Rapport Projet 5

Catégorisation automatique des questions

Sommaire

I. Objectif

L’objectif de ce projet est de développer un système de suggestion de tag lorsque des utilisateurs de StackOverFlow posent des nouvelles questions.

Vous trouverez dans ce rapport, une explication des traitements effectués sur les données textuelles et l’étude des différents modèles testés. Ensuite une explication du choix du modèle sera faites.

II. Traitements sur les données textuelles

Afin de pouvoir développer ce système de tag, il faut partir de questions et tags déjà existants. Le texte étant brut, il faut faire un premier nettoyage pour récupérer les mots principaux.

II.1. Nettoyage générale

Le premier nettoyage permet de standardiser le texte. Pour cela on met en minuscule tout le texte, on supprime les liens et les balises de code, on supprime les ponctuations (sauf le # pour conserver le langage c#), on supprime les chiffres, et on en lève les contractions (exemple : i’m en i am)

Exemple de nettoyage générale

Avant	->	Après
` I have a class called <code>Order</code> which has properties such as <code>OrderId</code>, <code>OrderDate</code>, <code>Quantity</code>, and <code>Total</code>. I have a list of this <code>Order</code> class: \n\n<pre><code>List<Order> objListOrder = new List<Order>();\nGetOrderList(objListOrder); // fill list of orders\n</code></pre>\n\n Now I want to sort the list based on one property of the <code>Order</code> object, for example I need to sort it by the order date or order id. \n\n How can i do this in C#? \n`		`i have a class called which has properties such as and i have a list of this class now i want to sort the list based on one property of the object for example i need to sort it by the order date or order id how can i do this in c#`

II.2. Suppression de catégories de mot

Afin de garder l'information essentiel, les verbes, les adjectifs, les adverbes vont être supprimés.

Exemple de suppression de catégories de mot

Avant	->	Après
`i have a class called which has properties such as and i have a list of this class now i want to sort the list based on one property of the object for example i need to sort it by the order date or order id how can i do this in c#`		`class properties list class list property object example order date order c#`

II.3. Tokenisation

Pour continuer le processus de nettoyage une tokenisation est nécessaire pour séparer les mots, tout en gardant le C et le # ensemble s'il le faut.

Exemple de tokenization

Avant	->	Après
`class properties list class list property object example order date order c#`		`[class, properties, list, class, list, property, object, example, order, date, order, i, c#]`

II.4. Lemmatisation

Une fois les mots séparés, une lemmatisation (ou racinisation) est faites. Ce processus consiste à garder la racine des mots et donc garder uniquement le sens des mots.

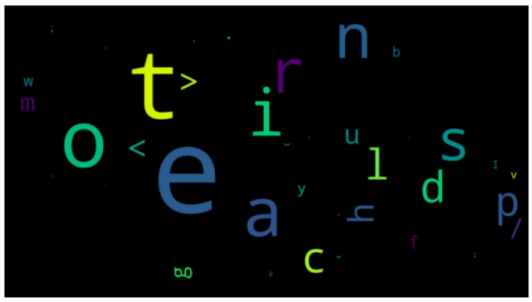
Exemple de lemmatisation

Avant	->	Après
`[class, properties, list, class, list, property, object, example, order, date, order, i, c#]`		`[class, property, list, class, list, property, object, example, order, date, order, i, c#]`

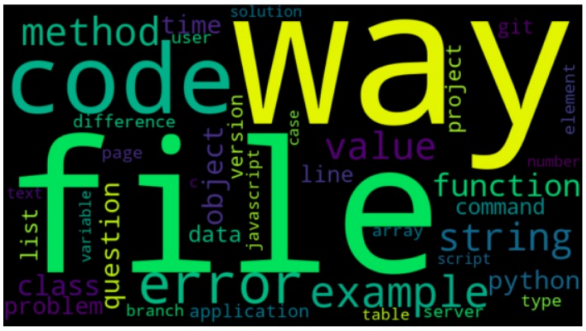
II.5. StopWords

Afin de garder les mots ayant uniquement une valeur informative, on effectue un nettoyage supprimant les “stopwords”. Les stopwords sont les mots très courant d’une langue, la librairie NLTK comporte une liste de ces mots par défaut.

Word Cloud des 40 mots trouvés dans les body non nettoyés sur StackOverflow



Word Cloud des 40 mots trouvés dans les body nettoyés sur StackOverflow



III. Les modélisations

Une fois le texte nettoyé et contenant uniquement l’information utile, il faut trouvé le bon modèle afin de prédire les tags associés aux questions.

Pour cela il faut transformer le texte en structure compréhensible par le système de modélisation. Et ensuite il faut tester différents modèles afin de trouver le plus efficace.

III.1. Représentation du corpus

III.1.a) Bag of Words

La première représentation statistique de nos textes est par l'intermédiaire de la méthode bag of words. Elle permet de créer un vecteur du dictionnaire du corpus, en indiquant le nombre de fois que chaque mots apparait dans le texte.

Exemple de bag of words

Texte ->

```
[past, microsoft, web, application, stress, tool, pylot,
stress, test, web, application, home, page, login,
script, site, walkthrough, ecommerce, site, item, cart,
checkout, homepage, handful, developer, problem,
scalability, problem, stage, launch, url, tool,
microsoft, homer, microsoft, web, application, stress,
tool, pylot, report, tool, sense, hour, kind, load,
site, bug, bottleneck, instance, web, server,
misconfigurations, tool, success, approach, part, kind,
formula, number, user, app, number, stress, test,
application, stress, test, web, application]
```

Bag Of Words

```
[[('app', 1),
  ('application', 5),
  ('approach', 1),
  ('bottleneck', 1),
  ('bug', 1),
  ('cart', 1),
  ('checkout', 1),
  ('developer', 1),
  ('formula', 1),
  ('handful', 1),
  ('home', 1),
  ('homepage', 1),
  ('hour', 1),
  ('instance', 1),
  ('item', 1),
  ('kind', 2)…]]
```

III.1.b) TF IDF (Term Frequency Inverse Document Frequency)

Une autre représentation statistique de nos textes peut se faire avec la méthode TF IDF. Cette fois-ci le calcul n'est pas uniquement fait sur la fréquences d'apparition dans la phrase mais elle est aussi pondéré par la fréquence d'apparition dans le corpus (savoir si c'est un mot rare ou commun dans le corpus)

$$TFIDF_{t,d,D} = TF_{t,d} \times IDF_{t,D}$$

- $TFIDF_{t,d,D}$: Importance d'un terme t dans un document d
- $TF_{t,d}$: Fréquence d'un terme t dans un document d
- $IDF_{t,D}$: Importance du terme t dans l'ensemble des documents D

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

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Texte

```
[past, microsoft, web, application, stress, tool, pylot,
stress, test, web, application, home, page, login,
script, site, walkthrough, ecommerce, site, item, cart,
checkout, homepage, handful, developer, problem,
scalability, problem, stage, launch, url, tool,
microsoft, homer, microsoft, web, application, stress,
tool, pylot, report, tool, sense, hour, kind, load,
site, bug, bottleneck, instance, web, server,
misconfigurations, tool, success, approach, part, kind,
formula, number, user, app, number, stress, test,
application, stress, test, web, application]
```

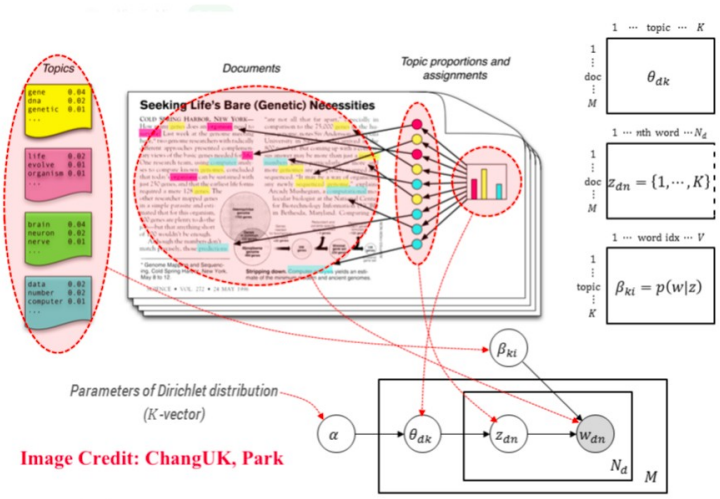
Exemple de TF IDF

```
access  action  activity  address  advantage  algorithm  alternative  \
   0.0     0.0       0.0      0.0        0.0        0.0          0.0

amount  android  angularjs  answer  api       app  application  approach  \
   0.0      0.0        0.0     0.0  0.0  0.081998     0.381566  0.103392

apps  area  argument  array  article  aspnet  attempt  attribute  \
 0.0   0.0       0.0    0.0      0.0     0.0      0.0        0.0

background  bar  base  bash  behavior  behaviour  benefit  bit  block  \
       0.0  0.0   0.0   0.0       0.0        0.0      0.0  0.0    0.0

book  bootstrap  bottom  box  branch  browser       bug  build  builtin  \
 0.0        0.0     0.0  0.0     0.0      0.0  0.117281    0.0      0.0
```

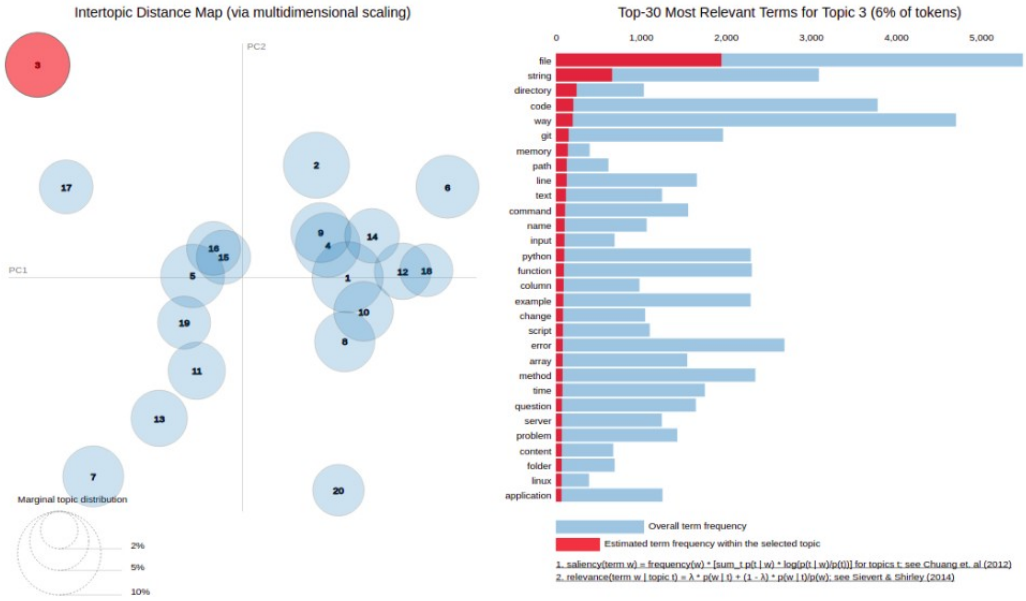
III.2. Modèles d'apprentissage

III.2.a) LDA (Latent Dirichlet Allocation)

La première méthode d'apprentissage testée est une méthode non supervisée qui se nomme LDA. C'est un modèle génératif permettant de déterminer des thèmes et la distribution de chaque mot dans les thèmes.



Résultat de prédiction faîtes avec le LDA



	MainTags	best_topic	nb_tags	y_true	y_pred
0	[performance]	7	1	[26]	[28]
1	[git]	14	1	[11]	[11]
2	[git]	15	1	[11]	[11]
3	[python, string, arrays]	7	3	[4, 28, 35]	[28, 20, 35]
4	[c, #, list]	9	3	[0, 6, 20]	[6, 16, 0]

III.2.b) Régression multi logistique

Vu que le corpus étudié comporte déjà des tags, il est possible de tester une méthode supervisé. Ici la régression multi logistique est testé. Pour cela les tags sont passés dans un MultiLabelBinarizer qui permet de binariser les tags.

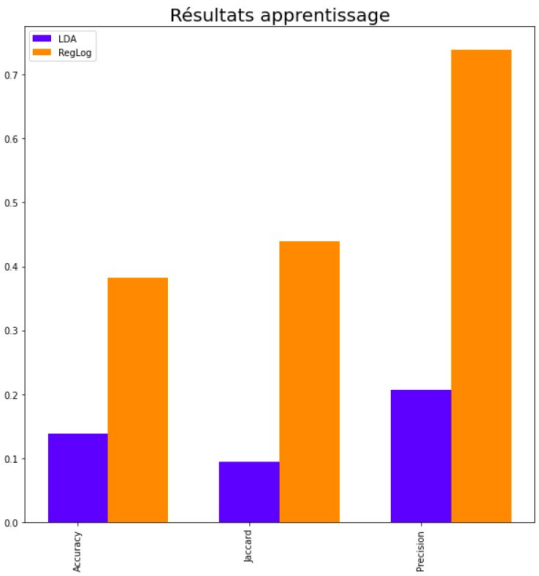
Résultat de prédiction faîtes avec la régression mutli logistique

```
Prédiction
[('css',), ('android',), ('git',), ('git',), ('arrays',)]
------------------------------
Réel
[('css', 'html'), ('android',), ('git',), ('git',), ('c++', 'string')]
```

IV. Choix du modèle

Afin de trouver le meilleur modèle, plusieurs métrics ont été comparés :

- Accuracy qui indique le pourcentage de bonnes prédictions
- Jaccard qui permet d'évaluer la similartié entre les ensembles
- Precision qui indique le pourcentage de bonnes prédictions positives



Comme attendu, avec les 3 métrics, la régression multi logistique est la méthode d'apprentissage donnant le meilleur résultat.

V. API

Afin de pouvoir se servir de cet apprentissage, une interface en ligne a été crée.

Pour cela il a fallut sauvegarder le modèle de régression multi logistique via pickle et joblib. L'interface a été faite via streamlit et joblib a permis de charger le modèle dans l'interface. Cette nouvelle interface est déployé grâce à la solution Heroku (adresse : https://app-classification-cteurio.herokuapp.com/)