OPENCLASSROOMS

Mars 2022

RAPPORT DU PROJET 5 CATÉGORISEZ AUTOMATIQUEMENT DES QUESTIONS

Laurent BOUTROIX

étudiant Ingénieur Machine Learning

SOMMAIRE

OBJECTIF DU PROJET

Amateur de Stack Overflow, qui nous a souvent sauvé la mise, nous décidons d'aider la communauté en retour. Pour cela, nous développons un système de suggestion de tag pour le site.

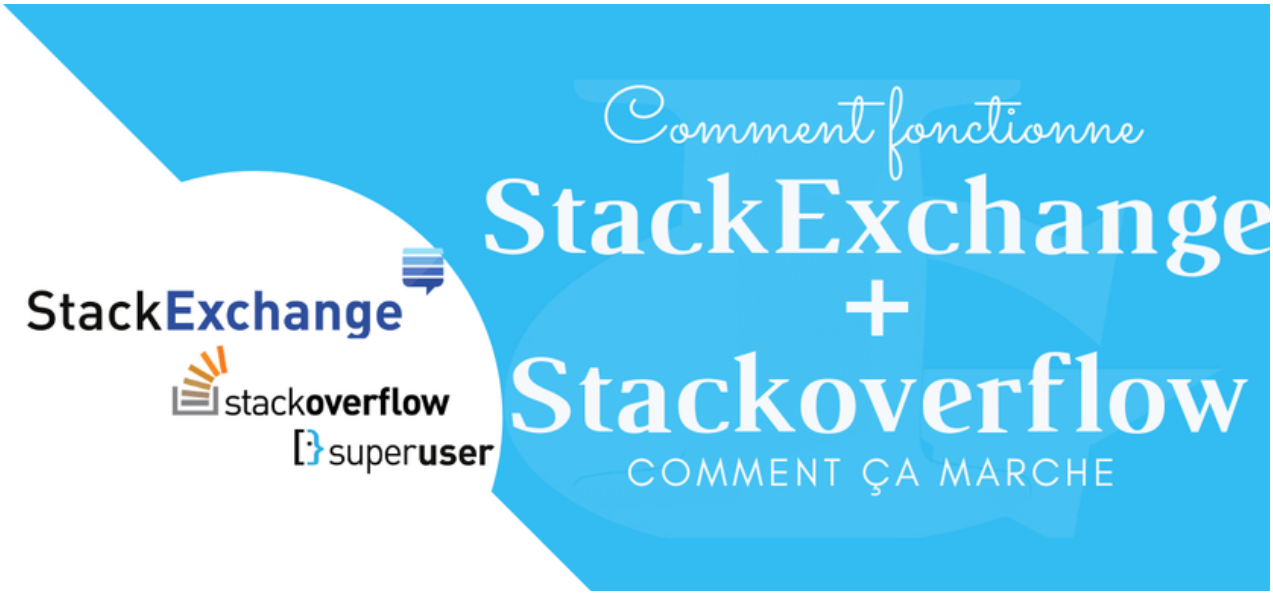
Le système de suggestion de tags prendra la forme d’un algorithme de machine learning qui assigne automatiquement un ou plusieurs tags pertinents à une question.

Une fois l'objet du projet défini, nous nous posons les questions suivantes :

Pouvons-nous mettre en place un algorithme de machine learning qui assigne automatiquement plusieurs tags pertinents à une question ?

Pouvons-nous mettre en place une méthode d’évaluation propre, avec une séparation du jeu de données pour l’évaluation ?

RÉCUPÉRATION 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.

Nous importons les données par **requêtes SQL** :
selon les champs Id, CreationDate, Title, Body, Tags, ViewCount, AnswerCount, Score, FavoriteCount.

Importation de 12 échantillons de 50 000 individus chacun entre le 31/07/2008 et 16/01/2022.

Soit 600 00 rows × 8 columns au total.

```
SELECT Id,CreationDate,Title,Body,Tags,ViewCount,AnswerCount,Score,FavoriteCount
FROM posts WHERE Id BETWEEN i AND j AND ParentId IS NULL
```

EXPLORATION, ANALYSE ET PRÉPARATION DES DONNÉES

En utilisation le langage Python 3, des 12 bases de données importées depuis StackExchange, nous avons constitué une Master base de données nommée df.
Nous devons nettoyer, analyser et préparer cette base de données pour pouvoir l'utiliser de façon optimale dans les modélisations.

N° 01 - **Exploration**

Inspection et nettoyage de df : valeurs manquantes, doublons, suppression de la variable FavoriteCount.

N° 02 - **Analyse**

Features engineering avec création de variables quantitatives à partir des variables qualitatives. Analyse univariée et multivariée des variables quantitatives. Mise en place de filtres permettant d'améliorer la qualité des données qualitatives.

N° 03 - **Préparation**

Utilisation de différentes méthodes et bibliothèques Python pour rendre les variables qualitatives, qui sont des chaînes de caractères, utilisables par les algorithme de modélisation.

Exemple d'une question et de son titre avant et après traitements :

TITLE + BODY = QUESTION

Le titre de la question du premier exemple est :
How do I calculate someone's age based on a DateTime type birthday?

Le corps de la question du premier exemple est :
Given a <code>DateTime</code> representing a person's birthday, how do I calculate their age in years?

Le titre et le corps de la question du premier exemple après un nettoyage des balises html et uniformisation est :
how do i calculate someone s age based on a datetime type birthday given a datetime representing a person s birthday how do i calculate their age in years

lower() + regex + BeautifSoup + languedetect

Le titre et le corps de la question du premier exemple une fois tokénisés, lemmatizés et les stopwords retirés est :
['calculate', 'age', 'base', 'datetime', 'type', 'birthday', 'give', 'datetime', 'represent', 'person', 'birthday', 'calculate', 'age', 'year']

Spacy

TAGS

Les tags de la question du premier exemple à vérifier sont :
<c#><.net><datetime>

Les tags de la question du premier exemple après un nettoyage des balises html et uniformisation est :
csharp dotnet datetime

lower() + regex + BeautifSoup + languedetect

Les tags de la question du premier exemple une fois tokénisés, lemmatizés et les stopwords retirés est :
['csharp', 'dotnet', 'datetime']

Spacy + NLTK/FreqDist()

Word Cloud des 300 Tags les plus fréquents de StackOverflow entre 2008 et 2022

Word Cloud des 300 mots contenus dans le corpus des Titres et des Questions les plus fréquents de StackOverflow entre 2008 et 2022

Visualisation sous forme de Word Cloud des fréquences des mots dans le corpus de la variables Tags et le corpus de la variable Question.

MODÉLISATIONS

Mise en place des modèles

Avant de procéder aux tests des modèles nous devons appliquer des transformations à nos variables :

Variable explicative question_lemtokcl, extraction de features avec le modèle Bag-of-Words :
utilisation du module TfidfVectorizer de la librairie Scikit-Learn pour combiner le CountVectorizer et TfidfTransformer afin de minimiser la redondance de certains mots en appliquant le modèle du Bag-of-Words tout en créant des n-grams de type (1,2) et réduction à 950 features maximum.

Variable cible tag_pop_tok:
utilisation de MultiLabelBinarizer de Scikit-Learn pour encoder nos multiples Tags par individus.

Création des jeux de données train et test pour la variable explicative et la variable cible.

Choix des algorithmes de classification à tester de la bibliothèque Scikit-Learn

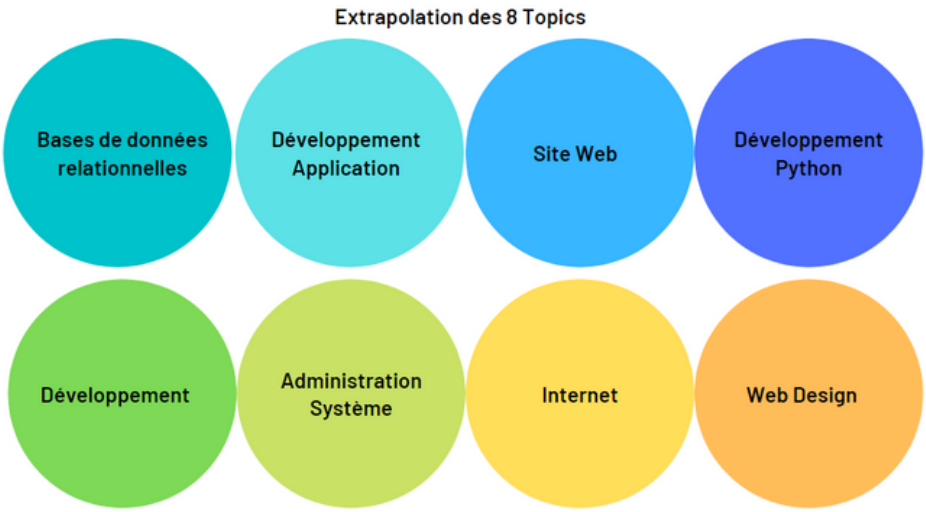
Méthodes supervisées :
SGDClassifier
LogisticRegression
MultinomialNB
ComplementNB
Perceptron
LinearSVC

Méthodes non supervisées :
LatentDirichletAllocation LDA
LDAmulticore de Gensim

3 stratégies de classification :
Multiclass OneVsRestClassifier
Multioutput ClassifierChain
MultioutputClassifier

Méthodes non supervisées :
LatentDirichletAllocation LDA
LDAmulticore de Gensim

Des 2 méthodes de topic modeling, nous retenons la méthode dite de Fast LDA proposée par la bibliothèque Gensim de Scikit-Learn : LDAmulticore. Cet algorithme nous a permis d'extraire de façon optimale 8 topics.



8 topics

Fast LDA

Méthodes supervisées :
3 stratégies 6 algorithmes 8 métriques

	Accuracy	F1	Macro_f1_score	Micro_f1_score	Jaccard	Recall	Precision	Run Time
SGDClass_1VsRestClass	0.217347	0.611639	(0.26903346754666163,)	(0.4922728209355038,)	0.321574	0.346031	0.682643	1.703125
LogisticReg_1VsRestClass	0.186570	0.537518	(0.25313920500263964,)	(0.45007388642600804,)	0.287909	0.308806	0.748492	8.312500
MultinomialNB _1VsRestClass	0.067404	0.405572	(0.07239939598569518,)	(0.2272221535418469,)	0.122387	0.133111	0.548545	0.703125
Perceptron_1VsRestClass	0.153631	0.440573	(0.3422211939158581,)	(0.447454076097123,)	0.312299	0.433082	0.490590	1.437500
LinearSVC_1VsRestClass	0.249777	0.598900	(0.38296501281677636,)	(0.5453339649455234,)	0.374966	0.416860	0.746258	4.000000
ComplementNB_ClassChain	0.001017	0.165594	(0.16397387686250436,)	(0.18767099975127308,)	0.178783	0.519119	0.241034	3.296875
MultinomialNB_ClassChain	0.083047	0.432142	(0.11062797166656929,)	(0.2968837261253211,)	0.163844	0.192497	0.567735	3.125000
LinearSVC_multiClass_y1	0.249777	0.598900	(0.38296501281677636,)	(0.5453339649455234,)	0.374966	0.416860	0.746258	3.062500
SGDClass_multiClass_y1	0.218237	0.610747	(0.27040760951573023,)	(0.4925027052094605,)	0.321993	0.346104	0.683842	1.609375
LogisticRegression_multiClass_y1	0.186570	0.537518	(0.25313920500263964,)	(0.45007388642600804,)	0.287909	0.308806	0.748492	8.062500
SGDClass_1VsRestClass_y1	0.216330	0.609142	(0.2705116554006975,)	(0.4912588314166366,)	0.320809	0.344945	0.686233	1.640625
LogisticReg_1VsRestClass_y1	0.186570	0.537518	(0.25313920500263964,)	(0.45007388642600804,)	0.287909	0.308806	0.748492	8.281250
MultinomialNB _1VsRestClass_y1	0.067404	0.405572	(0.07239939598569518,)	(0.2272221535418469,)	0.122387	0.133111	0.548545	0.734375
Perceptron_1VsRestClass_y1	0.153631	0.440573	(0.3422211939158581,)	(0.447454076097123,)	0.312299	0.433082	0.490590	1.406250
LinearSVC_1VsRestClass_y1	0.249777	0.598900	(0.38296501281677636,)	(0.5453339649455234,)	0.374966	0.416860	0.746258	3.843750
ComplementNB_ClassChain_y1	0.001017	0.165594	(0.16397387686250436,)	(0.18767099975127308,)	0.178783	0.519119	0.241034	3.171875
MultinomialNB_ClassChain_y1	0.083047	0.432142	(0.11062797166656929,)	(0.2968837261253211,)	0.163844	0.192497	0.567735	3.125000

Métriques des tests des 8 algorithmes selon les trois stratégies.

SÉLECTION DU MEILLEUR MODÈLE

	Accuracy	F1	Macro_f1_score	Micro_f1_score	Jaccard	Recall	Precision	Run Time
LinearSVC_multiClass_opt_5	0.257408	0.627853	(0.3618741875618396,)	(0.5507107643708419,)	0.376580	0.416643	0.742038	0.062500
LinearSVC_1VsRestClass_opt_4	0.257408	0.627782	(0.3618645476872594,)	(0.5506413938349607,)	0.376514	0.416570	0.742017	0.062500
LinearSVC_1VsRestClass_opt_3	0.249142	0.598060	(0.38352907665103375,)	(0.5449810606060606,)	0.374747	0.416787	0.744678	0.062500
LinearSVC_1VsRestClass_opt_2	0.257408	0.627853	(0.3618741875618396,)	(0.5507107643708419,)	0.376580	0.416643	0.742038	0.078125
LinearSVC_1VsRestClass_opt_1	0.247234	0.594602	(0.3778800197051535,)	(0.5378336854234684,)	0.367911	0.405634	0.749669	0.031250
LinearSVC_1VsRestClass_opt	0.249777	0.598819	(0.38295720836886543,)	(0.5452650528210716,)	0.374904	0.416787	0.746233	0.046875
LinearSVC_1VsRestClass_y1	0.249777	0.598900	(0.38296501281677636,)	(0.5453339649455234,)	0.374966	0.416860	0.746258	3.843750
LinearSVC_1VsRestClass	0.249777	0.598900	(0.38296501281677636,)	(0.5453339649455234,)	0.374966	0.416860	0.746258	4.000000
LinearSVC_multiClass_y1	0.249777	0.598900	(0.38296501281677636,)	(0.5453339649455234,)	0.374966	0.416860	0.746258	3.062500

Métriques des tests optimisés de l'algorithme LinearSVC() selon les 2 stratégies MultiClass OneVsRest et MultiOuputClassifier.

	Test	Predictions_le_ best_model
0	(mysql, php, sql)	(mysql,)
1	(on, rails, ruby)	(on, rails, ruby)
2	(csharp, dotnet, html, vbdotnet)	()
3	(csharp, date)	()
4	(javascript, reactjs)	(reactjs,)
5	(csharp, dotnet, exception)	(csharp,)
6	(winformicrosoft,)	()
7	(perl,)	()
8	(vba,)	(vba,)
9	(python,)	(excel, python)
10	(aspnet, jquery, mvc)	(aspnet, jquery, mvc)
11	(csharp,)	()
12	(android, ios, objectivec)	()
13	(python,)	(python,)
14	(c++,)	()
15	(ios,)	()
16	(algorithm, python)	()
17	(aspnet, csharp, winformicrosoft)	(aspnet,)
18	(aspnet,)	(aspnet,)
19	(html, mysql, php)	(php,)
20	(iphone,)	(iphone,)
21	(dataframe, pandas, python)	(pandas, python)
22	(csv, pandas, python)	(csv, pandas, python)
23	(c++,)	(c++,)
24	(javascript,)	(html, javascript)
25	(flutter,)	(css,)
26	(csharp, data, mysql)	(mysql,)
27	(javascript, nodedotjs, reactjs)	()
28	(aspnet,)	()
29	(aspnet,)	(aspnet,)

Comparaison des Tags prédits par le best model et le jeu de données test.

La meilleure optimisation via GridSearch est :

Stratégie Multiouptut Classifier pour l'algorithme LinearSVC()

avec les paramètres :

{'estimator__C': 1, 'estimator__dual': True, 'estimator__intercept_scaling': 5,
'estimator__max_iter': 500,
'estimator__multi_class': 'crammer_singer',
'estimator__random_state': None}

CONCLUSION

Pour catégorisez automatiquement des questions du site web stackoverflow.com nous avons mis en oeuvre l'algorithme LinearSCV() de la bibliothèque Scikit-Learn en lui appliquant la stratégie MultiOuputClassifier

Réponses aux hypothèses de départ :

Nous pouvons mettre en place un algorithme de machine learning qui assigne automatiquement plusieurs tags pertinents à une question, il s'agit de l'agorithme LinearSCV() de Scikit Learn avec la stratégie Multiouptut Classifier.

Nous pouvons mettre en place une méthode d'évaluation propre, avec une séparation du jeu de données pour l'évaluation par la création de jeux de données train et test pour la variable explicative et la variable cible et la mise en place de différents métriques permettant la sélection du meilleur modèle.

Nous vous remercions pour votre intérêt pour cette étude de modélisation de générateur de mots clés.