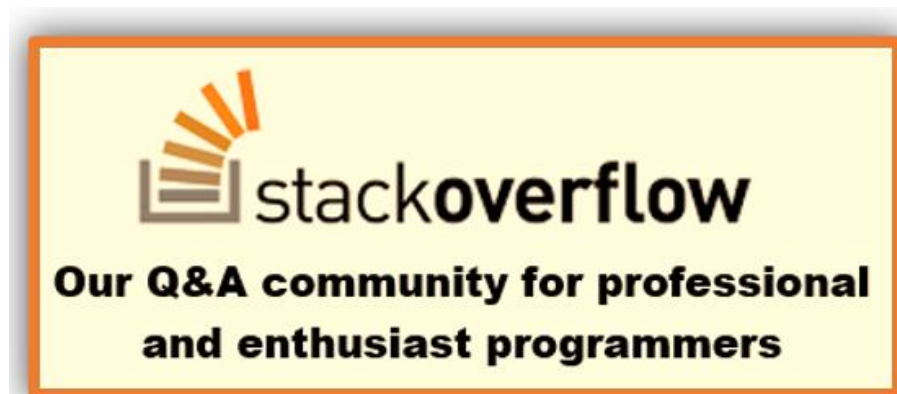




CATEGORISEZ AUTOMATIQUEMENT DES QUESTIONS

PRÉDICTION DE TAGS



Sommaire

I.	Introduction.....	3
1.	Objectif.....	3
2.	Méthode.....	3
II.	Traitement des données.....	3
1.	Présentation des données	3
2.	Préparation des données.....	4
2.2	Séparation du jeu de données	5
2.3	Analyse du corpus de tags	5
2.4	Analyse du corpus de posts	5
2.5	Occurrence Tokens / Tags.....	5
3.	Features engineering	6
3.1	Tags	6
3.2	Bag of words (gensim)	6
3.3	Tf-idf (gensim)	6
3.4	Document embedding : universal sentence encoder (tensorflow)	7
III.	Modélisation.....	7
1.	Baseline	7
2.	Apprentissage non supervise	7
2.1	Topics detection avec lda	8
2.2	Topic detection avec nmf	8
3.	Apprentissage supervise	8
3.1	Régression logistique (scikit learn).....	Error! Bookmark not defined.
3.2	Support Vector Machine (scikit learn)	Error! Bookmark not defined.
3.3	Random Forest (scikit learn)	Error! Bookmark not defined.
3.4	Document embedding + réseau de neurones (tensorflow).....	Error! Bookmark not defined.

4. Apprentissage semi-supervisé	8
4.1 Topics detection suivi d'une classification supervisée	Error! Bookmark not defined.
V. Présentation des résultats	8
1. Métrique.....	8
2. Synthèse	9
3. Choix du meilleur modèle	10
VII. Déploiement du modèle	12
IX. Conclusion	13

I. Introduction

Stack Overflow est un site célèbre de questions-réponses liées au développement informatique. Chaque question est constituée principalement de trois parties : le titre de la question, l'intitulé de la question et des tags. Les tags permettent de catégoriser la question afin d'améliorer sa visibilité.

1. Objectif

L'objectif de notre projet est de développer un système de suggestion de tag afin d'aider les amateurs du site à trouver les tags appropriés associés à leur question. Pour cela nous allons construire un modèle de prédiction de tags performant à partir des questions posées et qui ont déjà des tags.

2. Méthode

Les données que nous avons nécessitent une première phase de traitement de données. A l'issue de cette phase, nous aurons la modélisation. Elle nous permettra de tester plusieurs modèles issus de des apprentissages supervisé, non supervisé et semi-supervisé. Cette étape nous permettra de déterminer le meilleur modèle de prédiction associé à notre jeu de données. La dernière étape sera de déployer notre modèle sur la plate-forme « PaaS » Heroku.

II. Traitement des données

Le traitement des données est une phase impérative dans la modélisation prédictive. En effet, les données mises à notre disposition comportent des irrégularités et doivent être traitées avant de pouvoir être utilisées par les algorithmes de prédiction. Dans cette partie, nous présenterons les données utilisées dans notre projet, puis nous préparerons les données pour la modélisation et enfin nous appliquerons le feature engineering à notre base de données afin d'adapter nos données aux différents algorithmes.

1. Présentation des données

Stack Overflow propose un outil d'export de données « [stackexchange explorer](#) », qui recense un grand nombre de données authentiques de la plateforme d'entraide. Les données seront importées grâce à des requêtes SQL.

```
SELECT FROM posts WHERE Id  
BETWEEN value1 AND value2
```

Figure 1: Syntaxe SQL

Par défaut, il y a une limite sur le temps d'exécution de chaque requête SQL, ce qui peut rendre difficile la récupération de toutes les données d'un coup. Pour récupérer plus de résultats, nous avons fait des requêtes avec des contraintes sur les « **id** » :

Nous avons effectué 21 requêtes en changeant à chaque fois les valeurs des paramètres « **value1** » et « **value2** » dans la syntaxe ci-dessus (**fig.1**). Au total 21 fichiers csv ont été téléchargés de « [stackexchange explorer](#) ». Chaque fichier comporte 23 variables et un nombre de postes moyen de 15 280. Après avoir concaténer les 21 fichiers csv et sélectionner les variables d'intérêt, nous avons obtenu 552 487 questions et 3 variables d'intérêt que sont : « **Body** », « **Title** » et « **Tags** ». La variable « **Title** » capture le titre de la question, « **Body** » capture l'intitulé de la question et « **Tags** » l'ensemble des tags associé à la question.

2. Préparation des données

Nous allons tirer un échantillon aléatoire de 200 000 questions sur les 552 487 questions afin de réduire le temps d'exécution des algorithmes. La suite de notre projet va se baser sur l'échantillon ainsi obtenu.

2.1 Traitement des intitulés et titres des questions

Dans un premier temps, nous allons concaténer le titre et l'intitulé de la question dans une colonne de notre dataframe appelée « **Texte** ». La nouvelle colonne crée comporte des tags HTML ainsi que des caractères non pertinents que nous allons nettoyer à l'aide de la librairie **bs4** de python. Après cette étape, nous allons supprimer les **stopwords** dans la colonne « **Texte** » et transformer le texte de chaque question en une liste constituée de ses mots. Cette étape s'appelle la « **tokenization** ». Afin de prendre en compte les expressions c'est-à-dire les groupes de mots qui n'ont de sens que une fois employés ensemble, nous allons procéder à la construction bigrams/trigrams. Et pour finir, nous allons appliquer à la colonne « **Texte** » la **Lemmatization** dans le but de réduire les mots en leur expression élémentaire.

A l'issue du traitement des intitulés et des titres des questions, notre dataset comporte une nouvelle colonne « **Texte** » constituée de listes de mots et expressions essentiels de chaque post.

2.2 Séparation du jeu de données

Après la phase précédente, nous allons scinder notre dataset en jeux de données d'apprentissage (**training set**), de validation (**validation set**) et de test (**testing set**).

- Le training set comporte 119 997 questions (soit 60% du dataset) et 2 colonnes (« **Texte** » et « **Tags** »)
- Le validation set ainsi que le testing set comporte chacun 39999 questions (soit 20% du dataset) et 2 colonnes (« **Texte** » et « **Tags** »)

Nous utiliserons le training set pour l'analyse du corpus et l'entraînement des modèles de prédiction. Le validation set sera utilisé pour déterminer le meilleur modèle de prédiction et enfin, le testing set pour évaluer le meilleur modèle.

2.3 Analyse du corpus des tags

Le corpus des tags comporte 13 866 tags différents. Afin de réaliser un algorithme de prédiction efficace, le corpus des tags sera réduit arbitrairement à 100 tags. Les questions sans tags dans les trois jeux de données seront exclues de notre étude.

2.4 Analyse du corpus des posts

Le corpus des posts comporte 238 700 tokens différents. Nous allons procéder à deux types d'analyse sur le corpus des posts. D'une part, nous allons sélectionner les tokens porteurs de sens et éliminer les autres. Cela nous donnera 4187 tokens. D'autre part, nous chercherons à conserver les relations entre les tokens composant le post. Nous obtenons alors 7862 tokens. Le premier type d'analyse nous donnera un corpus qui nous sera utile pour l'apprentissage supervisé tandis que le second sera destiné à l'apprentissage non supervisé.

2.5 Occurrence Tokens / Tags

Le lien entre un tag et un token est représenté par le nombre d'occurrences croisées entre ce tag et ce token dans l'ensemble du corpus. L'analyse de l'occurrence tokens/tags nous

donne une matrice de taille (m, t) où m est le nombre de tokens et t celui des tags. La valeur se trouvant à la i ème ligne et la j ème colonne de la matrix représente le nombre de questions de notre dataset où on retrouve à la fois le token i et le tag j . Cette matrice est utile pour explorer les liens tokens-tags, pour formuler des hypothèses, et éventuellement pour réaliser des prédictions.

3. Features engineering

Il s'agira de trouver une représentation numérique des variables « **Texte** » et « **Tags** » de notre dataset.

3.1 Variable Tags

Nous utiliserons la fonction **MultiLabelBinarizer** du module **preprocessing** de **sklearn** pour transformer la variable « **Tags** » en un **ndarray** de dimension (m, k) où m est le nombre de question et k le nombre de tags. La valeur se trouvant à la i ème ligne et la j ème colonne du **ndarray** est 1 si le tag j est un tag de la question i , 0 sinon.

3.2 Variable Texte

Nous utiliserons trois (03) méthodes différentes pour obtenir une représentation numérique de la variable « **Texte** » : **Bag of words**, **Tf-idf** et **Universal Sentence Encoder**.

➤ Bag of words

Nous utiliserons la fonction **CountVectorizer** du module **feature_extraction** de **sklearn** pour transformer la variable « **Texte** » en un **ndarray** de dimension (m, t) où m est le nombre de question 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** représente le nombre de fois le token j est présent dans la question i .

➤ Tf-idf (gensim)

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 définit par:

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

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 et **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.

➤ Universal sentence encoder (tensorflow)

Nous utilisons ici deux variantes de l'algorithme « universal sentence encoder » :

- Universal sentence encoder à architecture Deep Averaging Network (DAN)
- Universal sentence encoder à architecture Transformer

Ces algorithmes transforment le texte brut des questions en un **ndarray** de dimension **(m, 512)** où **m** est le nombre de questions.

III. Modélisation

1. Baseline

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

2. Apprentissage non supervise

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)**.

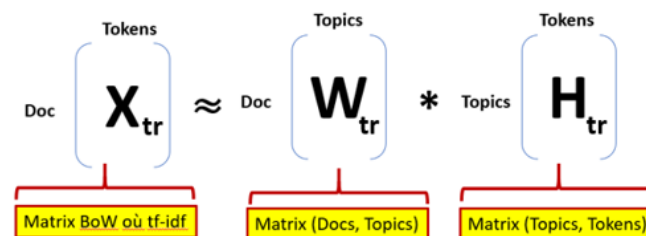


Figure 2: Décomposition de la matrix BoW/tf-idf

Le principe général de ces deux (02) algorithmes est de décomposer la matrice X bag of words ou tf-idf en deux matrices W et H .

2.1 Topics détection avec 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.

2.2 Topics détection avec nmf

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

3. Apprentissage supervise

Différents modèles, linéaires (**régression logistique** et **SVM**) et non linéaires (**random forest** et **réseau de neurones**) ont été entraînés sur les différents train sets. 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. Nous utiliserons le module **tensorflow** de python pour le réseau de neurones.

4. Apprentissage semi-supervisé

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 en entrée des algorithmes de classification supervisés.

IV. Présentation des résultats

1. Métriques

Les métriques nous permettent de mesurer la performance des modèles et de comparer ainsi des modèles entre eux.

- Métrique pour apprentissage non supervisé

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**.

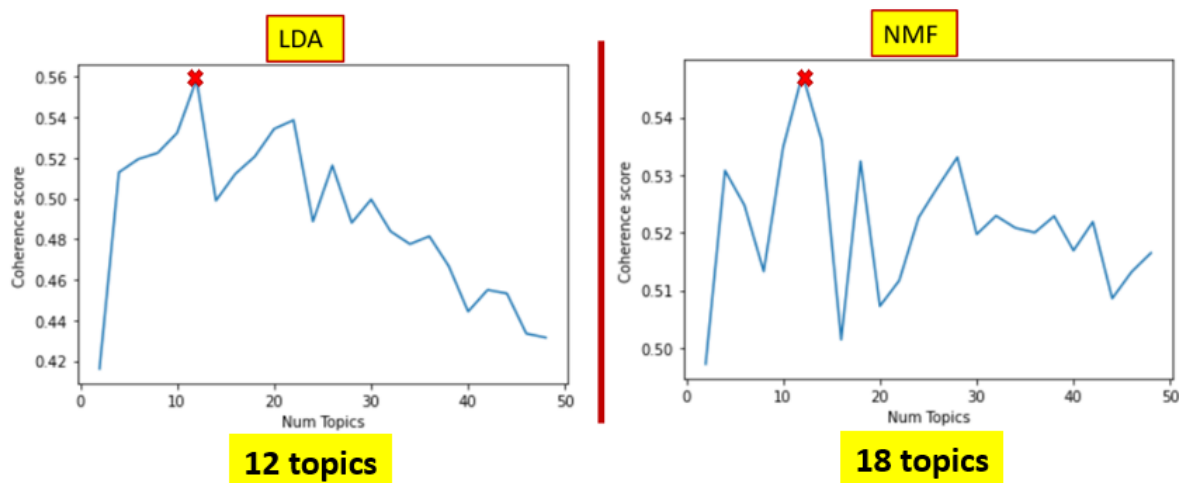


Figure 3: Score de Cohérence lda & nmf

Le nombre de topics qui donne la plus grande valeur du score de cohérence sera retenue pour la modélisation.

- Métrique pour apprentissage supervisé et semi-supervisé

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 la F1 score macro. Elle calcule la moyenne des F1 scores calculée sur chaque tag.

2. Synthèse

Nous avons réalisé au total 16 modèles de prédiction dont les performances ont été représentées dans la figure ci-dessous (*fig4*).

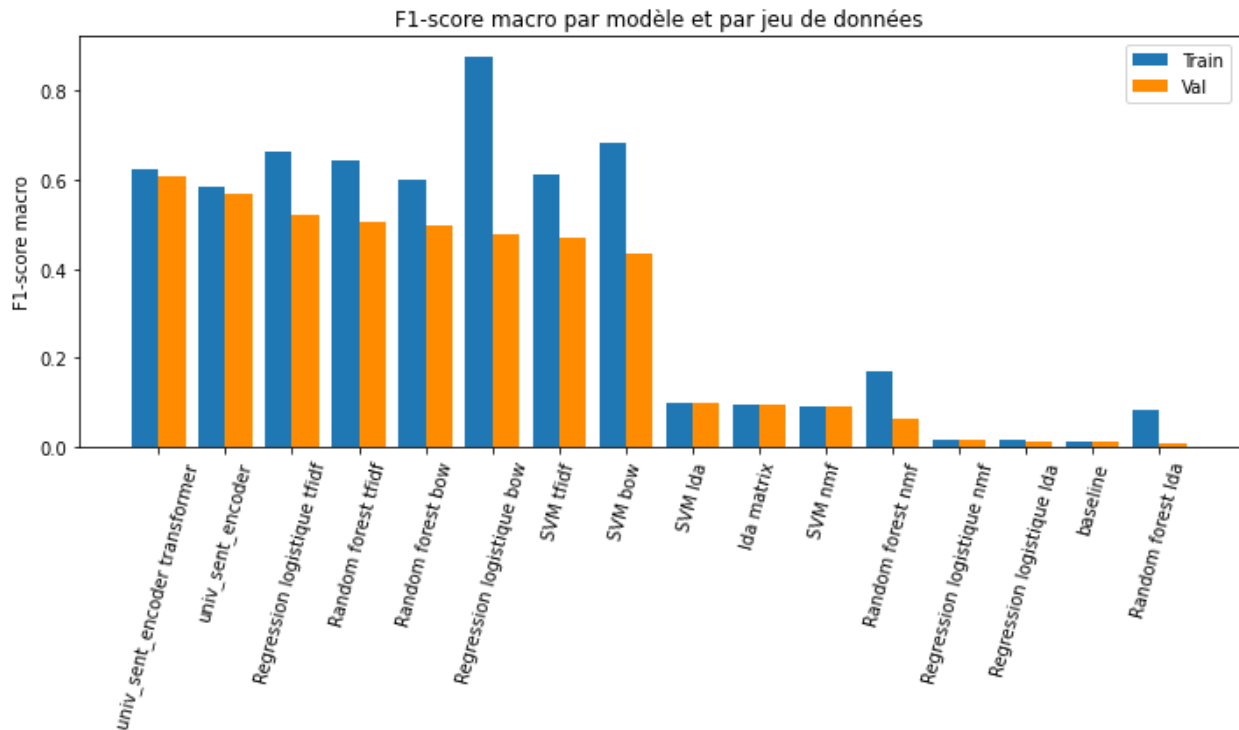


Figure 4: Comparaison des performances des modèles d'apprentissage supervisés

Nous pouvons regrouper les modèles en 4 groupes suivant leur performance :

- Les modèles à très faibles performance : **Régression logistique nmf**, la **régression logistique lda**, le modèle **baseline** et le modèle **random forest lda**
- Les modèles à faible performance : **SVM lda**, **lda matrix**, **SVM nmf** et le **random forest nmf**.
- Les overfitting modèles : Ce sont des modèles qui font une assez bonne performance mais qui sur-apprend. On retrouve la **régression logistique tfidf**, le **random forest tfidf**, le **random forest bow**, la **régression logistique bow**, le **SVM tfidf** et le **SVM bow**.
- Les modèles performants : **Universal Sentence Encoder DAN** et **Universal Sentence Encoder transformer**

3. Choix du meilleur modèle

D'après la figure 4, le meilleur modèle associé à nos données est l'universal sentence encoder transformer. Cependant compte tenu des contraintes de temps d'exécution,

nous avons choisi de considérer l'universal sentence encoder DAN comme le meilleur modèle.

D'après la figure 5 ci-dessous, le modèle DAN ne sur-apprend pas et a une performance stable sur le jeu de données test. Nous pouvons dire que le modèle généralise bien.

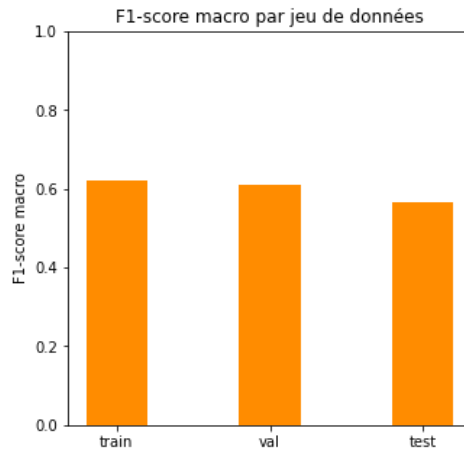


Figure 5: Performance du modèle DAN

L'analyse du F1 score macro par tags nous montre que le modèle n'a pas une performance uniforme sur l'ensemble des tags (**fig**).

F1 score macro pour l'ensemble des tag (classé par ordre décroissant)

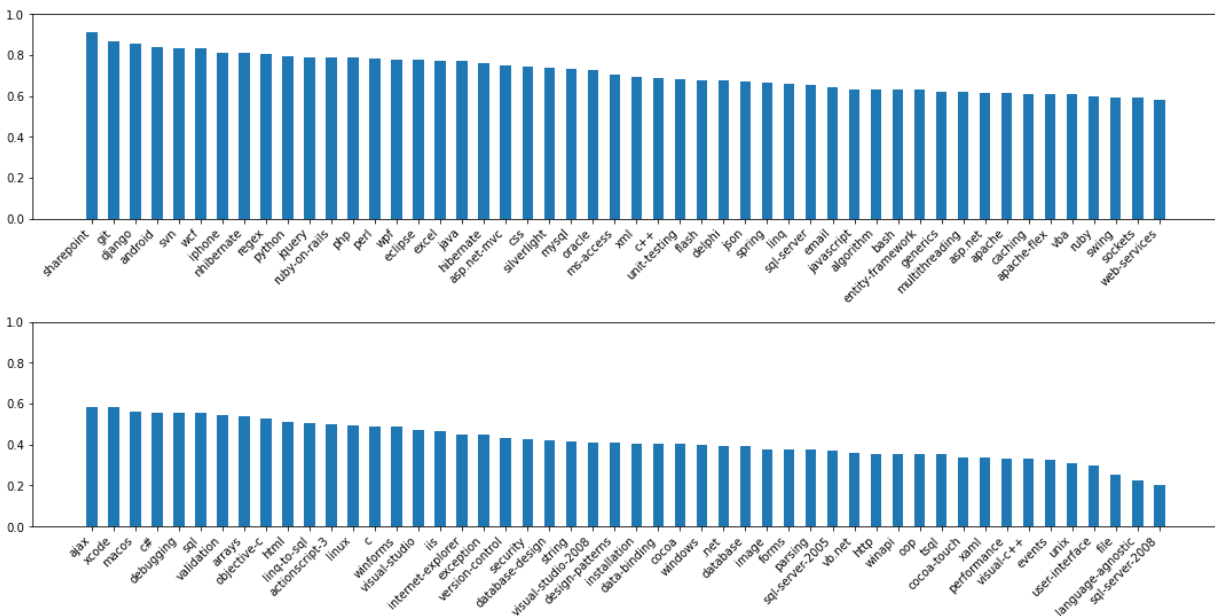


Figure 6: F1 score macro pour l'ensemble des tags

La figure 7 ci-dessous montre que pour certains tags, nous avons un problème de classification déséquilibrée. D'où la faible performance du modèle sur certains tags.

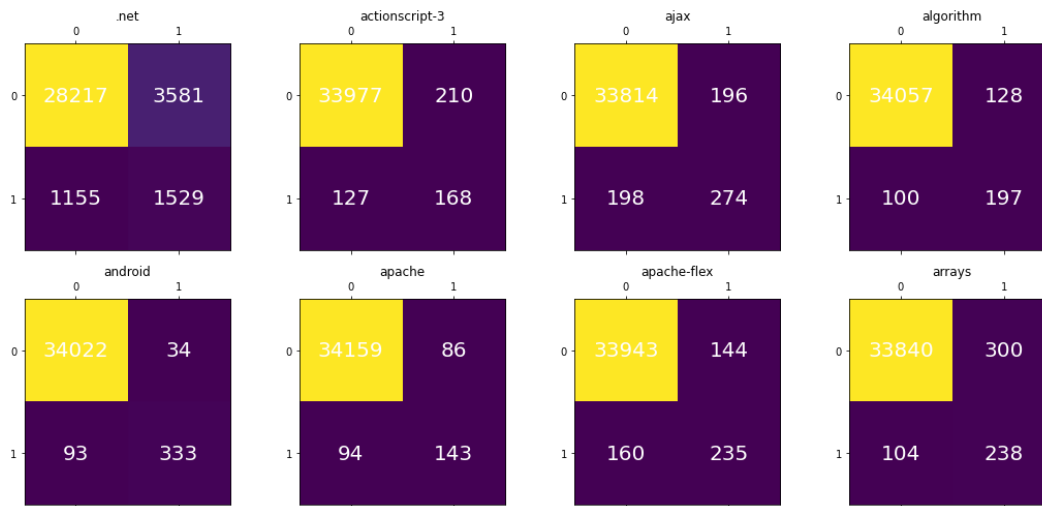


Figure 7: Matrices de confusion

V. Déploiement du modèle

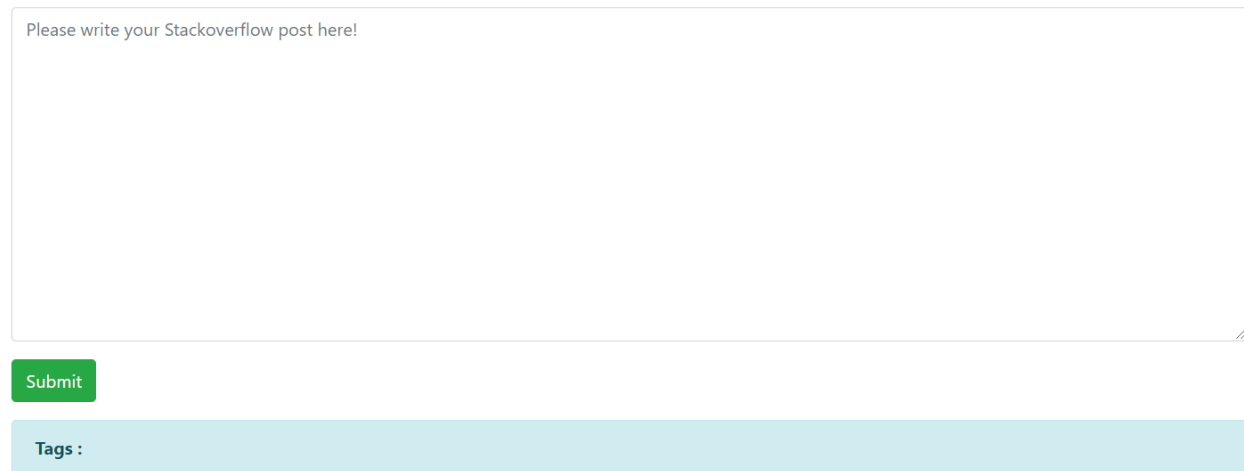
Nous allons déployer notre modèle de prédiction en ligne afin qu'il soit accessible à tous. Pour ce faire, nous choisirons Heroku comme plate-forme as a service (PaaS) et Flask comme micro web framework.

Comme tenue de la limite de mémoire allouée sur Heroku, nous avons déployé la régression logistique tfidf. Le lien associé au déploiement est :

<https://stackoverflow-tags-prediction.herokuapp.com/>

Stackoverflow Tags prediction

By Didier ILBOUDO



Please write your Stackoverflow post here!

Submit

Tags :

Figure 8: Page d'accueil du modèle sur Heroku

VI. Conclusion

Les amateurs de Stack Overflow ont du mal à trouver les tags appropriés à leurs questions. Afin de venir en aide à la communauté, nous avons alors décidé de développer un modèle de prédiction de tags en fonction des questions. Nous avons à faire à un problème de classification multi-labels. Après avoir implémenté des modèles d'apprentissage non-supervisé, supervisé et semi-supervisé, nous avons identifié l'universal sentence encoder transformer comme étant le meilleur modèle associé à notre jeu de données. Cependant, ce modèle est assez lourd en termes de ressources informatiques nécessaires lors de l'entraînement et lors de l'inférence. Nous avons retenu comme meilleur modèle pour notre étude, le modèle DAN qui est plus léger. Pour la mise en production de notre modèle de prédiction des tags, nous avons préféré déployer la régression logistique sur features tfidf qui est beaucoup plus légère et sans forte baisse de performance.

Ce dernier modèle a été déployé sur Heroku et est maintenant accessible sur <https://stackoverflow-tags-prediction.herokuapp.com/>.

Différentes pistes peuvent être envisagées pour améliorer notre projet :

- ❖ Séparation stratifiée des jeux de données en vue d'avoir un échantillon représentatif (« **train** », « **val** », « **set** ») ;

- ❖ Prise en compte de la corrélation entre les différents tags lors de l'apprentissage (par exemple le tag « **pandas** » est certainement associé au tag « **python** ») ;
- ❖ Évaluation d'autres méthodes de features engineering (« **word embedding** », « **POS** »), et d'autres modèles ;
- ❖ Prédire plus de tags ou sélectionner manuellement un nombre limité de tags à prédire (remplacer « **python-3.x** » et « **python-2.7** » par « **python** »).
- ❖ Résoudre le problème de « **classification déséquilibrée** »