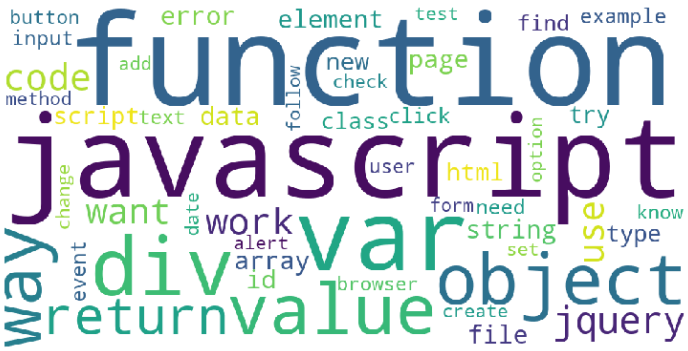


Figure 2. Mots du corpus plus fréquemment associés au mot clé ‘‘javascript’’

Pour effectuer automatiquement cette tache, il faut d’abord numériser les mots du corpus et les mots clés plus importants et représenter chaque mot comme un vecteur (vectorisation), puis appliquer aux données numériques des algorithmes d’apprentissage automatique.

4. Vectorisation

La vectorisation consiste en transformer chaque variable catégorique, ou chaque mot dans cette étude en un vecteur que pourra être traité par des algorithmes de catégorisation.

4.1 Vectorisation des mots-clés

Du moment que plusieurs mots-clés (variable cible) peuvent être associés à une question (variable à classifier), pour transformer en vecteur la colonne des tags (v. Tableau 2) j’ai utilisé la fonction‘‘MultiLabelBinarizer’’du paquet sklearn. Cela donne une matrice avec plus de 8000 colonnes, avec chaque colonne correspondant à un mot-clé. Une matrice creuse, avec beaucoup des colonnes que correspondent à des tags rarement utilisés. Pour améliorer la qualité des données cibles (tags) au lieux de l’utiliser la matrice obtenue à partir de la totalité des tags j’ai fait une réduction dimensionnelle avec une sélection des variables. J’ai appliqué la fonction ‘‘filter’’ et choisi que les 110 mot clés plus fréquents avec une forte réduction des nombre des variables/colonnes.

4.2 Vectorisation du corpus

Pour le transformer le corpus en une matrice des termes j'ai testé deux méthodes de vectorisation:

- Représentation vectorielle du sacs de mots

Le corpus est considéré comme un ensemble nonstructuré (sac-de-mots), sans information sur la séquentialité des mots dans le texte. À chaque mot présent dans le document est associée une valeur reflétant son importance comme descripteur du document. Pour cette méthode l'importance est calculée à partir de la fréquence du mots dans le corpus, term frequency (TF) en anglais. Finalement,le texte est donc décrit comme un vecteur d'un espace ayant pour dimensions les mots du vocabulaire, beaucoup plus des colonnes que pour la matrice initiale des tags. Dans ce cas la on peut réduire les nombre des colonnes en sélectionnant celles avec les valeurs plus importants. Il suffit d'indiquer le numéro dans les options de la fonction ‘‘CountVectorizer’’ du paquet sklearn, et j'ai réduit la dimension de la matrice obtenue à 220 colonnes.

- Représentation vectorielle de la matrice des termes TF-IDF

Avec cette méthode on relativise l'importance d'un terme dans un document (TF) par son importance dans le corpus (InverseDocumentFrequency /IDF). Autrement dit, plus un terme apparaît dans un grand nombre de documents, moins il est pertinent. Au lieux de ‘‘CountVectorizer’’, pour obtenire la matrice TF-IDF j'ai utilisé la fonction ‘‘TfidfVectorizer’’. Après avoir effectué la réduction dimensionnelle à 220 colonnes j'ai obtenu le mêmes variables (features) correspondant à des mots du corpus. La différence est que chaque variable est pondérée autrement. De conséquence les 10 mots les plus importants du corpus ne sont pas les mêmes si on utilise la note TF-IDF au lieux de la fréquence (Tableau 3).

Mots avec la meilleure note TF-IDF

words	idf_weight	count	TF_IDF score
file	2.736351	17370	47530.417314
android	3.915541	10096	39531.297872
string	3.048507	12716	38764.813379
class	3.145947	11375	35785.151931
error	3.020527	11575	34962.603081
use	2.386402	14324	34182.821721
like	2.264955	14620	33113.642445
function	3.100097	10272	31844.193251
value	3.12546	9943	31076.444477
code	2.616212	11800	30871.307214

Mots plus frequéntes

words	idf_weight	count	TF_IDF score
file	2.736351	17370	47530.417314
like	2.264955	14620	33113.642445
use	2.386402	14324	34182.821721
string	3.048507	12716	38764.813379
way	2.421332	11923	28869.540056
code	2.616212	11800	30871.307214
error	3.020527	11575	34962.603081
want	2.448669	11504	28169.487636
class	3.145947	11375	35785.151931
work	2.576914	11094	28588.279975

Tableau 3. Fréquence (count) vs. TF-IDF des mots dans le corpus.

On trouve dans la littérature du traitement automatique du langage que la représentation pondérée TF-IDF est meilleure que la fréquence non pondérée pour choisir les tokens associés aux mots clés[2]. Cette conclusion à été vérifié pour cette étude. Après une évaluation des résultats obtenu avec le même modèle d'analyse textuelle applique respectivement à la matrice obtenue avec CountVectorizer et TfidfVectorizer, cette dernière à été celle traitée avec les différents modèles.

2 Raffael Vogler, ‘The Tf-Idf-Statistic For Keyword Extraction’, *Joy of Data*, 2014, World <https://www.joyofdata.de/blog/tf-idf-statistic-keyword-extraction/> [accessed 5 June 2020].

5. Modèles non-supervisés

Le modèle non-supervisé classique pour l'extraction des thémes ou ‘‘sujets latents’’ d'un texte, à partir d'une matrice terme – document, est l'allocation latente de Dirichlet, Latent Dirichlet Allocation (LDA) en anglais. Pour cette modélisation, il faut définir le nombre de sujets et le nombre de mots attribués à chaque sujet. Si on choisi 5 sujets pour notre corpus factorisé avec la méthode TF-IDF on obtient :

```
Topic #0: value data var table array key select like id date
Topic #1: android div text view page image class html button input
Topic #2: use difference line like find code python app version application
Topic #3: file error git run command try project work change script
Topic #4: string class return function method public new list int object
```

On voit que est raisonnable assigner au sujet 1 le mot clé ‘android’ et ‘git’ au sujet 3.

Une autre méthode pour l'extraction des thèmes des documents est la factorisation en matrices non négatives (NMF). Les mots associés aux 5 sujets du corpus avec l'NMF sont:

```
Topic #0: android id px text true view background button app item
Topic #1: string public new return int code function like use method
Topic #2: option value time select key php input type id name
Topic #3: file error git directory version run find http line build
Topic #4: div class id px text button type width input form
```

Pour choisir le nombre optimale des thèmes, j'ai partagé le documents du corpus entre une partie d'entraînement du modèle (data_train) et une de validation (data_test). La perplexité estimée sur un corpus de validation est inversement proportionnelle à la précision du modèle. La variation de la perplexité pour l'LDA en fonction des nombres des thèmes est représentée en Figure 3.

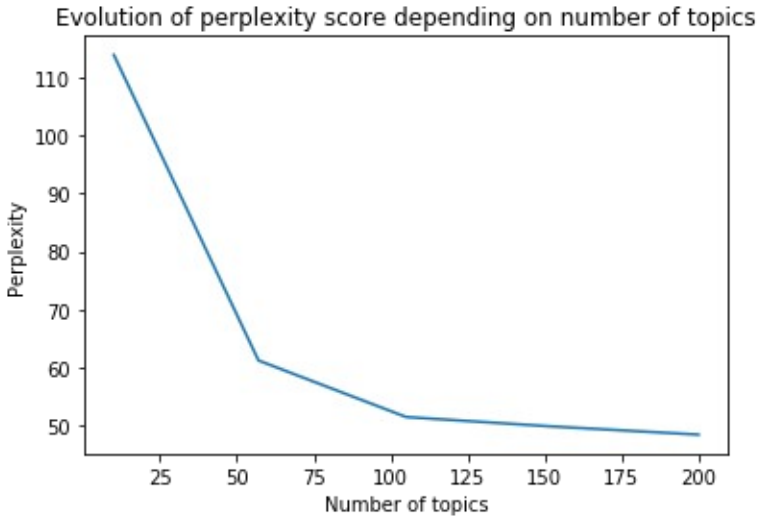


Figure 3. Perplexité de l'LDA en fonction du nombre des thèmes du corpus.

Le valeur optimale est celui du ‘‘coude’’, le minimum des sujets latents avec une faible perplexité. Dans ce cas la il correspond à 55. En suite pour chaque thème j'ai codé une fonction (decorator) que

assigne des mots-clés basés sur la note TF-IDF des mots contenus chaque document (question/post sur Stackoverflow). En suite , j'ai appliqué la même fonction en utilisant, à la place de l'LDA, l'extraction des sujets par factorisation en matrices non négatives (NMF).

Finalement j'ai testé le paquet d'extraction automatique des mots clés ''YAKE!''. Avec YAKE! Il suffit d'appliquer la fonction ''extract_keywords'' directement aux documents du corpus sans les vectoriser auparavant. Les principes de fonctionnement de YAKE! Sont décrits en détail ici[3].

Pour comparer les méthodes non-supervisées utilisés dans cette étude, j'ai vectorisé la colonne de prédictions de chaque modèle j'ai comparé la similarité du vecteur à celui des tags. La similarité textuelle a été mesurée avec la similitude cosinus des vecteurs de terme[4]. Les résultats montrent que la performance de YAKE! est la meilleure (Figure 4).

```
y_tfidf_YAKE = tfidfVectorizer.transform(df_test['YAKE tags'])
print('Dummy score:',cosine_similarity(y_tfidf_test.reshape(1,-1), y_tfidf_dummy.reshape(1,-
1)))
print('YAKE score:',cosine_similarity(y_tfidf_test.reshape(1,-1), y_tfidf_YAKE.reshape(1,-1)))
print('LDA score:',cosine_similarity(y_tfidf_test.reshape(1,-1), y_tfidf_LDA.reshape(1,-1)))
print('NMF score:',cosine_similarity(y_tfidf_test.reshape(1,-1), y_tfidf_NMF.reshape(1,-1)))
```

```
Dummy score: [[0.05342378]]
YAKE score: [[0.29256526]]
LDA score: [[0.18009483]]
NMF score: [[0.18110464]]
```

Figure 4. Résultats obtenus par les différents modèles non-supervisés.

6. Modèles supervisés

L'objectif de la classification supervisée est principalement de définir des règles permettant de classer des objets dans des classes à partir de variables qualitatives ou quantitatives caractérisant ces objets à partir des données dont le classement est connu. Pour cette étude les variables qualitatives sont les tokens du corpus, représentés par les vecteurs de la matrice TF-IDF, et le classement est fait selon les mots-clés (matrice des tags, obtenue avec la fonction ''MultiLabelVectorizer''). Pour valider chaque modèle, on dispose au départ d'un échantillon dit d'apprentissage, 80% du corpus (X_train) avec les tags correspondants à chaque question (y_train). Cet échantillon est utilisé pour l'apprentissage des règles de classement que sont en suite testées sur le restant 20% du corpus (échantillon de validation X_test). pour mesurer la exactitude (''accuracy'' en anglais) [5] de chaque modèle, ses prédictions (y_pred) sont comparées aux mots clés de l'échantillon de validation (y_test). Les résultats obtenus sont représentés par l'histogramme en Figure 5. Le modèle supervisé plus performant est clairement celui des ''forets aléatoires'',Random Forest en anglais, avec une optimisation des hyper-paramètres.

3 Ricardo Campos and others, 'YAKE! Collection-Independent Automatic Keyword Extractor', in *European Conference on Information Retrieval* (Springer, 2018), pp. 806–10.

4 Baoli Li and Liping Han, 'Distance Weighted Cosine Similarity Measure for Text Classification', in *International Conference on Intelligent Data Engineering and Automated Learning* (Springer, 2013), pp. 611–18.

5 'Précision et rappel', *Wikipédia*, 2020 <https://fr.wikipedia.org/w/index.php?title=Pr%C3%A9cision_et_rappel&oldid=169870260> [accessed 5 June 2020].

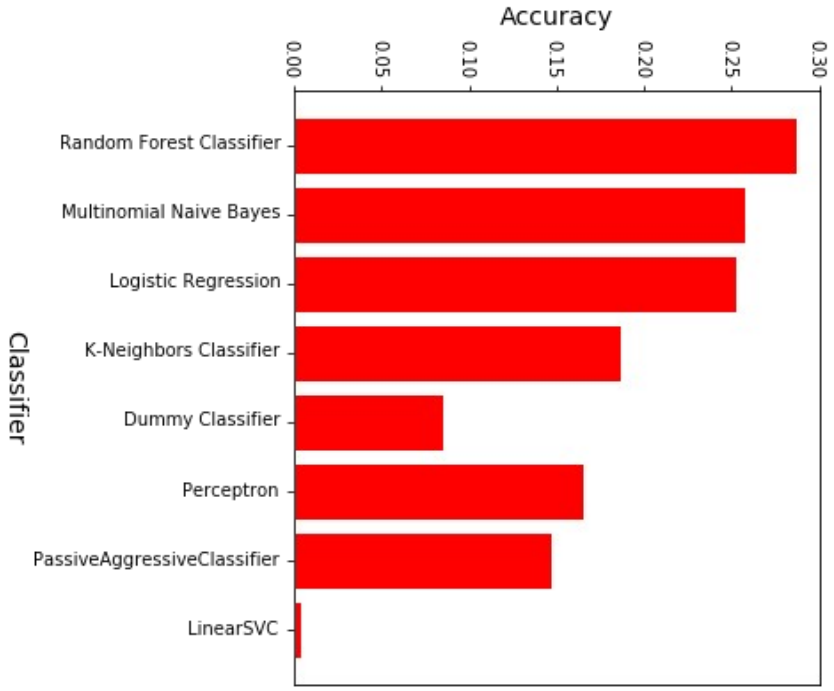


Figure 5. Exactitude (‘‘*accuracy*’’) de chaque modèle.

Par contre il présente des inconvénients par rapport au calcul car il occupe presque 10 Gb de RAM si on modèle un corpus de 220 tokens et 110 mots clés. Si on réduit les dimensions du corpus avec une analyse des composants principaux (ACP) tous les modèles perdent fortement de la précision. En fonction de la puissance du système on pourrait donc choisir soit le Random Forest soit un autre algorithme un peu moins performant mais beaucoup moins lourd comme la Régression Logistique. Si on compare la mesure de similitude cosinus des vecteurs de terme avec celles des modèles non-supervisés, les modèles supervisés donnent des meilleurs résultats. La similitude cosinus est de 0.616 pour le Random Forest, et de 0.564 pour la Régression Logistique.

7. Conclusions

Dans cette étude la meilleure représentation vectorielle du corpus des questions téléchargées depuis Stackoverflow était la matrice des termes TF-IDF. Le nombre des thèmes principaux du corpus obtenus par des modèles non-supervisés LDA et NMF est 50-60. La meilleure performance pour l’extraction non-supervisée des tags était obtenue avec le paquet ‘‘YAKE!’’. Parmi les modèles supervisés le Random Forest (optimisé) est le meilleur pour l’exactitude des prédictions, par contre il est assez lourd pour les système avec la puissance actuelle des ordinateurs. Pour un API (Interface pour la programmation d'applications) script que donne une prédiction automatique des mots-clés des questions j’ai utilisé la Régression Logistique comme modèle supervisé que donne une performance proche a celle du Random Forest. Le script donne aussi les mots-clés prédits par ‘‘YAKE!’’et est hébergée par un serveur à l’adresse http://giulio.gd:5924/api_message.