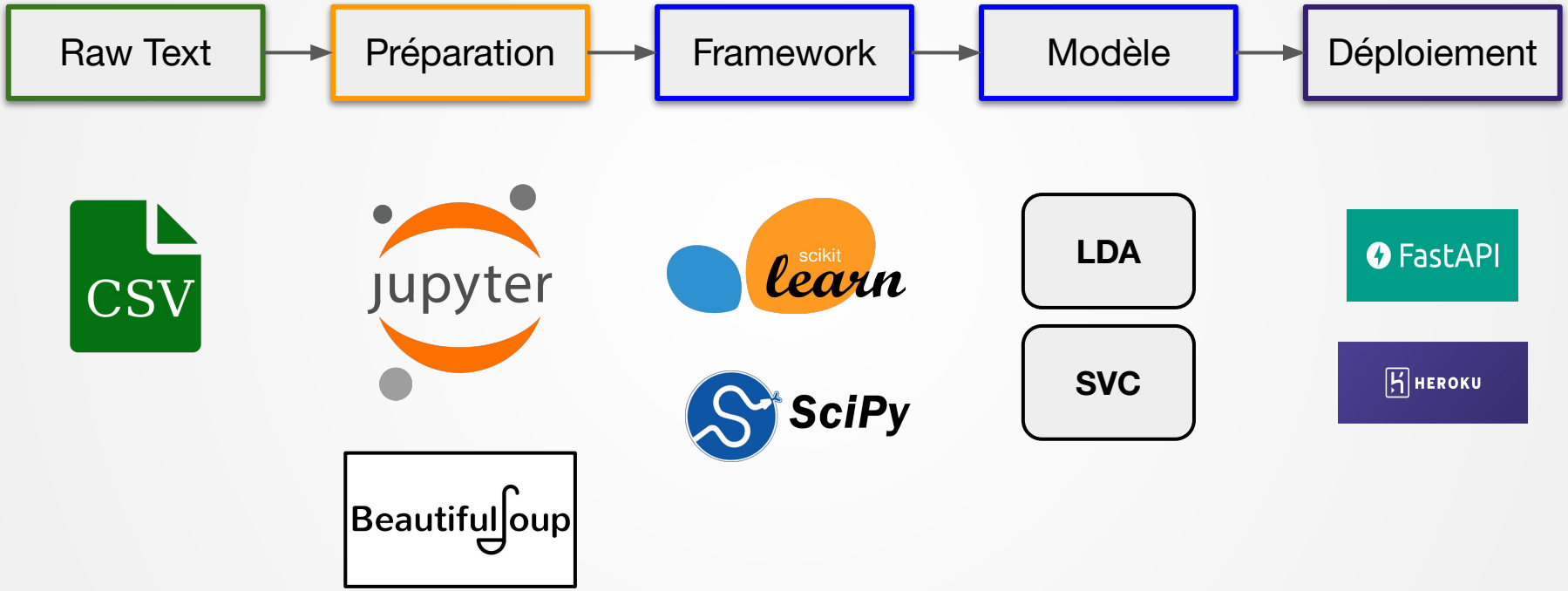
Projet 5
Parcours Machine Learning

Catégoriser automatiquement des questions

LAHMER Arslane

Sommaire

Schéma déploiement



I. Introduction

Stackoverflow est une célèbre plateforme d'entraide liée au monde de l'informatique. Pour poser des questions sur le site, il est demandé à l'utilisateur d'entrer des tags en rapport avec la question afin de la retrouver par la suite. Pour les nouveaux utilisateurs cela pourrait s'avérer un véritable challenge, raison pour laquelle il serait judicieux d'orienter l'utilisateur en lui suggérant des tags.

Pour ce faire nous allons développer un algorithme de Machine Learning qui fera office de système de suggestion de tags.

II. Prétraitement des données

Dans un premier temps nous allons récupérer nos données grâce à l'outil d'export de données "stackexchange explorer" mis en place par Stackoverflow. Pour cela nous exécuterons une requête sql. Nous prendrons soin de filtrer les questions sur la variable Score (>5) ainsi que sur la longueur des Tags (minimum 3) afin de faciliter l'entraînement à notre modèle en lui fournissant les données les plus pertinentes possibles.

Le jeu de données comportera 50.000 entrées (=questions) et contiendra des informations sur le nombre de vues, de réponses, de votes approbateurs aux réponses entre autres.

II. Prétraitement des données (suite)

Une fois les données récupérées et les features sélectionnées (Body et Title dans notre cas), nous allons les prétraiter avant de les représenter.

Le prétraitement des variables Body et Title consistera à :

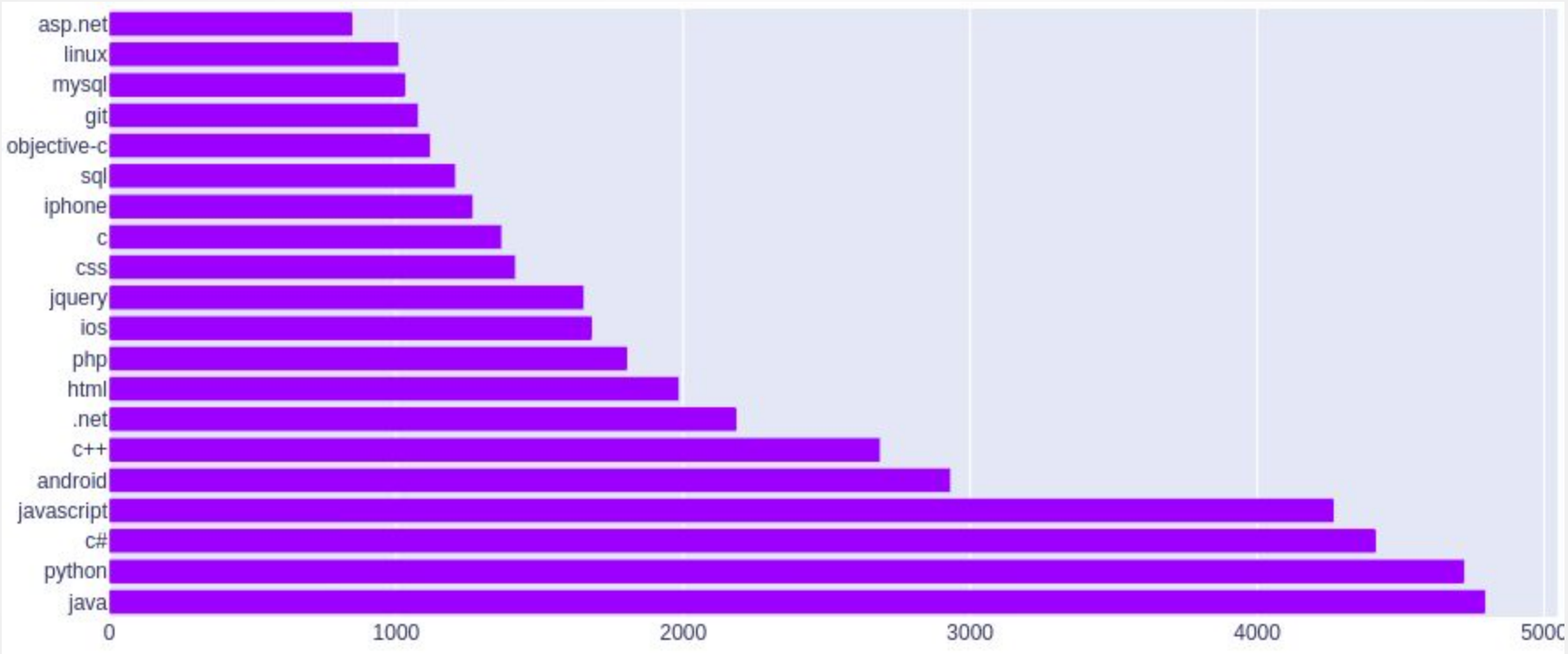
- Filtrer les balises html à l'aide de l'outil BeautifulSoup ainsi que les url.
- Normaliser les documents (remplacer les contractions par les expressions d'origine, exemple : "what's" sera transformée en "what is").
- Raciniser (ou lemmatizer) documents (remplacement des mots par leur radical ou racine).
- Suppression des stopwords qui correspondent aux mots trop fréquents et qui n'ont que trop peu d'intérêt dans la détermination des tags.

En ce qui concerne les tags, ils seront nettoyés (filtrage des balises "><") et transformés en listes. Ne seront gardés que les 100 plus courants pour faciliter l'apprentissage.

III. Analyse exploratoire

A. Analyse univariée

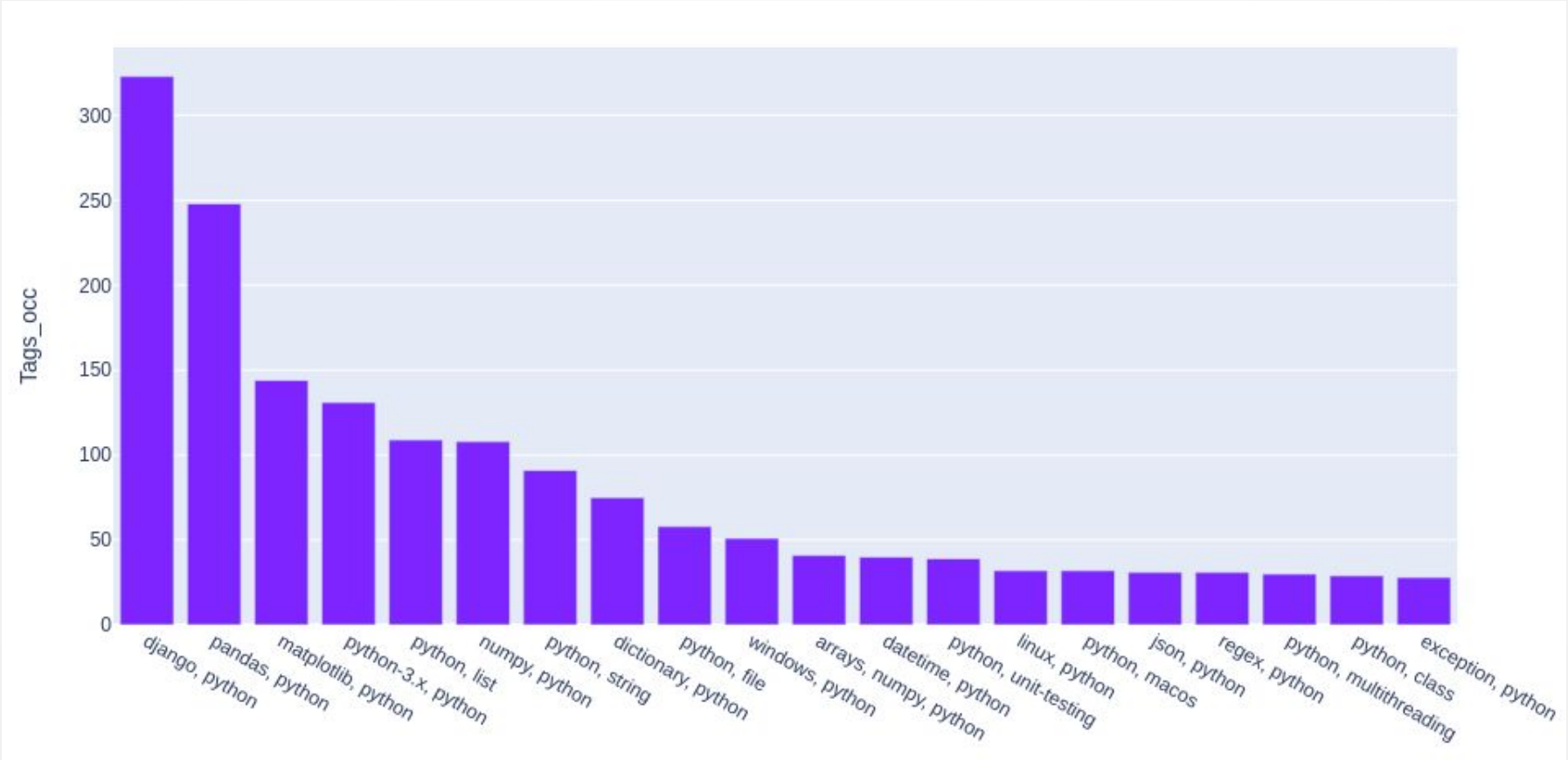
Le graphe correspond au comptage des tags les plus fréquents.

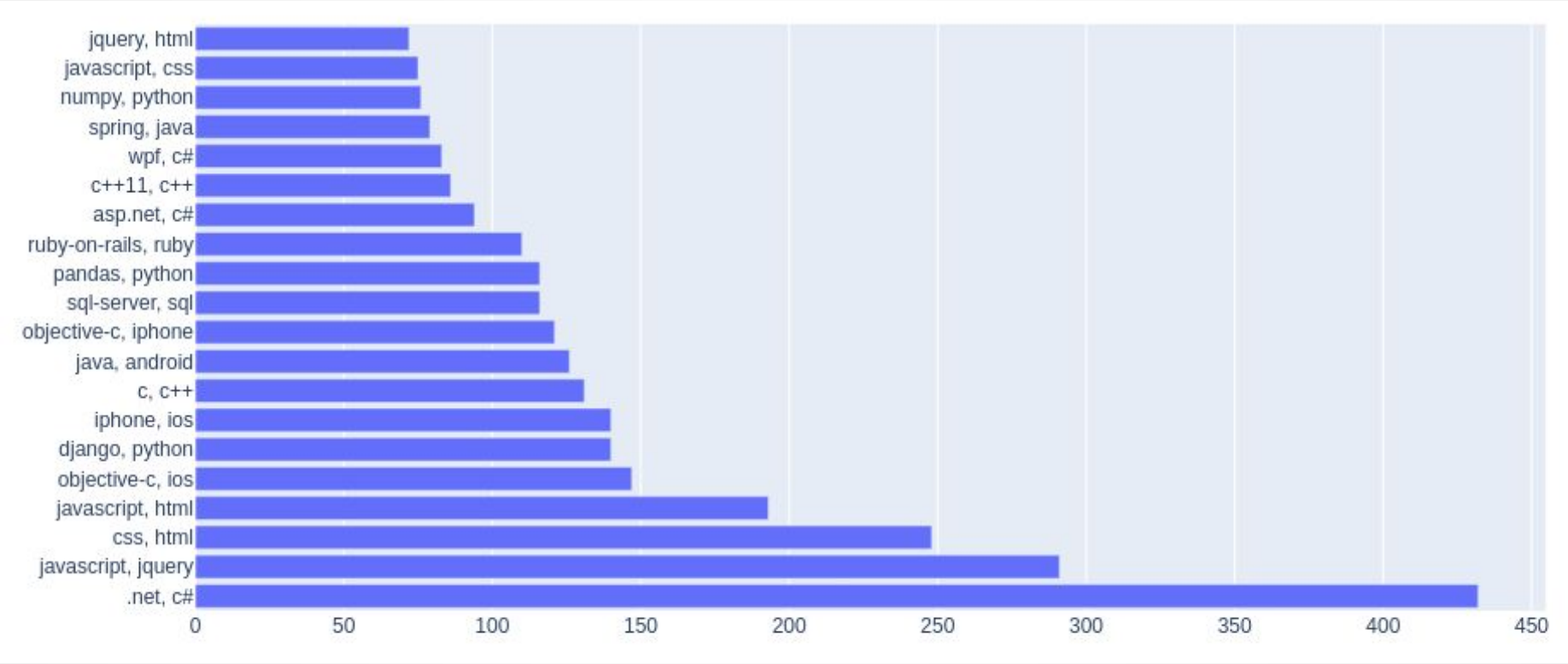

asp.net
linux
mysql
git
objective-c
sql
iphone
c
css
jquery
ios
php
html
.net
c++
android
javascript
c#
python
java
0
1000
2000
3000
4000
5000

III. Analyse exploratoire

B. Analyse multivariée

L’analyse multivariée nous révèlera les co-occurences les plus courantes ainsi que les combinaisons de tags les plus fréquentes.


Tags_occ
300
250
200
150
100
50
0
django, python
pandas, python
matplotlib, python
python-3.x, python
python, list
numpy, python
python, string
dictionary, python
python, file
windows, python
arrays, numpy, python
datetime, python
python, unit-testing
linux, python
python, macos
json, python
regex, python
python, multithreading
python, class
exception, python


jquery, html
javascript, css
numpy, python
spring, java
wpf, c#
c++11, c++
asp.net, c#
ruby-on-rails, ruby
pandas, python
sql-server, sql
objective-c, iphone
java, android
c, c++
iphone, ios
django, python
objective-c, ios
javascript, html
css, html
javascript, jquery
.net, c#
0
50
100
150
200
250
300
350
400
450

IV. Représentation du document en bags-of-words (1/2)

Avant de fournir les données à notre modèle, il nous faut donc les représenter numériquement.

Nous avons opté pour la technique du bags-of-words qui consiste à représenter les document (l'ensemble des questions dans notre cas) en un histogramme des occurrences des mots qui le composent. Chaque document est représenté par un vecteur de la taille du dictionnaire (taille qu'on limitera en ne gardant que les mots qu'on jugera utiles dans l'opération de prédiction).

La manière la plus simple pour ce faire serait de faire un comptage simple de tous les mots du document utilisés sans prise en compte du contexte (ordre, utilisation etc.).

Il serait néanmoins plus judicieux de pondérer les occurrences de mots en fonction de leur fréquence d'apparition dans un type donné de document.

IV. Représentation du document en bags-of-words (2/2)

En effet, si un mot donné est fréquent dans tous les types de documents, il ne sera pas indicatif du type de document à l'inverse des mots qui ne seront fréquents que dans un type donné de document.

On aura recours au transforme tf-idf (Term-Frequency - Inverse Document Frequency) utilise comme indicateur de similarité *l'inverse document frequency* qui est l'inverse de la proportion de document qui contient le terme, à l'échelle logarithmique.

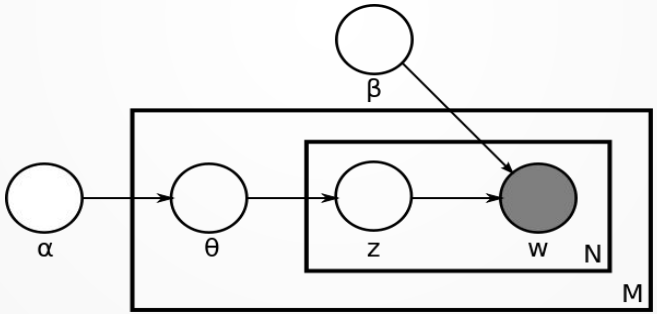
Formule du tf-idf :

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

V. Modèle non-supervisé Latent Dirichlet Allocation (1/2)

L’apprentissage non-supervisé permet de modéliser des sujets abordés dans des documents et d’assigner les sujets détectés à ces documents

LDA est un modèle probabiliste génératif dont la formule est :

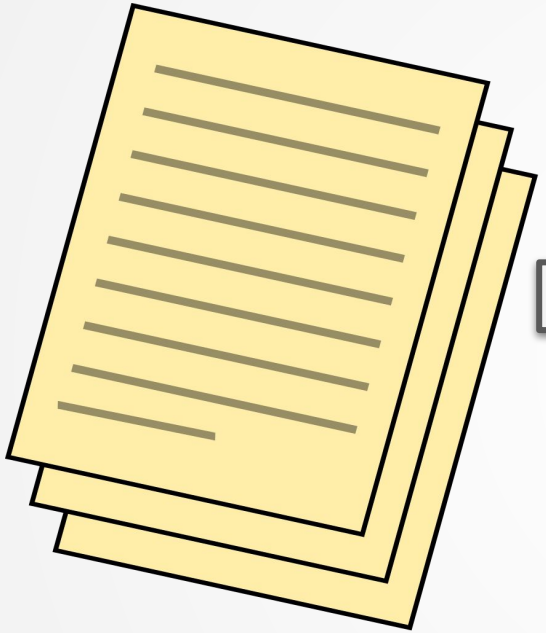


V. Modèle non-supervisé Latent Dirichlet Allocation (2/2)

L'objectif étant de retrouver les distributions de mots sur les différents thèmes, les différentes proportions de thèmes pour chaque document, les proportions d'apparition d'un thème sur le corpus. Tout cela à partir des différents documents. Ce qui nous permet par la suite de déterminer le thème d'un document, les mots les plus associés à certains thèmes

En appliquant notre modèle à notre corpus qui représente 50.000 questions, nous afficherons les mots les plus représentatifs par sujet modélisé. On en jugera de leur pertinence par leur compréhensibilité de la part de l'utilisateur moyen.

En guise d'évaluation de performance du modèle, nous opèrerons une intersection entre les 100 tags les plus courants et les mots les plus représentatifs par sujet modélisé.



```
--------------------------------
Topic 11:

--------------------------------
Topic 12:
ruby database sql mysql
--------------------------------
Topic 13:
date database sql
--------------------------------
Topic 14:
git file
--------------------------------
Topic 15:
eclipse xcode file android
--------------------------------
Topic 16:
list string
--------------------------------
Topic 17:
database sql mysql
--------------------------------
```

VI. Modèle supervisé

Une fois qu'on aura représenté les features (Body et Title) choisies pour notre modèle de prédiction en bags of words, on empilera ("stack") les matrices correspondantes à chaque feature.

En ce qui concerne les tags, ils seront binarisés par le moyen de MultiLabelBinarizer qui va affecter une séquence binaire à chaque combinaison de tags.

Nos données sont désormais prêtes pour l'entraînement. Ils seront divisés en données d'entraînement et en labels

Nous évaluerons l'efficacité de notre modèle par le moyen de la métrique score jaccard qui est la plus adapté à la problématique des prédictions multilabels.

Après avoir testé des modèles de la famille des algorithmes linéaires, il s'avère que celui ayant le meilleur score est le SVC qui arrive à prédire en moyenne la moitié des tags.

- linux : clipboard retain sed mutex job mouse grep detach linux
- ruby : older rake privilege hash rspec gem activerecord
- python : dict virtualenv flask tkinter sqlalchemy matplotlib pandas numpy
- json : serialize job expect backbonejs efficient conditionally
- mongodb : sync query collection foreign expressjs
- pandas: solution commas groupby letter column none
- database: database trouble increase outer dump wrapper soft kind

TITRE

Change a HTML5 input's placeholder color wil

CONTENU

Chrome supports the placeholder attribute on input[type=text] elements (others probably do too).

But the following CSS doesn't do anything to the placeholder's value:

Nettoyer

Soumettre

OUTPUT 1.7s

['css', 'html']

Signaler

VII Déploiement

A l'issue de notre choix qui s'est porté sur le modèle Support Vector Machine en raison de son bon compromis performance/poids, on crée un repository sur github afin de pousser notre code en vue du déploiement sur la plateforme Heroku.

En raison de sa simplicité d'utilisation ainsi qu'à sa familiarité avec Python, le Framework FastAPI sera choisi pour mettre au point notre API.

On commence par importer les transformeurs qu'on aura fité sur le jeu de données ainsi que notre modèle de prédiction SVC, avant de regrouper toutes les opérations nécessaires à la prédiction en une seule fonction. Pour s'assurer du fonctionnement de l'API, de l'absence de bugs notamment, on fait différents tests de requêtes get et post à l'aide de Postman. Notre code est fin prêt pour être déployé.

Avant cela il reste à créer un fichier texte requirements pour indiquer au serveur distant quels packages installer pour faire fonctionner l'API ainsi qu'un fichier Procfile et gitignore et de les pousser sur notre repository Github avant le déploiement sur Heroku.