

# StackOverFlow

Catégoriser automatiquement  
des questions

*Rapport*

**Sofiane Mouhab**  
**Janvier 2022**

## 1 - Introduction

1.1 - Objectif

1.2 - Méthode

## 2 - Traitement des données

2.1 - Téléchargement des données

2.2 - Préparation des données

*2.2.1 - Analyse des tags*

*2.2.2 - Analyse des messages et du titre*

## 3 - Features engineering

3.1 - Tf-Idf

3.2 - Countvectorizer

3.3 - Réduction de dimensions

## 4 - Modélisation

4.1 - Présentation

4.2 - Non-Supervisée

*4.2.1 - LDA*

*4.2.2 - NMF*

4.3 - Supervisée

4.4 - Semi-Supervisée

## 5 - Résultats

5.1 - Métriques

*5.1.1 - Cohérence Score*

*5.1.2 - Jaccard index*

5.2 - Meilleur modèle

## 6 - Déploiement du modèle

6.1 - GitHub

6.2 - Django

6.3 - Heroku

6.4 - Visualisation

## 7 - Conclusion

# 1 - Introduction

Stack Overflow est un site célèbre de questions-réponses liées au développement informatique. Pour poser une question sur ce site, il faut entrer plusieurs tags de manière à retrouver facilement la question par la suite. Pour les utilisateurs expérimentés, cela ne pose pas de problème, mais pour les nouveaux utilisateurs, il serait judicieux de suggérer quelques tags relatifs à la question posée.

## 1.1 - Objectif

Amateur de Stack Overflow, qui vous a souvent sauvé la mise, vous décidez d'aider la communauté en retour. Pour cela, vous développez **un système de suggestion de tag** pour le site. Celui-ci prendra la forme d'un algorithme de machine learning qui assigne automatiquement plusieurs tags pertinents à une question.

## 1.2 - Méthode

- Récupération des données
- Traitement des données
- Modélisation non-supervisé
- Modélisation semi-supervisé
- Modélisation supervisé
- Création API
- Déploiement d'une API

# 2 - Traitement des données

Les données récoltées sont évidemment imparfaites, "brutes", ces dernières nécessitent donc une phase essentielle afin de poursuivre notre travail, un nettoyage, Passons en revue les différentes étapes qui nous permettent de travailler sur une base propre et solide.

## 2.1 - Téléchargement des données

Nous avons à notre disposition l'outil "Star Exchange Explorer" qui permet via des requêtes SQL d'importer des données. Il existe une limite pour éviter les données trop volumineuses.

Via quelques filtres (date, Id...), nous sommes en mesure de télécharger quelques fichiers .csv (11 pour être précis) qu'il sera nécessaire de concaténer pour arriver à une base d'environ 200.000 posts du site StackOverFlow.

```
EX: SELECT * FROM posts WHERE Id < 50000
```

## 2.2 - Préparation des données

Nous sommes donc en présence d'une base de 200.000 questions répartis en 23 colonnes (ci-dessus)

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

Etape 1 - On ne retient que les variables nécessaires à savoir :

- Body - Question posé par l'utilisateur
- Title - Titre de la question
- Tags - Tags associés à la question

Etape 2 - On procède alors à de nouveaux nettoyages:

- Suppression des balises HTML
- Suppression des questions sans Tags
- Regroupement des titres et questions
- Suppression des doublons

A ce stade notre base possède 130.000 items soit 30% de moins qu'au départ

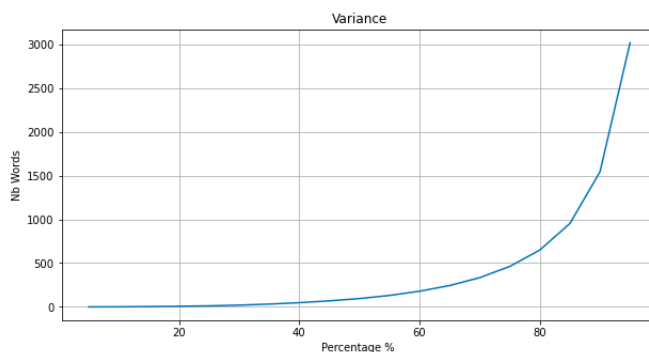
Etape 3 - On travaille enfin sur le sens des mots, via différents procédés :

- Suppression des majuscules
- Suppression des nombres
- Suppression de la ponctuation
- Suppression des StopWords (*ensemble des mots non-nécessaire*)
- Lemmatisation (*réduction des mots à leur forme essentielle*)

### 2.2.1 - Analyse des tags

Sur les 130.000 questions que nous sommes en mesure d'étudier, On constate au niveau de la répartition des tags :

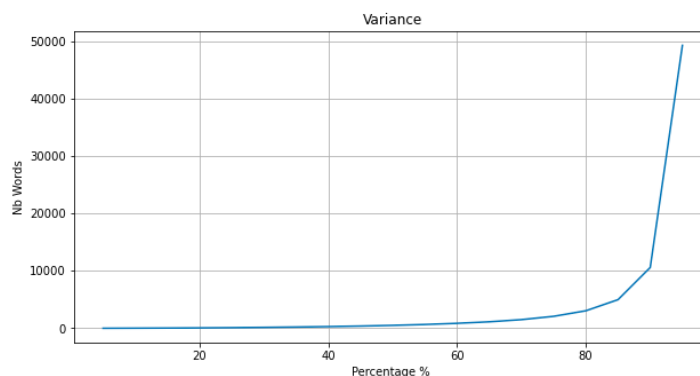
- 12.000 tags différents
- 3 tags par question en moyenne
- Un fort déséquilibre
- 462 Tags couvent 75% des questions



### 2.2.2 - Analyse des messages et du titre

De même, On constate au niveau de la répartition des questions/Titres :

- 300.000 mots différents
- 60 mots par question en moyenne
- 2000 Tags couvent 75% des questions



## 3. Features engineering

### 3.1 - Tf-Idf

Nous utiliserons la fonction *TfidfVectorizer* du module *feature\_extraction.text* de *sklearn* pour transformer la variable « *Texte* » en un *ndarray* de dimension  $(m, t)$  où  $m$  est le nombre de questions et  $t$  le nombre de tokens présent dans la variable « *Texte* ». La valeur se trouvant à la  $i$ ème ligne et la  $j$ ème colonne du *ndarray* est le nombre  $X$  défini par:

$$X = bow \times \log \left( \frac{m}{m'} \right) T m'$$

Où : **bow** = nombre de fois le token  $j$  est présent dans la question  $i$

$T$  = nombre de tokens de la question  $i$ ;

$m$  = nombre de questions e

$m'$  = nombre de questions comportant le token  $j$ .

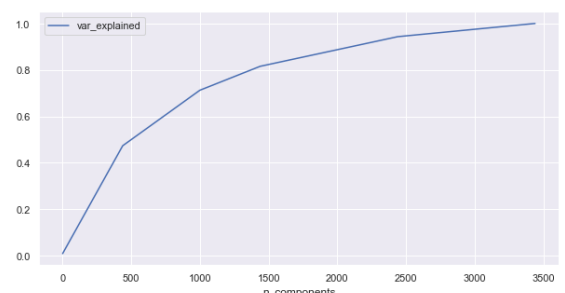
Ce traitement permet de donner un poids plus important aux tokens les moins présents dans le corpus et donc considérés comme plus porteurs d'information.

### 3.2 - CountVectorizer

Scikit-learn CountVectorizer utilisé pour convertir une collection de documents texte en un vecteur de nombre de termes / jetons. Il permet également le prétraitement des données textuelles avant de générer la représentation vectorielle. Cette fonctionnalité en fait un module de représentation d'entités très flexible pour le texte.

### 3.3 - TruncatedSVD

Afin d'accélérer le traitement des modèles, nous appliquons une réduction de dimension (Truncated SVD) qui nous permet de passer de 3500 à 1000 variables pour un taux d'explicabilité de 70%



## 4 - Modélisation

### 4.1 - Présentation

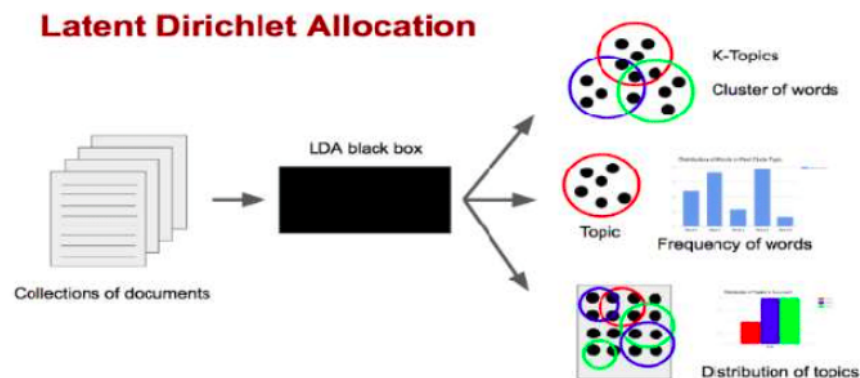
La modélisation consiste à prédire aléatoire les tags qui seront affectés à une question donnée.

### 4.2 - Non Supervisée

L'apprentissage non supervisé permet de détecter des topics dans un corpus donné. Nous utiliserons pour notre projet, l'algorithme *Latent Dirichlet Allocation (lda)* et l'algorithme *Non-negativ Matrix Factorization (nmf)*.

#### 4.2.1 - LDA

L'algorithme *lda* est basé sur un modèle probabiliste : un document est composé de différents topics suivant une certaine loi de distribution. Ces topics sont composés à leur tour de tokens suivant une autre loi de distribution. Le modèle *lda* détermine, pour un nombre de topics donnés  $k$ , la distribution des topics dans les documents et la distribution des tokens dans les topics.



#### 4.2.2 - NMF

L'algorithme *nmf* est basé sur une approche de résolution par itération visant à réduire la distance entre  $X$  et  $H*W$

$$\begin{matrix} & W \\ \begin{bmatrix} \square & \square \\ \square & \square \\ \square & \square \\ \square & \square \end{bmatrix} & \times & \begin{matrix} & H \\ \begin{bmatrix} \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square \end{bmatrix} \end{matrix} & \approx & \begin{matrix} & V \\ \begin{bmatrix} \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square \end{bmatrix} \end{matrix} \end{matrix}$$

## 4.3 - Supervisée

Différents modèles, linéaires (*régression logistique* et *SVM*) et non linéaires (*random forest* et *réseau de neurones*) ont été entraînés sur les différents trains. Quelques hyper-paramètres ont été optimisés par cross-validation sur le train set. Le module *scikit learn* de python nous fournira les algorithmes nécessaires pour implémenter la régression logistique, le SVM et le random forest:

- SGDClassifier
- LogisticRegression
- LinearSVC
- RandomForestClassifier
- AdaBoostClassifier
- ExtraTrees Classifier
- LGBMClassifier

## 4.4 - Semi Supervisée

Nous allons combiner l'apprentissage non supervisé et supervisé pour faire l'apprentissage semi-supervisé. Pour ce faire, nous utiliserons alors les sorties des algorithmes non supervisés (LDA et NMF) en entrée des algorithmes de classification supervisés.

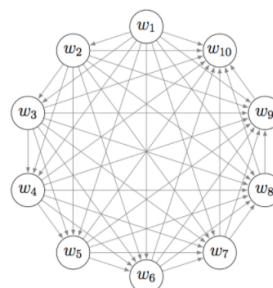
# 5 - Résultats

## 5.1 - Métriques

### 5.1.1 - Cohérence Score

Afin de déterminer le nombre de topics qui assure la plus grande performance des modèles d'apprentissage *lda* et *nmf*, nous utiliserons le **score de cohérence**.

Les mesures de cohérence de sujet évaluent un seul sujet en mesurant le degré de similitude sémantique entre les mots.

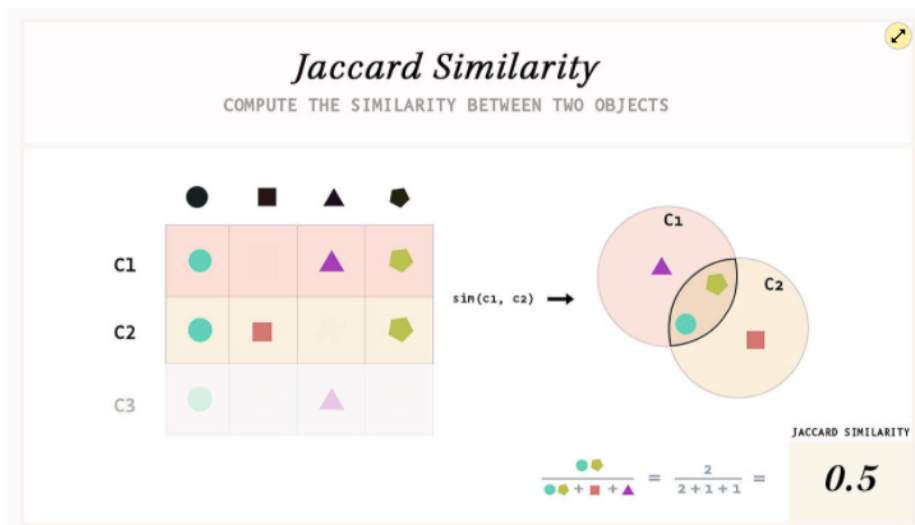




### 5.1.2 - Jaccard Index

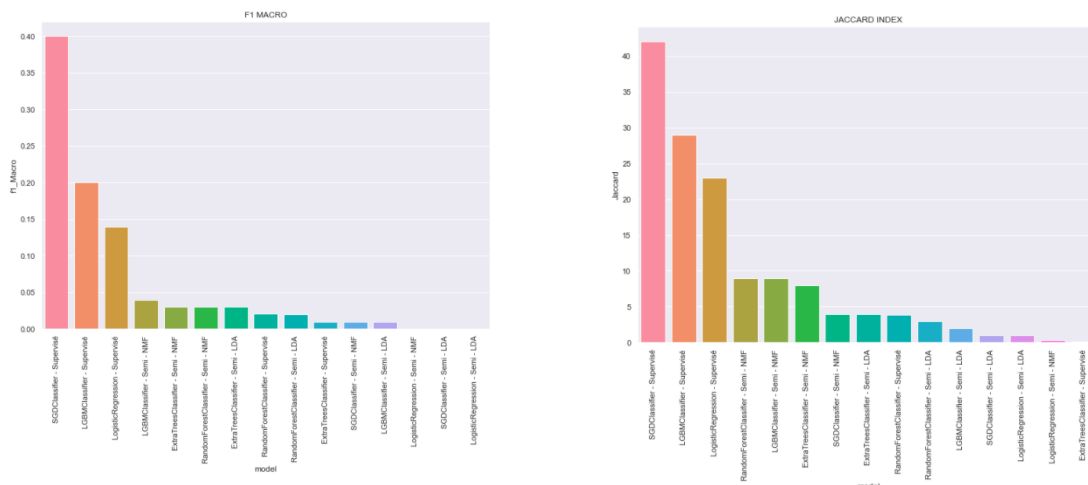
La métrique que nous choisirons d'utiliser afin de mesurer la performance des modèles d'apprentissage supervisé et semi-supervisé dans le cas du multi-label sera le jaccard Score.

L'indice de Jaccard, également connu sous le nom de coefficient de similarité de Jaccard, est une statistique utilisée pour comprendre les similitudes entre les ensembles d'échantillons



## 5.2 Meilleur modèle

Après avoir réuni l'ensemble des résultats, il est donc temps de choisir un modèle en particulier. A partir des données, nous appliquons un compromis entre la performance pure mais aussi le temps nécessaire à la réalisation des prédictions.



Nous avons donc décidé d'utiliser le modèle SGDC Classifier. A partir de là, nous effectuons un rapide Grid Search CV afin de déterminer le modèle le plus performant :

```
SGDClassifier(class_weight={0: 0.3, 1: 0.7}, eta0=10)
Accuracy : 0.3629
Macro f1 score : 0.3105744288954594
Micro f1 score : 0.5586267325155329
Jacard score: 41.31890072176892
Hamming loss: 1.1082
---
time: 17.6 s (started: 2021-12-31 14:42:35 +01:00)
```

Il est temps d'exporter l'ensemble de nos modèles ( SGDC Classifier, TruncatedSVD, Td-Idf Vectorizer afin de déployer notre application)

## 6 - Déploiement du modèle

### 6.1 - GitHub

L'ensemble du code est sauvegardé sur le site GitHub, qui permet une gestion de version collaborative



Disponible ici : [https://github.com/rkoper/IML\\_projet\\_5](https://github.com/rkoper/IML_projet_5)

### 6.2 - Django

Nous utilisons dans un premier temps le cadre de développement web : Django. Cette interface est open-source en python.



### 6.3 - Heroku

Pour la mise en ligne de l'Api, le choix se porte sur Heroku, qui nous permet de déploiement d'applications.

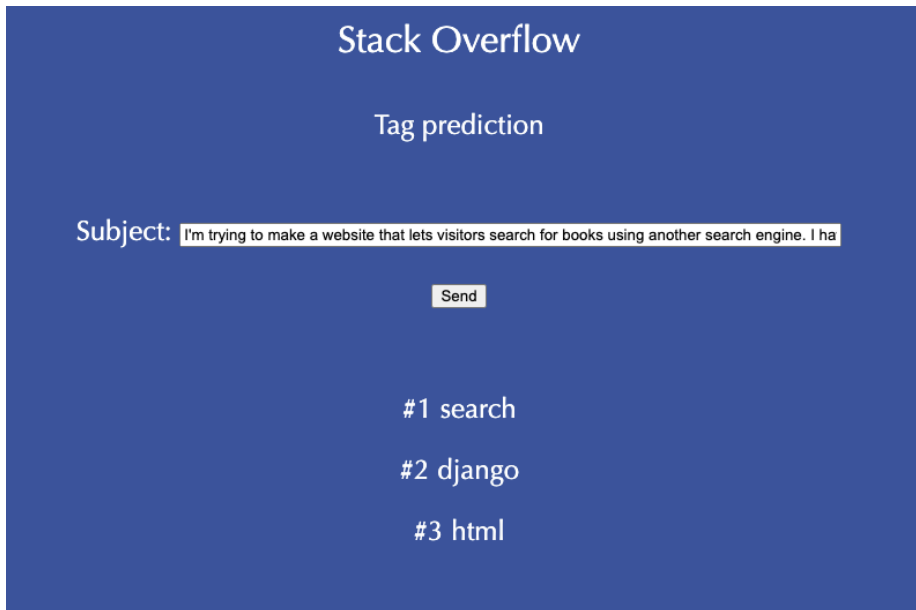


On retrouve le travail à l'adresse suivante :

<https://desolate-reaches-73171.herokuapp.com/>

## 6.4 - Visualisation

Une interface sobre et efficace pour la prédiction :

A screenshot of a web interface for Stack Overflow. The background is a solid blue color. At the top, the text "Stack Overflow" is written in white. Below it, "Tag prediction" is also in white. Further down, the word "Subject:" is in white, followed by a white text input field containing the text "I'm trying to make a website that lets visitors search for books using another search engine. I ha". Below the input field is a small white button with the text "Send". At the bottom, three predicted tags are listed in white: "#1 search", "#2 django", and "#3 html".

Stack Overflow

Tag prediction

Subject:

#1 search

#2 django

#3 html

## 7. Conclusion

Après avoir passé en revue ces différentes approches, il semble clair que la meilleure méthode est la classification supervisée avec le modèle SDG Classifier.

Néanmoins quelques améliorations ou tests peuvent être apporté :

- Amélioration de la corrélation entre tags
- Explorer d'autres méthodes (Bow, Pos, word embedding...)
- Améliorer nos filtres (nombres et ponctuation en particulier)
- Approfondir le Deep Learning