https://github.com/fedecabre/P5_StackOverflow/

Projet 5 Ingénieur Machine learning

Catégoriser automatiquement des questions

Federico CABRERA

juin 2021

Federico CABRERA
juin 2021

Problematique

Sur le site de questions réponses liées au développement informatique **StackOverFlow** il est important d'utiliser des *Tags* de manière à classer les questions et pouvoir retrouver facilement les questions qui sont similaires.
Afin d'aider à l'ajout de ces Tags des suggestions peuvent être très utiles en particulier pour les utilisateurs débutants.
L'objectif de ce travail est donc de développer **un modèle de suggestion de *Tags*** sous forme d'un algorithme de machine learning qui assigne plusieurs tags de manière automatique.
Le modèle choisi sera disponible via une **API** avec un point d'entrée pour une suggestion de tags pour des nouveaux posts.
Pour suivre les modifications du code, nous allons utiliser un logiciel de gestion de versions Git et tout le code sera disponible sur **GIthub**.

Les données

En utilisant le "stackexchange explorer", nous avons pu obtenir un grand nombre de données authentiques de la plateforme.
Nous avons choisi pour cette étude de nous focaliser sur les post du **langage "Python"**, afin d'avoir des posts de qualité nous avons ajouté un filtre sur le score des posts pour avoir un Score > à 50. Avec ce filtre nous obtenons plus de 10 000 posts avec leur tags correspondant nous allons vérifier par la suite si ce nombre est suffisant pour entraîner le modèle le valider et tester ses performances. Pour que notre modèle sache reconnaître les post python, vu que tous les post importés ont le tag "Python", nous avons aussi importé les post avec le Tag "R" deux fichiers ont été récupérés pour avoir une diversité dans les tags.

Requête pour le tags python avec score > 50

```
select
    q.Id,
    q.Title,
    q.Body,
    q.Tags,
    q.Score,
    q.Acceptedanswerid,
    a.Body as Answerbody
from posts q
inner join posts a
  on a.Id = q.Acceptedanswerid
where
  q.posttypeid = 1
  and q.Score >= 50
  and q.Tags like ('%<python>%')
```

Federico CABRERA
juin 2021

Pré traitement

Un premier notebook pour mettre en oeuvre des approches non supervisées pour prétraiter les données issues de l'outil d'export de données de Stack Overflow. Nous allons faire un traitement classique du texte avec les étapes suivantes :

Étapes de prétraitement du texte (hors Tags)

1) Suppression du bruit*

1. Suppression du code HTML avec **BeautifulSoup**
2. Suppression des contractions avec **contractions**
3. Mettre en minuscule le texte
   * La correction orthographique n'a pas été jugé nécessaire pour cette étude mais nous avons évalué la librairie **autocorrect**

2) Suppression des caractères

1. Suppression de la ponctuation, des caractères spéciaux et des nombres avec **regex**
2. Suppression de la présence d'un seul caractère avec **regex**

3) Suppression de StopWords

1. Suppression du mot le plus fréquent avec **nltk**
2. Suppression des adjectives avec la tag list de **nltk**

4) Stemming / Lemmatization*

1. Lemmatization avec **nltk**
   * Le stemming n'a pas été appliqué car la lemmatisation a été jugée suffisante

5) Export des données pour entrainement des modèles

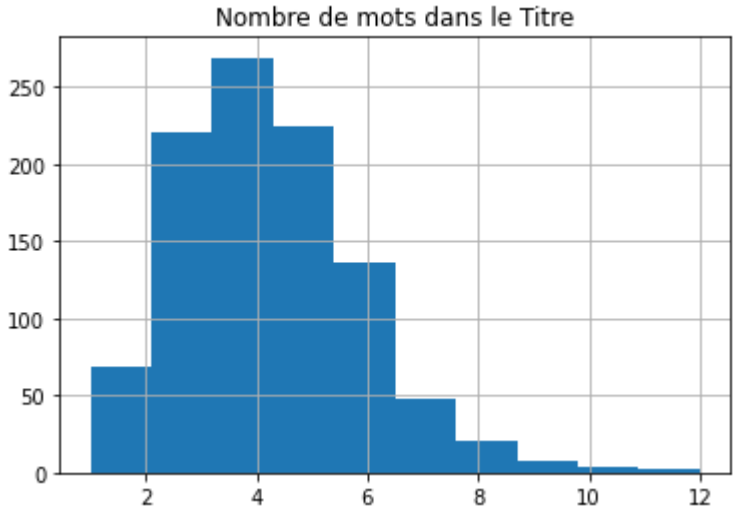
Ce pré traitement sera utilisé pour effectuer des détection de sujet (LDA, NMF) et plusieurs modèles de classification supervisés (kNN, RandomForest, MLP), des informations nécessaires à certaines analyses peuvent être perdues.
Les tags ont été traités différemment avec seulement une suppression du code html et nous avons gardé les tags avec une seule lettre pour ne pas perdre les tags "r".

Méthodes non-supervisés

Moteur de recherche avec embedding pre-entraînés GPT-2 et PyTorch

En utilisant une représentation type GPT-2 dense dans laquelle le mot possède une représentation dans un espace qui le positionne en fonction des mots adjacents nous avons créé un **moteur de recherche** qui parcourt la collection de questions StackOverFlow importée et récupère les questions et les tags les plus similaires à la question d'entrée. Nous avons trouvé les recherches sur les titres plus pertinentes que sur tout le texte car l'information qui contient le titre est déjà bien condensée et le moteur de recherche avec l'embedding "généraliste" a moins de difficultés à trouver des titres similaires.

Une fois que nous avons créé la représentation avec l'embedding pour chaque titre. Nous pouvons maintenant entrer une question et rechercher parmi les titres de question StackOverflow lesquels sont les plus similaires à l'entrée en utilisant la similitude cosinus entre l'embedding de la question d'entrée et les titres.



Le nombre de mots dans le titre se trouve autour de 4 mots après le pré-traitement décrit précédemment.

Ci-dessous un exemple de propositions de tags pour la question "How to install Pandas"

```
getMostSimilarQuestions(5, "How to install Pandas", df_questions ,QIDList, df_tags_list)
```

```
Most similar  5  questions of :  How to install Pandas

1 Question Id :  13611065
 Question :  Efficient way to apply multiple filters to pandas DataFrame or Series
Tags suggérés :  python , algorithm , pandas

2 Question Id :  17141558
 Question :  How to sort a dataFrame in python pandas by two or more columns?
Tags suggérés :  python , pandas , python-2.7 , sorting , data-analysis

3 Question Id :  12439588
 Question :  How to maximize a plt.show() window using Python
Tags suggérés :  python , matplotlib

4 Question Id :  16852911
 Question :  How do I convert strings in a Pandas data frame to a 'date' data type?
Tags suggérés :  python , date , pandas

5 Question Id :  13999850
 Question :  How to specify date format when using pandas.to_csv?
Tags suggérés :  python , pandas , export-to-csv , datetime-format , date-formatting
```

Modélisation automatique des sujets - Extraction des tags du texte

L'objectif de ce type de modélisation de sujets est de récupérer de potentielles catégories pour des traitements ultérieurs. Cette modélisation offre surtout une meilleure compréhension de la structuration du texte en vue de création de features manuelles (mettre l'accent sur certains mots, comprendre ce qui définit une catégorie, etc.).
Nous avons transformé nos données pré-traités qui étaient dans une forme non structurée dans des features utilisables et structurés à l'aide de l'outil **CountVectorizer** la bibliothèque **skylearn**.
Les données sont transformés en vecteurs du type "**Bag of Words"** et un **dictionnaire** est créé avec l'ensemble de mots présents dans les textes. il est aussi possible d'utiliser le même outil pour générer de **n-grams** de mots pour garder le sens du texte qui serait perdu dans le cas d'utiliser seulement des mots individuels
Deux méthodes non supervisés ont été étudiés et pour une meilleure visualisation nous avons projeté le texte vectorisé dans un espace à deux dimensions à l'aide d'une transformation **T SNE** qui nous permet de comparer visuellement les deux modèles

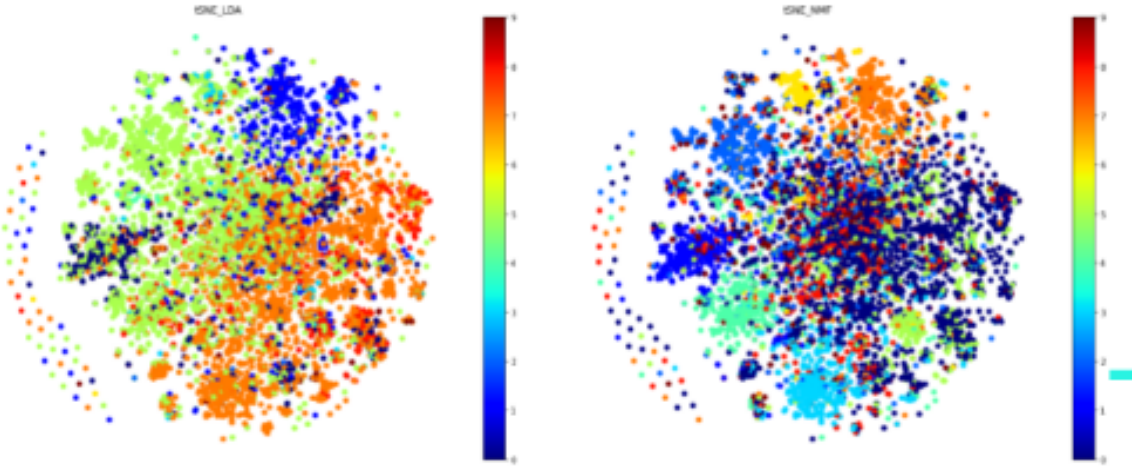
1) LDA, Latent Dirichlet Allocation

L'objectif de ce type de modélisation de sujets est de récupérer de potentielles catégories pour des traitements ultérieurs. Cette modélisation offre surtout une meilleure compréhension de la structuration du texte en vue de création de features manuelles (mettre l'accent sur certains mots, comprendre ce qui définit une catégorie, etc.)

Les meilleurs paramètres pour ce modèle ont été alpha = 0.005 et betha = 0.005, celà signifie que chaque document possède peu de topiques à la fois et que chaque topique est composé de peu de mots.

2) NMF, Non-Negative Matrix Factorisation

Ce modèle trouve les composantes principales du texte tout en gardant des composants positifs pour qu'elles restent compréhensibles. Les meilleurs topics ont été obtenus avec une transformation **tf-idf** avec une régularisation avec alpha = 0.1 et l1_ratio = 0.5 soit une combinaison entre L1 et L2



Représentation dans des topiques de chaque modèle dans la projection en 2 dimensions tsne. On constate que les topics LDA-0 et NMF-1, LDA-1 et NMF-7 ont des sujets similaires.

LISTE DES TOPICS LDA	LISTE DES TOPICS NMF/TF IDF
Thèmes similaires	
TOPIC 0 : ['LIST', 'PRINT', 'IMPORT', 'ELEMENT']	TOPIC 1 : ['LIST', 'ELEMENT', 'ITEM', 'COMPREHENSION']
TOPIC 1 : ['PLOT', 'IMAGE', 'GGPLOT', 'LABEL']	TOPIC 7 : ['PLOT', 'GGPLOT', 'LABEL', 'COLOR']
TOPIC 3 : ['ARRAY', 'NONE', 'INDEX', 'IMPORT']	TOPIC 6 : ['ARRAY', 'INDEX', 'ELEMENT', 'NUMPY']
TOPIC 5 : ['STRING', 'COLUMN', 'NUMBER', 'ROW']	TOPIC 2 : ['COLUMN', 'DATA', 'ROW', 'FRAME']
	TOPIC 4 : ['STRING', 'CHARACTER', 'CONVERT', 'PYTHON']
TOPIC 7 : ['CLASS', 'DEF', 'RUN', 'CALL']	TOPIC 3 : ['CLASS', 'DEF', 'METHOD', 'INITSELF']
	TOPIC 0 : ['PYTHON', 'USE', 'RUN', 'CODE']
TOPIC 8 : ['ERROR', 'LINE', 'PACKAGE', 'MODULE']	TOPIC 5 : ['FILE', 'LINE', 'READ', 'IMPORT']
Thèmes non similaires	
TOPIC 2 : ['LETTER', 'OPERATOR', 'AGE', 'CHR']	TOPIC 8 : ['FUNCTION', 'RETURN', 'DEF', 'ARGUMENT']
TOPIC 4 : ['DATE', 'TIME', 'FORMAT', 'DAY']	TOPIC 9 : ['VALUE', 'KEY', 'RETURN', 'NA']
TOPIC 6 : ['DOG', 'TIMEOUT', 'CAT', 'ELAPSE']	
TOPIC 9 : ['LINE', 'MODEL', 'TASK', 'PROCESS']	

Méthodes Supervisées

Méthode

Vectorisation

Les Post et les Labels ont été vectorisés séparément avec CountVectorizer() avec des paramètres différents. Les posts ont été vectorisés en bag-of-words et nous avons gardé le comptage des mots mais pour les tags nous nous sommes contentés avec un bag-of-word binaire. Cela nous permet d'encadrer notre problème à un **MultiLabel** car un post peut avoir plusieurs labels, mais il n'est pas nécessaire d'avoir le même label plus d'une fois. Nous avons aussi limité le nombre de labels à utiliser car la plupart de labels n'ont pas de post suffisants pour pouvoir entraîner correctement nos modèles.Nous avons séparé nos données en un jeu de données pour l'entraînement des modèles, un jeu de données pour l'évaluation et un dernier jeu pour la validation des performances de chaque modèle..

Optimisation

Tous les modèles ont été entraînés sur plusieurs paramètres avec une grid search avec une cross validation de 5 folds pour les modèles sklearn et évalués avec le set de validation pour les modèles keras-tf, jusqu'à obtenir les meilleurs hyperparamètres dont les performances ont été évalués avec le test set.

Export

Le meilleur modèle, et les transformateurs des post et tags sont enregistrés pour leur utilisation dans l'API. Celle-ci peut recevoir des requêtes qu'elle transforme en vecteurs, fait une prédiction avec le modèle enregistré et transforme la prédiction en tags qu'il envoi en forme de réponse à la requête.

1) Modèle KNN

Nous avons utilisé les modèles de Scikit Learn qui sont compatibles avec les problèmes multi label dont le modèle KNN, le randomForest et le MLPC (multilayer perceptron clasifier). Nous avons entraîné notre modèle KNN avec tous les labels soit 1837 tags différents et un k=5 en privilégiant les plus proches voisin avec une distance plus proche et validé avec une cross validation avec 5 fold

Résultats sur test set après Cross validation avec tous les labels

f1-score:	0.42
precision_score:	0.76
recall_score:	0.29

Le résultats n'étant pas très satisfaisants, nous avons entraîné d'autres modèles avec un nombre limité de tags (2,4,8,16,32) et une gridsearch pour définir le meilleur hyper paramètre K pour notre analyse.
Nous avons constaté que le f1-score augmentait avec la diminution du K atteignant jusqu'à 0.77 lorsque seulement deux Tags (python et r) étaient utilisés pour entraîner le modèle.

Résultats sur test set après gridsearch avec CV avec 32 labels et K=6

f1-score:	0.62
precision_score:	0.80
recall_score:	0.50

Nous avons décidé de garder 32 features pour les tags afin de comparer les modèles supervisés, la précision du modèle ne semble pas être mauvaise (peu de faux positif) mais le recall est encore très bas, notre modèle arrive à détecter seulement la moitié de labels.
32 Tags les plus utilisés dans le dataframe
['class', 'csv', 'data.table', 'dataframe', 'date', 'datetime', 'dictionary', 'django', 'dplyr', 'flask', 'ggplot2', 'json', 'list', 'matplotlib', 'numpy', 'pandas', 'performance', 'pip', 'plot', 'python', 'python-2.7', 'python-3.x', 'r', 'r-faq', 'regex', 'scipy', 'sorting', 'sqlalchemy', 'statistics', 'string', 'unicode', 'windows']

2) Modèle RandomForest

Nous avons procédé de la même manière que précédemment avec un gridsearch avec CV de 5 folds et nous avons optimisé le modèle sur deux paramètres max_depth et n_estimators. Les résultats ne se sont pas améliorés avec ce modèle, le modèle KNN reste notre référence.

Résultats sur test set après gridsearch avec CV avec 32 labels et K=6

f1-score:	0.57
precision_score:	0.72
recall_score:	0.46

3) Modèle Multi Layer Perceptron Classifier SciKit Learn

Le dernier modèle SciKitLearn compatible avec notre problème de classification multi label est le réseau de neurones ou MLPC que nous avons aussi entraîné avec une optimization sur plusieurs paramètre dont alpha qui régularise notre modèle pour éviter un overfit sur le train set le learning rate pour nous assurer que l'entraînement se fasse correctement et nous avons aussi testé plusieurs architectures du réseau de neurones : une couche avec 30 ou 100 unité ou deux couches avec 30 et 100 unités
Résultats sur test set après random search avec CV avec 32 labels et alpha=0.01, learning_rate_init=0.01 et une couche caché avec 100 éléments

f1-score:	0.78
precision_score:	0.87
recall_score:	0.72

Les résultats obtenus étant sensiblement meilleurs que le modèle KNN

4) Modèle Multi Layer Perceptron Classifier KERAS/TensorFlow

Vu les bons résultats du modèle précédent nous avons utilisé le framework KERAS/TensorFlow qui nous permet d'entraîner les modèles plus rapidement avec une carte GPU, l'entraînement du modèle scikit learn prend plusieurs heures à chaque fois vs qq secondes avec TF.

Nous avons, par ailleurs, constaté que nous obtenons un boost de performance en utilisant le `texte brut` sans prétraitement autre que le transformateur keras qui garde tous les mots et enlève les numéros et symboles. Nous sommes passés d'un f1 score de 80.6 % à un f1 Score de **82 %,** nous allons garder par la suite de l'étude le texte sans prétraitement des posts du premier notebook.

Nous créons un tokenizer, configuré pour ne prendre en compte que les 10000 mots les plus courants pour le texte et 33 pour les tags (le premier feature est laissé vide par keras)
Les posts ont été randomisés dans le prétraitement, nous utilisons les premiers 1000 post comme Validation et les suivants 1000 comme notre test set le restant de notre dataset sera utilisé pour entrainement du modèle.

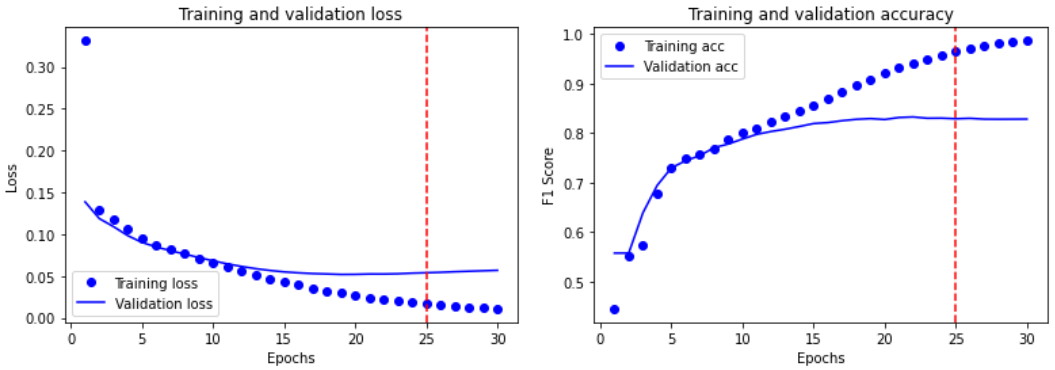
Nous créons deux dictionnaires avec tous les mots trouvés dans le datasets et pas seulement ceux utilisés pour vectoriser les datas, nous avons donc 61763 mots pour les posts et 2966 mots pour les tags. Nous créons aussi un dictionnaire inversée que va nous servir pour traduire les prédictions du modèle

Le modèle choisi est composé de deux couches denses avec 100 unité chacune qui semble être bien adapté pour 32 tags, nous avons tenté un passage à 100 tags avec cette architecture mais le réseau n'est plus capable d'apprendre plus de deux tags.

Exploration des pré traitement du texte

Aucun des modes de tokenisation ne nous ont donné des résultats plus satisfaisants que le mode binaire. Nous avons testé le comptage des mots, le tf idf, la fréquence

Résultats

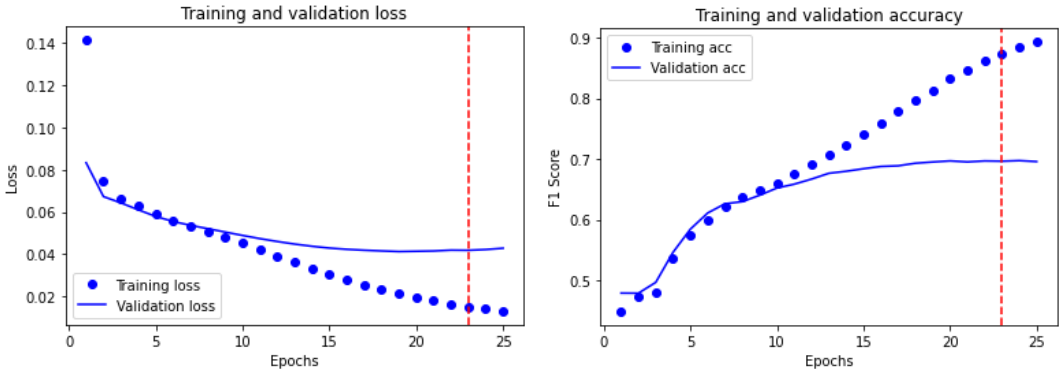


Résultats sur test set avec 32 labels et deux couches cachés avec 100 éléments, régularisation par earlystopping sur 20 epoch

f1-score: 0.82
precision_score: 0.92
recall_score: 0.74

5) Modèle Multi Layer Perceptron Classifier avec Embedding KERAS/TensorFlow

Pour avoir une meilleur représentation de nos données nous allons employer une couche d'embedding avec des vecteurs de dimension 30 et une régularisation du modèle par la méthode Dropout et early stopping lorsque le loss dans le validation arrête de diminuer.



Résultats sur test set après random search avec CV avec 32 labels et deux couches cachés avec 100 éléments, régularisation par earlystopping sur 20 epoch

f1-score: 0.70
precision_score: 0.94
recall_score: 0.55

Nous avons une très bonne précision à 96% (TP/(TP+FN)) et un recall de 56 % (TP/(TP+FN)) cela signifie que notre modèle se trompe rarement sur le prédictions positives mais laisse passer comme négatif un peu moins de la moitié de labels positifs

Lors du déploiement de notre API, il sera possible d'ajuster le seuil de positivité à moins de 0.5 pour avoir un peu moins de précision mais un meilleur recall car le but est de proposer un maximum de labels à l'utilisateur final