Ingénieur Machine Learning

Projet 5

Catégorisez automatiquement des questions

Pelletier Gaëtan

Table des matières

I. Présentation du projet

Ce projet porte sur la manipulation et la prédiction de données textuelles.

Pour se faire, il est demandé de pouvoir déterminer les mots-clefs d'une question postée sur le forum Stack Overflow.

Pour réaliser ce projet, il a d'abord fallu effectuer un nettoyage des données, ainsi qu'une phase de préprocessing. Une fois les données prêtes à être exploitées, nous avons effectué deux types de modélisations pour prédire les mots-clefs d'une phrase. La première a été une approche purement non-supervisée. À l'opposé, la seconde a utilisé un algorithme supervisé.

Enfin, les modèles développés ont été intégrés à une API disponible via un navigateur web.

II. Récupération des données

Pour récupérer des données de Stack Overflow, ce dernier possède un outil d'export appelé « *stackexchange explorer* ». Il faut envoyer des requêtes SQL afin d'afficher les données (*figure 1*).

Pour récupérer des données de « bonne » qualité, nous allons filtrer les questions en fonction de Score, ViewCount et FavoriteCount. L'idée est que, si une question possède de bons scores, alors elle semble compréhensible, bien formulée, bien identifiée pour les autres utilisateurs. De plus, nous ne prendrons que les questions les plus récentes pour garantir un vocabulaire « moderne ». Nous effectuons cette opération afin d'obtenir 200 000 données.

```
SELECT CreationDate, Body, Tags, Title, Score, ViewCount
FROM posts
WHERE Tags IS NOT NULL
AND Title IS NOT NULL
AND Score > 0
AND ViewCount > 0
AND FavoriteCount > 0
ORDER BY CreationDate desc
```

Figure 1: Requête SQL afin de récupérer les données

III. Nettoyage et preprocessing

1. Suppressions des balises html

Les données récupérées sont encodées avec des balises html. Il est possible d'utiliser le module *re* (regular expression) afin de les supprimer.

2. Fréquence des tags

En étudiant la fréquence des différents tags utilisé, nous constatons qu'il y a un fort déséquilibre. En effet, une grande quantité de tags n'est présente qu'une seule fois dans le jeu de données. Nous faisons donc le choix de trier les tags selon leur fréquence (f*igure 2*), puis de n'en sélectionner qu'un nombre restreint. Sur les 25 019 tags, nous retenons 296 tags. Ces derniers représentent un peu moins de 60 % du total des tags.

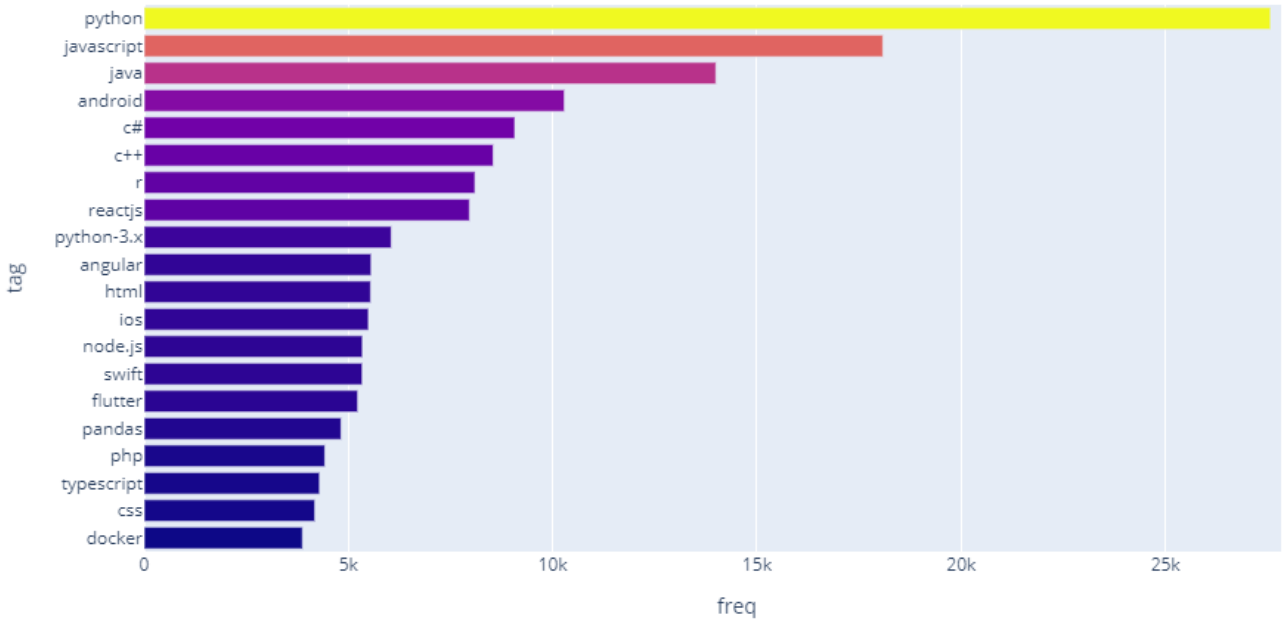

Figure 2: Fréquence des 20 tags les plus représentés

3. Filtrage des mots

a) Tokenization

Grâce à la librairie *spaCy*, nous pouvons décomposer une phrase en une succession de mots, appelés *tokens*. Les adjectifs et les adverbes n'ayant peu de valeur sémantique, nous décidons de les retirer. Nous pouvons, ensuite, effectuer un traitement plus poussé des tokens restants.

b) Lemmatization

Chaque token est analysé et transformé sous forme canonique, appelée *lemme*. Ce lemme permet de s'affranchir du genre et/ou du nombre pour chaque token. Par exemple, les mots « *petit* » et « *petites* » seront considérés identiques. Il n'y aura pas de doublon d'information.

IV. Modélisation

Pour chaque donnée, nous concaténons les questions et les titres. Nous allons, ensuite, mettre en place deux stratégies de prédictions des mots-clefs : l'une de façon non-supervisée, l'autre en utilisant un algorithme supervisé.

1. Apprentissage non supervisé

a) Algorithme TF-IDF

Lors de la création de *bag of words*, nous les pondérons selon l'apparition des mots au sein des différentes questions. Pour cela, nous utilisons l'algorithme TF-IDF (*term frequency-inverse document frequency*). Cet algorithme transforme les textes en matrices creuses. Ces dernières seront les données d'entrée de notre algorithme non-supervisé pour la prédiction des mots-clefs.

b) Latent Dirichlet Allocation

i. Entraînement de l'algorithme

Pour déterminer des tags de façon non supervisé, nous utilisons l'agolrithme LDA (Latent Dirichlet Allocation) afin de déterminer les topics représentés. Pour déterminer le nombre de topics à chercher, nous pouvons tracer l'évolution de la perplexité, ainsi que du logarithme de la vraisemblance (log likelihood) en fonction du nombre de topics déterminés (*figure 3*).

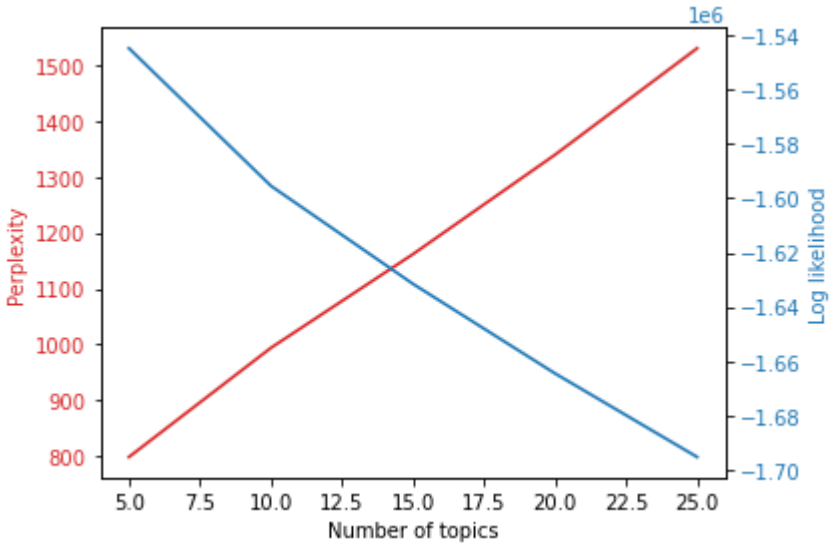

Figure 3: Perplexité et vraisemblance en fonction du nombre de topics pour une LDA

Nous entraînons alors notre LDA avec 15 topics. Puis, nous pouvons afficher les mots les plus présents dans chaque topic (*figure 4*). Pour effectuer notre prédiction, pour chaque texte, notre LDA prédit le topic auquel il semble appartenir, puis nous affichons les 5 mots les plus représentés dans ce topic. Nous obtenons ainsi nos mots-clefs pour chaque texte.

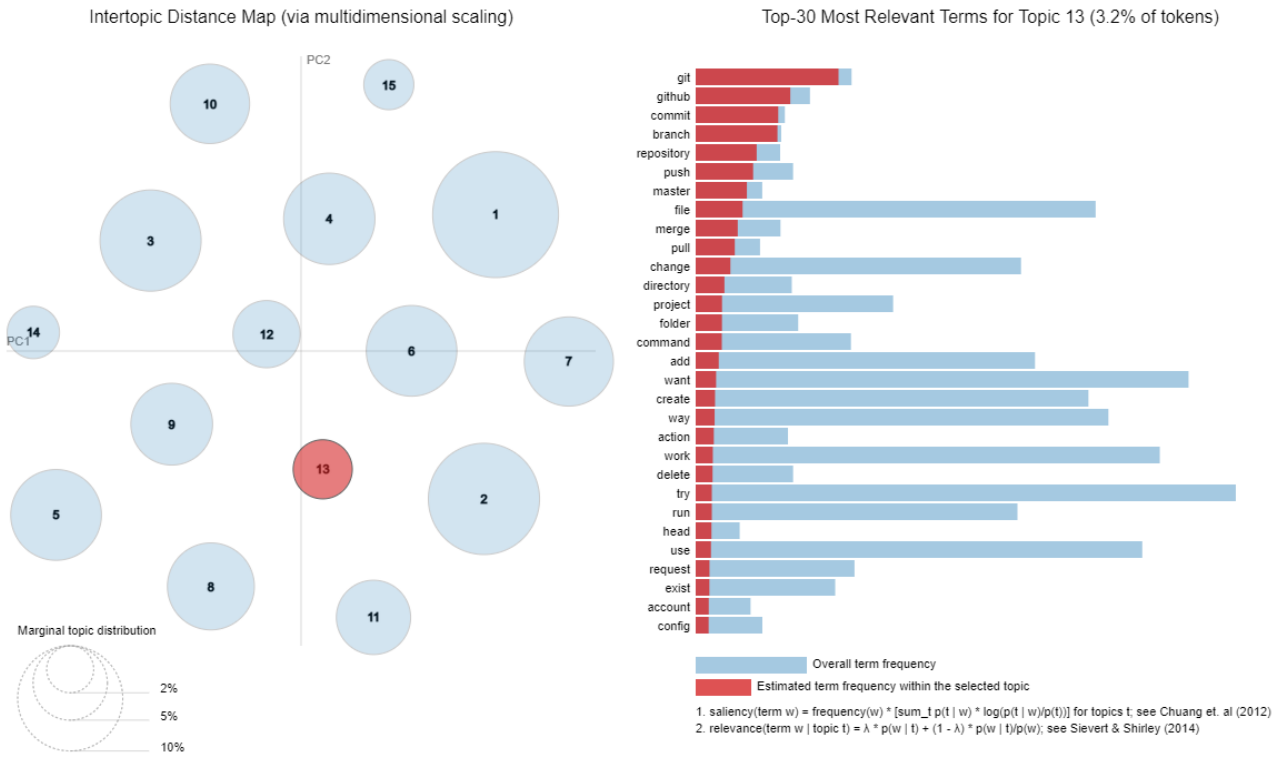

Figure 4: Top 30 des mots les plus présents pour chaque topic

ii. Évaluation de la qualité de l'algorithme

Pour évaluer les tags trouvés par la LDA, nous pouvons comparer ces derniers avec ceux qui seront utilisés pour l'apprentissage supervisé. En procédant ainsi, nous constatons que les scores F1 et Jaccard sont, quasiment, nuls. Cela peut s'expliquer par le fait que nos métriques comparent strictement les tags entre eux.

Or, dans le cadre de la LDA, il serait plus pertinent de comparer l'aspect sémantique des tags. Pour ce faire, nous pourrions utiliser des réseaux de neurones tels que *word2vec* ou *doc2vec* (*gensim*).

Ici, nous allons manuellement chercher le sens de chaque topic, ainsi que celui d'une trentaine de données à tester, puis les comparer (procédé chronophage). Notre but est d'observer une tendance dans nos métriques (*figure 5)*. Avec ces résultats, nous pourrons les comparer avec ceux de notre modèle supervisé.

```
LDA:
F1 score (weighted)      = 0.35
Jaccard score (weighted) = 0.25
```

Figure 5: Scores manuels de F1 et Jaccard pour LDA

2. Apprentissage supervisé

a) Encodage / décodage des tags

Pour encoder les mots-clefs, nous utilisons l'algorithme *MultiLabelBinarizer*. Ce dernier va créer, pour chaque entrée, un vecteur de taille égale au nombre de tags uniques. Il assignera des valeurs positives pour les tags présents, les autres valeurs du vecteur seront à 0.

Lors de la prédiction, la valeur obtenue est une liste de nombres. Ces derniers correspondent aux index des classes de *MultiLabelBinarizer*. Par exemple, si la prédiction retourne les nombres *i* et *j*, alors les i-ème et j-ème tags sont ceux prédits (les classes sont rangées par ordre alphabétique). Ce décodage est effectué par un *LabelEncoder*, entraîné avec les classes de *MultiLabelBinarizer*.

b) Algorithmes TF-IDF et TruncatedSVD

Comme lors de l'apprentissage purement non supervisé, nous utilisons un algorithme de type *TF-IDF*. Afin de réduire le nombre de features, on fait le choix d'utiliser un algorithme *TruncatedSVD*. L'avantage de ce dernier, par rapport à une *ACP*, est qu'il fonctionne bien avec des matrices creuses. Cela nous sera bénéfique pour mettre en place un *pipeline*, utilisant *TF-IDF* et *TruncatedSVD* pour pré-processer les données d'entrée. Ces dernières pourront être renseignées de façon brute dans l'algorithme final.

c) Choix du modèle

Pour des questions de temps d'exécution, nous effectuons les différents entraînements sur un jeu d'entraînement réduit. Il en va de même pour le jeu de test (utilisation d'un jeu réduit).

i. One versus rest

Puisque nous souhaitons prédire plusieurs tags pour une même entrée, nous devons mettre en place une stratégie de prédiction multiclasses. Pour ce faire, nous utilisons l'algorithme *OvR* (One-vs-the-rest). Ce dernier entraîne un modèle de classification binaire pour chaque classe.

Nous utiliserons ensuite la fonction *predict_proba* afin d'obtenir les probabilités d'appartenance à chaque classe pour une entrée donnée. Avec un seuil, nous déterminerons les tags à retenir.

La grande faiblesse de cette stratégie est le temps d'exécution : il faut entraîner, pour chaque modèle, un nombre de modèles de classification binaire égal au nombre de classes (ici 296, soit le nombre de tags uniques).

ii. Modèles testés

Via une recherche sur grille et une validation croisée, nous allons entraîner trois types de modèles :

- un modèle linéaire : *régression logistique*,
- un modèle non linéaire : *SVM* à noyau gaussien
- un modèle ensembliste : *forêt aléatoire*.

Pour ces trois modèles, nous définissons le paramètre *class_weight* à *balanced*. Le but est de pouvoir compenser le déséquilibre des classes (un petit nombre de tags est surreprésenté, e.g. le tag *python*). La métrique *neg_log_loss* est utilisée comme *scoring* pour le jeu de validation.

iii. Comparaison des scores

Pour choisir notre modèle, nous comparons trois métriques : la perte négative logarithmique sur le jeu de validation, la perte logarithmique sur le jeu de test et le temps de prédiction (*figure 6*).

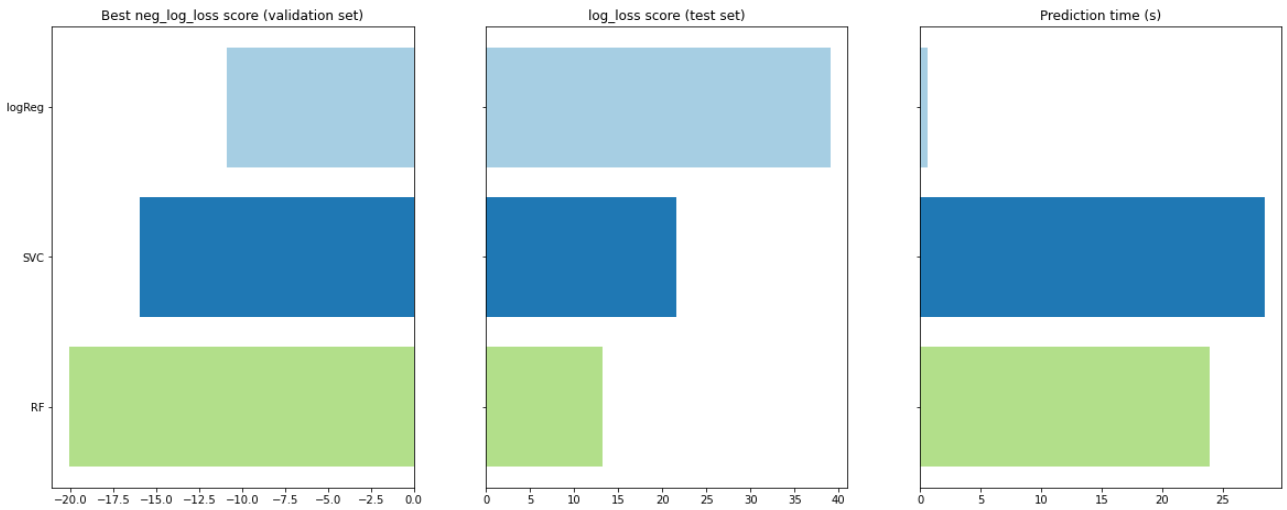

Figure 6: Comparaison des trois modèles supervisés

Durant l'entraînement, le modèle *Régression Logistique* possède le meilleur score. Cependant, il présente de moins bons résultats avec le jeu de test.

Concernant le temps de prédiction, la régression logistique est, de très loin, le modèle le plus rapide. Cette dernière métrique est primordiale, car nous ne souhaitons pas qu'un utilisateur attende une trentaine de secondes pour obtenir ses prédictions. C'est pourquoi, le modèle de régression logistique sera celui utilisé.

d) Évaluation de la qualité du modèle final

Le modèle de *régression logistique* étant retenu, nous effectuons un nouvel entraînement. Cette fois-ci, l'intégralité de tout le dataset est utilisé. De plus, nous mettons en place un *pipeline* appelant les algorithmes *TF-IDF*, *TruncatedSVD*, puis notre modèle (*One vs Rest* & *régression logistique*). Nous pouvons à présent mettre en place de nouvelles métriques pour tester la qualité du modèle final.

Pour le modèle final, nous calculons les scores *F1* (micro) et *Jaccard* (micro) sur un échantillon du jeu de test, en fonction d'un seuil de prédiction (*figure 7*). Ce seuil est nécessaire car, pour rappel, nous utilisons la méthode *predict_proba* pour effectuer les prédictions des mots-clefs.

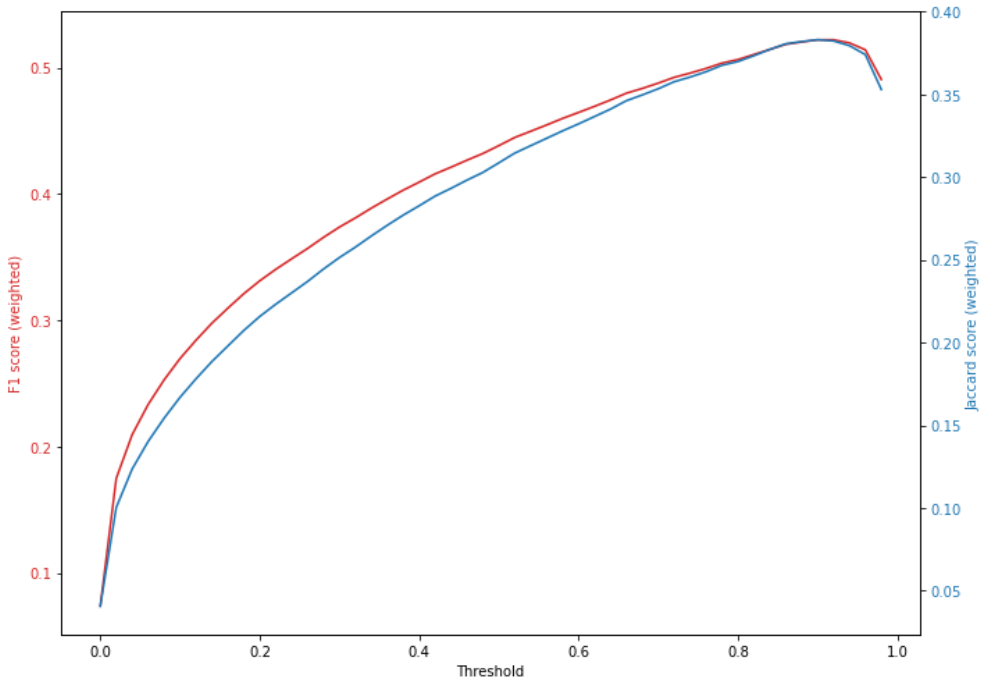

Figure 7: Scores F1 et Jaccard en fonction du seuil de predict_proba

Nous recherchons le seuil qui maximise le score de Jaccard. Puis, avec ce seuil, nous pouvons calculer le score F1 correspondant (*figure 8*). Pour le déploiement du modèle final, nous retiendrons le seuil de 0,90.

En comparant ces scores avec ceux du modèle LDA (*figure 5*), nous constatons que le modèle de régression logistique est meilleur pour la prédiction de tags. De plus, le score F1 tend à confirmer le fait que l'algorithme a été capable de gérer le déséquilibre de la distribution des tags.

```
Threshold =  0.9

Scores (average == weighted):

Jaccard score    = 0.38
F1 score         = 0.52
```

Figure 8: Scores F1 et Jaccard pour algorithme supervisé

Bien que ces scores soient meilleurs que ceux de la LDA, nous constatons qu'ils sont quand même relativement faible (i.e. inférieur à un score moyen de 0,5). Cela met en lumière les limites de la librairie scikit-learn. Pour augmenter la pertinence de notre modèle, il semble obligatoire de se tourner vers d'autre librairie de machine learning pour le traitement des données textuelles.

V. Déploiement API

Nous enregistrons nos différents algorithmes grâce à la librairie *dill*. Celle-ci est une extension du module *pickle*, qui permet de sérialiser et désérialiser des objets python (transformer ces derniers en série d'octets, et inversement).

Pour créer notre API, nous utilisons *Flask* afin d'utiliser nos modèles sérialisés. Le développement de l'application est géré grâce au logiciel de versionning *Git*. Enfin, *Heroku* permet de déployer l'application depuis la branche master du dépôt *Git*. Cette dernière est disponible à l'adresse suivante : https://gp-iml-project5-tags.herokuapp.com/

VI. Conclusion

Au cours de ce projet, nous avons pu découvrir le traitement des données textuelles, grâce à l'utilisation de librairies telles que *NLTK* ou *spaCy*.

Bien que *scikit-learn* semble efficace pour effectuer du prototypage, les modèles utilisés montrent leurs limites lors des prédictions (score *Jaccard* plutôt faible, par exemple).

Pour améliorer les performances de l'API, une piste intéressante est l'exploration des réseaux de neurones. Que cela soit pour effectuer de l'embedding (*Word2vec*, *Doc2vec* de la librairie *gensim*) ou des prédictions (communauté *HuggingFace*, modèle *Bert*, etc.), ces réseaux de neurones semblent être mieux adaptés au traitement des données textuelles.