



stack
overflow

Catégorisez automatiquement
des questions



Sommaire

1. Contexte du projet	3
2. Récupération des données	4
3. Exploration des données	5-7
4. Nettoyage des données	8-11
5. Prétraitement des données	12
6. Feature engineering	13
7. Méthode non supervisée	14-15
8. Méthode supervisée	16-17
9. Mise en place de l'API	18
10. Déploiement continu de l'API	19
11. Démonstration de l'API	20
12. Conclusion	21-22





Contexte du projet

Problématique : Le responsable de StackOverflow souhaite améliorer la gestion des tags, parfois mal définis par les utilisateurs du site.



Objectif : développer un système générant les suggestions de tags.



Missions :

- Récupérer les données, les nettoyer et réaliser un feature engineering.
- Présenter une approche non supervisée.
- Présenter un approche supervisée en enregistrant les résultats dans MLFlow.
- Sélectionner le modèle offrant les meilleurs résultats.
- Mettre en production une API pour illustrer le modèle à travers des requêtes utilisateurs.



Résultat attendu :

- Un notebook pour l'analyse exploratoire des données,
- Un notebook pour la requête via l'API StackAPI,
- Un notebook pour la méthode non supervisée,
- Un notebook pour les méthodes supervisées,
- Mise en production d'un API.



Récupération des données

Requête SQL et requête API

Requête SQL

```
1 SELECT TOP 50000 Title, Body, Tags, Id, CreationDate, Score, ViewCount, AnswerCount  
2 FROM Posts  
3 WHERE  
4     PostTypeId = 1 AND  
5     Body IS NOT NULL AND  
6     ViewCount > 100 AND  
7     Score > 5 AND  
8     AnswerCount > 0 AND  
9     LEN(Tags) - LEN(REPLACE(Tags, '<', '')) >= 5  
10 ORDER BY  
11     CreationDate DESC;
```

Extraction de 50 000 questions :

- Les plus consultées,
- Ayant au moins une réponse,
- Comportant au moins 5 tags,
- Récupération du titre, du corps, des tags, de l'ID, de la date de création, du score, du nombre de vues et de réponses.

Export :



Requête API

```
# Initialiser l'API Stack Overflow  
SITE = StackAPI('stackoverflow')  
  
# Paramètres de la requête  
params = {  
    'fromdate': 1457136000,  
    'todate': 1701388801,  
    'min': 50,  
    'tagged': 'python',  
    'score': 50,  
    'sort': 'votes',  
    'order': 'desc',  
    'pagesize': 10  
}  
  
# Effectuer la requête  
questions = SITE.fetch('questions', **params)  
questions
```

```
# Extraire les données nécessaires  
data = []  
for item in questions['items']:  
    question_data = {  
        'date': pd.to_datetime(item['creation_date'], unit='s'),  
        'question_id': item['question_id'],  
        'title': item['title'],  
        'tags': item['tags'],  
        'score': item['score'],  
    }  
    data.append(question_data)
```

Test de l'API :

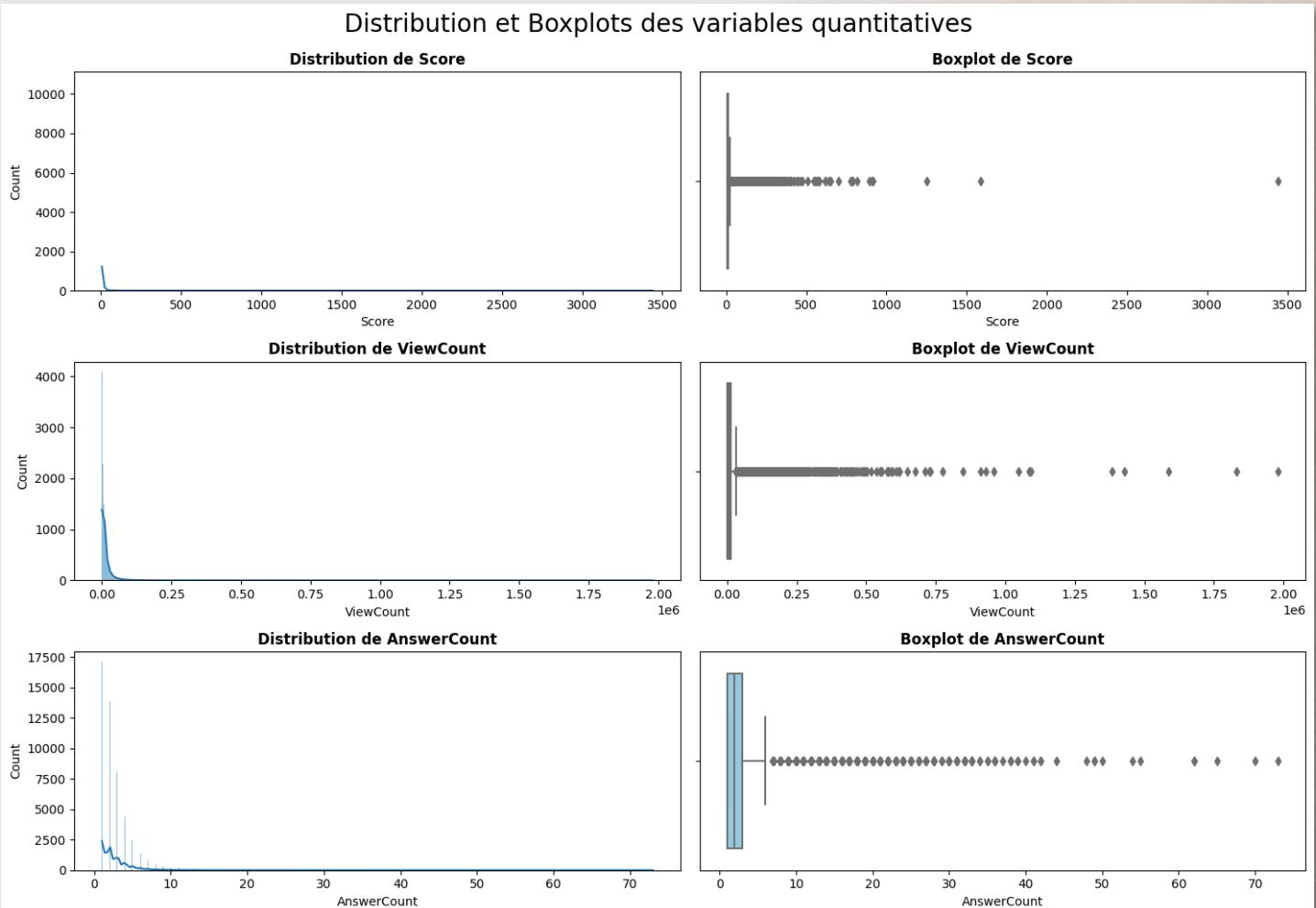
- 50 questions,
- Tagguées Python,
- Avec un score supérieur à 50,
- Récupération de la date, de l'ID, du titre, des tags et du score.

Nous avons utilisé le fichier .csv contenant les 50 000 questions pour la suite du projet, en accordant une attention particulière au respect des principes du RGPD lors de l'exportation.



Exploration des données .1/3

Distribution des variables quantitatives



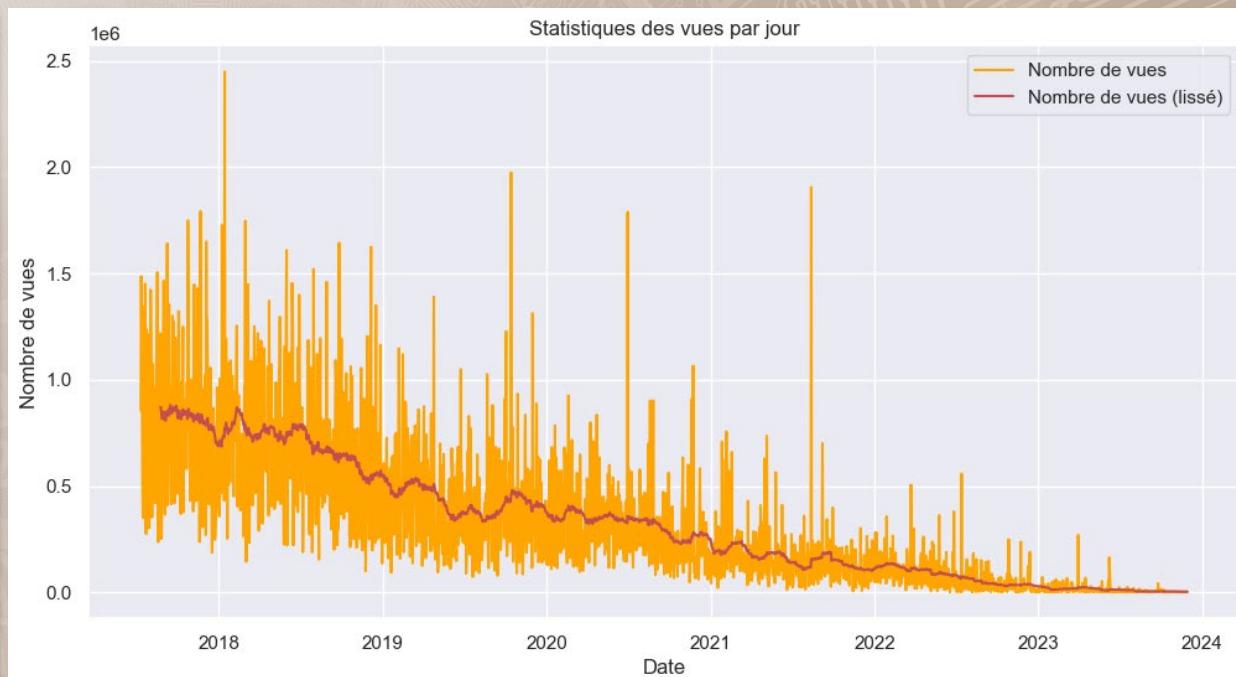
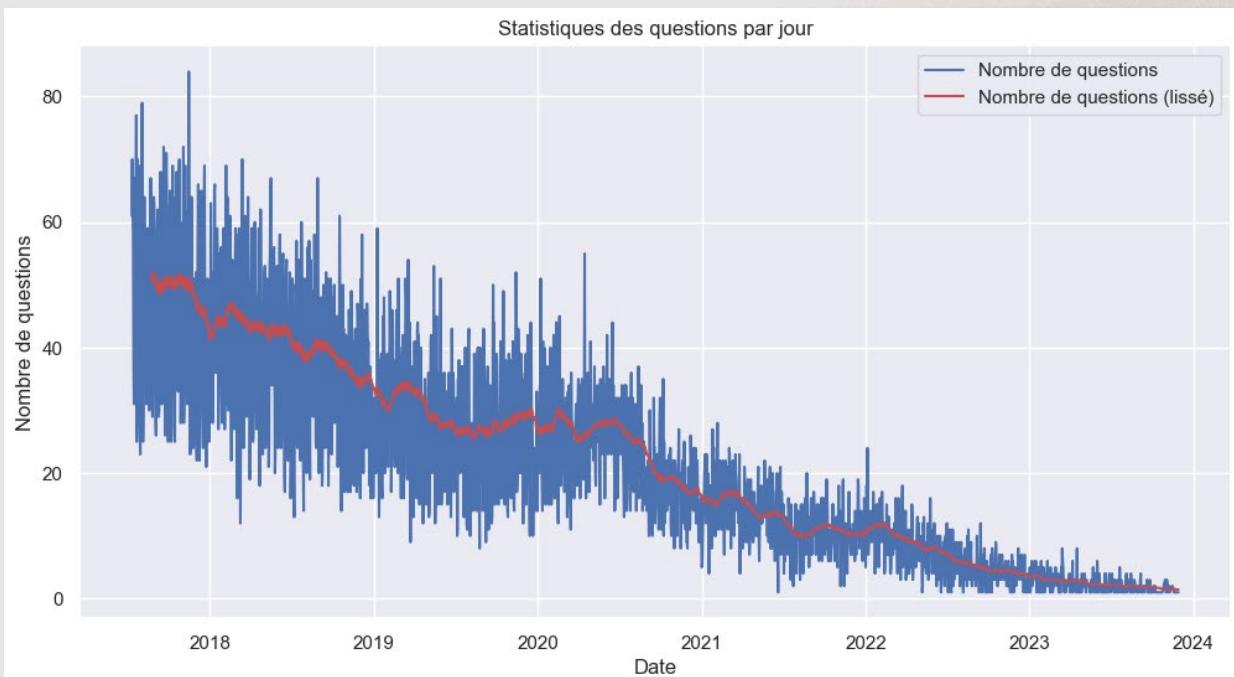
Toutes les données numériques présentent une distribution étalée vers la droite, ce qui signifie que certaines questions auront beaucoup plus de vues, de réponses et de scores plus élevés que d'autres :

- La majorité des scores se situe autour de 5.
- Le nombre de vues se situe généralement autour d'un minimum de 100.
- Le nombre de réponses est souvent compris entre 1 et 5.



Exploration des données .2/3

Statistiques par jour



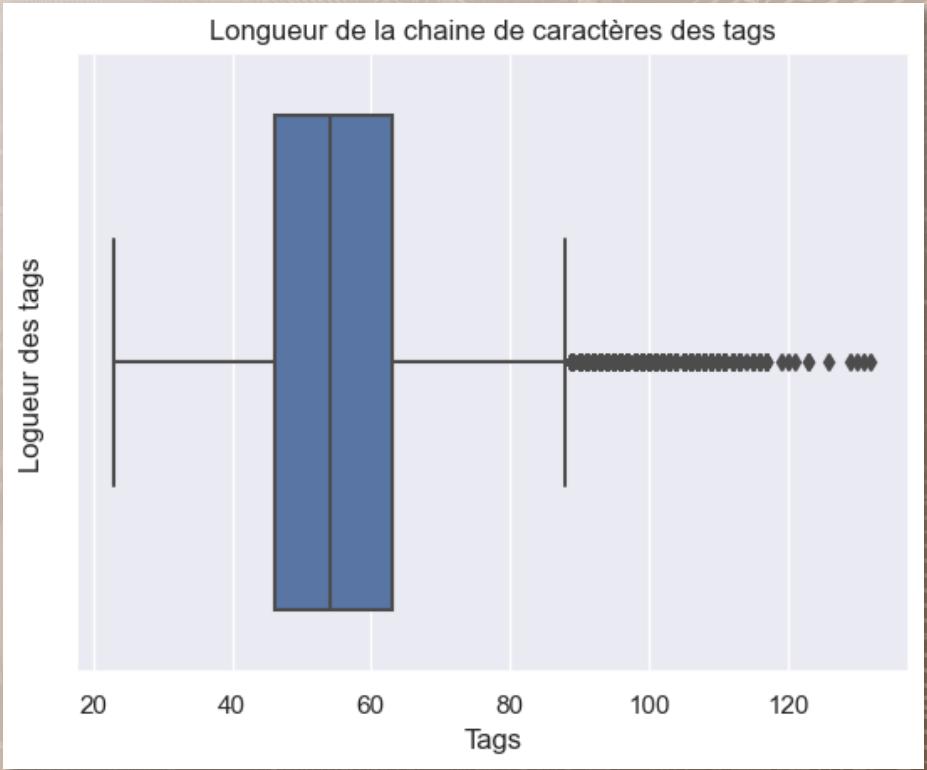
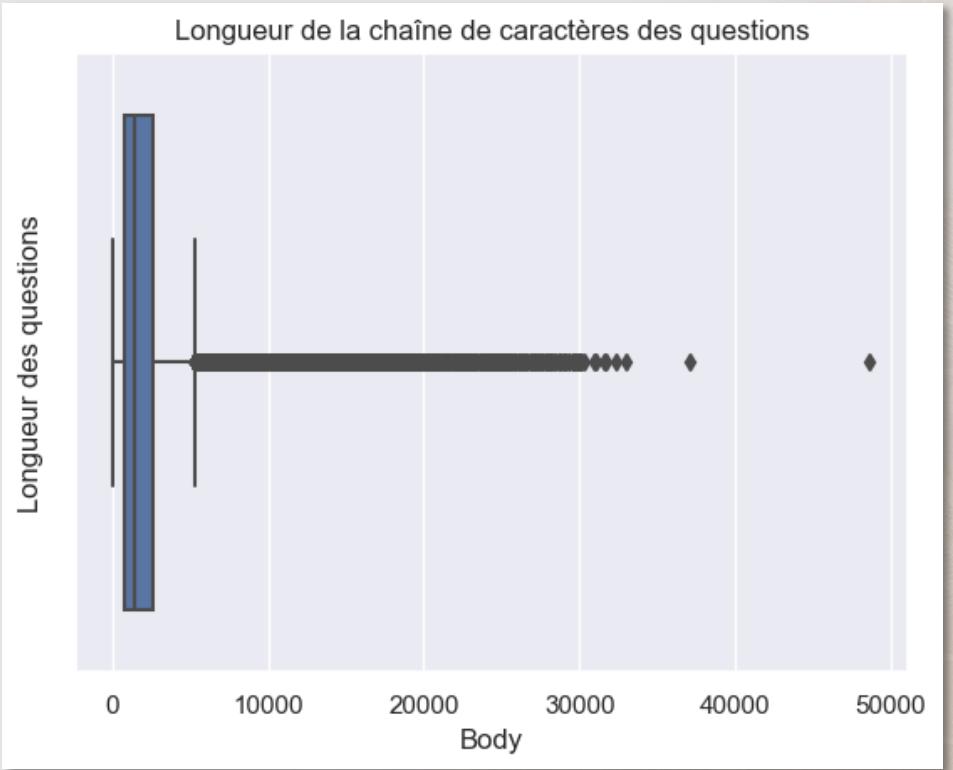
On peut remarquer une baisse du nombre de questions entre 2017 et 2023, atteignant même environ 5 questions par jour vers la fin de 2023. De manière similaire, le nombre de vues diminue également.

On peut donc supposer que l'accessibilité des LLM (Large Language Models) en libre accès a contribué à réduire l'intérêt pour StackOverflow.



Exploration des données .3/3

Nombre de caractères (questions et tags)



Étant donné que les questions peuvent être relativement longues, nous avons déjà envisagé qu'une étude approfondie du corpus serait nécessaire pour comprendre la composition des documents.

Par conséquent, une analyse détaillée de la structure des questions et des tags sera requise.



Nettoyage des données .1/4

Analyse du corpus

Après avoir concaténé les titres et les questions, nous avons examiné un échantillon du corpus pour comprendre sa composition :

```
UIImage Crop Out Transparent Pixels <p><strong>I need help to create perfect clear image after cropping transparent pixel from UIImage using swift 4.</strong></p>
<blockquote>
<p>I want to create image with remove transparent color, So for that i have used below code. But it blur image with small small square rectangles appears.</p>
</blockquote>
<h1>Using of below code</h1>
<pre><code>let imageCroppedBg = imgWithClearBackground!.cropAlpha()

//UIImage extension
extension UIImage {
    func cropAlpha() -> UIImage {
        let cgImage = self.cgImage!
        let width = cgImage.width
        let height = cgImage.height
    }
}</code></pre>
```

Il est observé que le corpus est constitué de texte, de code et de balises HTML. Pour éliminer le code et les balises HTML, nous avons utilisé BeautifulSoup.

BeautifulSoup

'Data binding in adapter of recyclerView - Android I am using in my project to populate a . How can I handle clicks on items? Here is my XML layout: I specified the handler in this line: And below is my adapter: However, the following code doesn't work: Detecting/Diagnosing Thread Starvation I am doing some performance/scalability testing of an IIS application that occasionally seems to slow down to a crawl in production. I'm able to reproduce the slowness consistently using NUnit. CPU and Memory do not spike during the testing, or when the slowness occurs in production. My strong suspicion is that the application is suffering from thread starvation, since it does not appear to be CPU, Memory, I/O, or database access that is causing the bottleneck. I do see signs of what appear to be thread starvation; for example, NLog's async log file writes tend to have long periods of silence followed by bursts of activity with older time stamps (i.e. a lower-priority thread is waiting for threads to free up in order to write). What steps can I take to definitively determine that the application is indeed thread starved, and (assuming that is the case) pinpoint the exact areas of the system that are causing the problem? Edit I neglected to mention that almost all the code is synchronous (it's a legacy system). HTTPS through proxy completely encrypted, including SSL CONNECT I am trying to test a proxy that is expecting an SSL handshake immediately, even if the client is going to use an SSL CONNECT method.'



Nettoyage des données .2/4

Analyse du corpus

Afin de prédire des tags pour les questions de StackOverflow, il est crucial d'établir une liste de vocabulaire technique.

Ainsi, nous avons sollicité ChatGPT pour obtenir une liste de termes techniques couramment utilisés dans le jargon du langage de programmation.



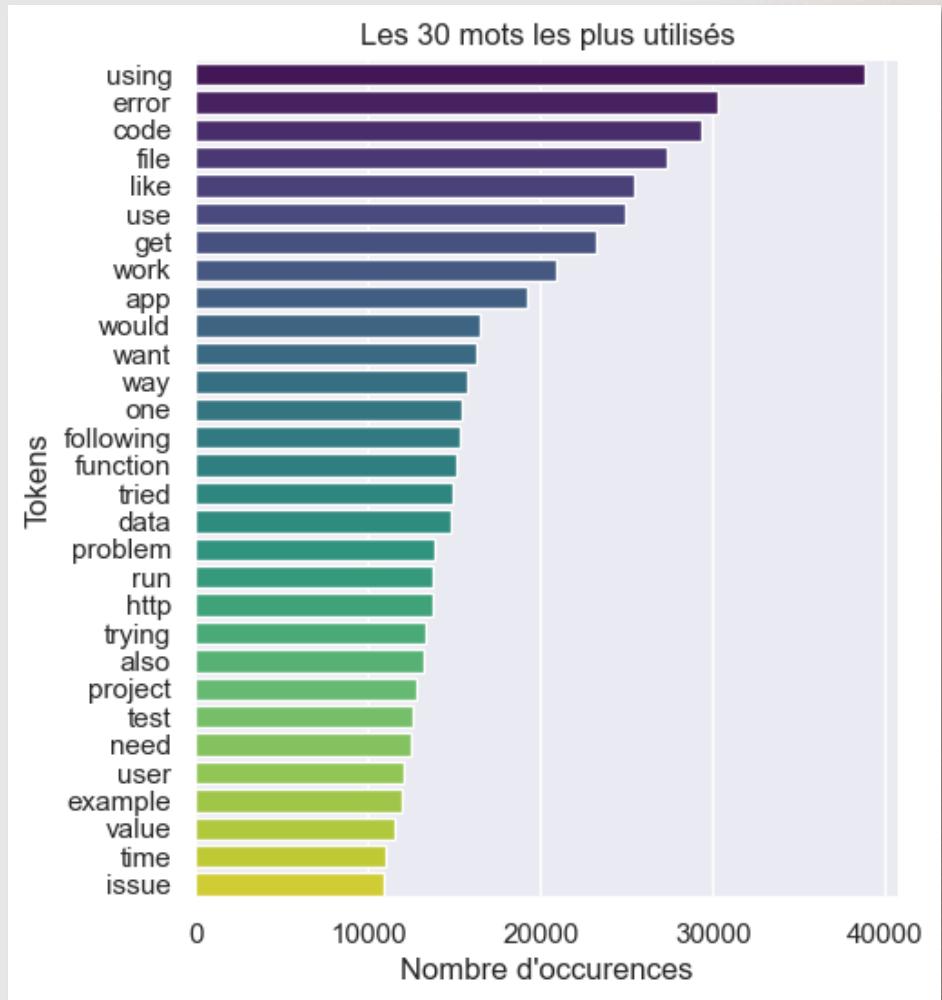
```
technical_terms = [  
    '.net',  
    'algorithm',  
    'api',  
    'artificialintelligence',  
    'bigdata',  
    'blockchain',  
    'cloudcomputing',  
    'cybersecurity',  
    'c++',  
    'c#',  
    'ios',  
    'datascience',  
    'database',  
    'deeplearning',  
    'devops',  
    'docker',  
]
```

Ensuite, nous avons enrichi cette liste au besoin afin de garantir une couverture maximale des termes techniques, ce qui nous permettra une meilleure catégorisation des questions.



Nettoyage des données .3/4

Analyse du corpus



Avant de procéder à la tokenisation de chaque document individuellement, nous avons d'abord tokenisé le corpus dans son ensemble. De plus, nous avons opté pour la lemmatisation plutôt que la stammatisation afin de préserver au mieux le sens des mots.

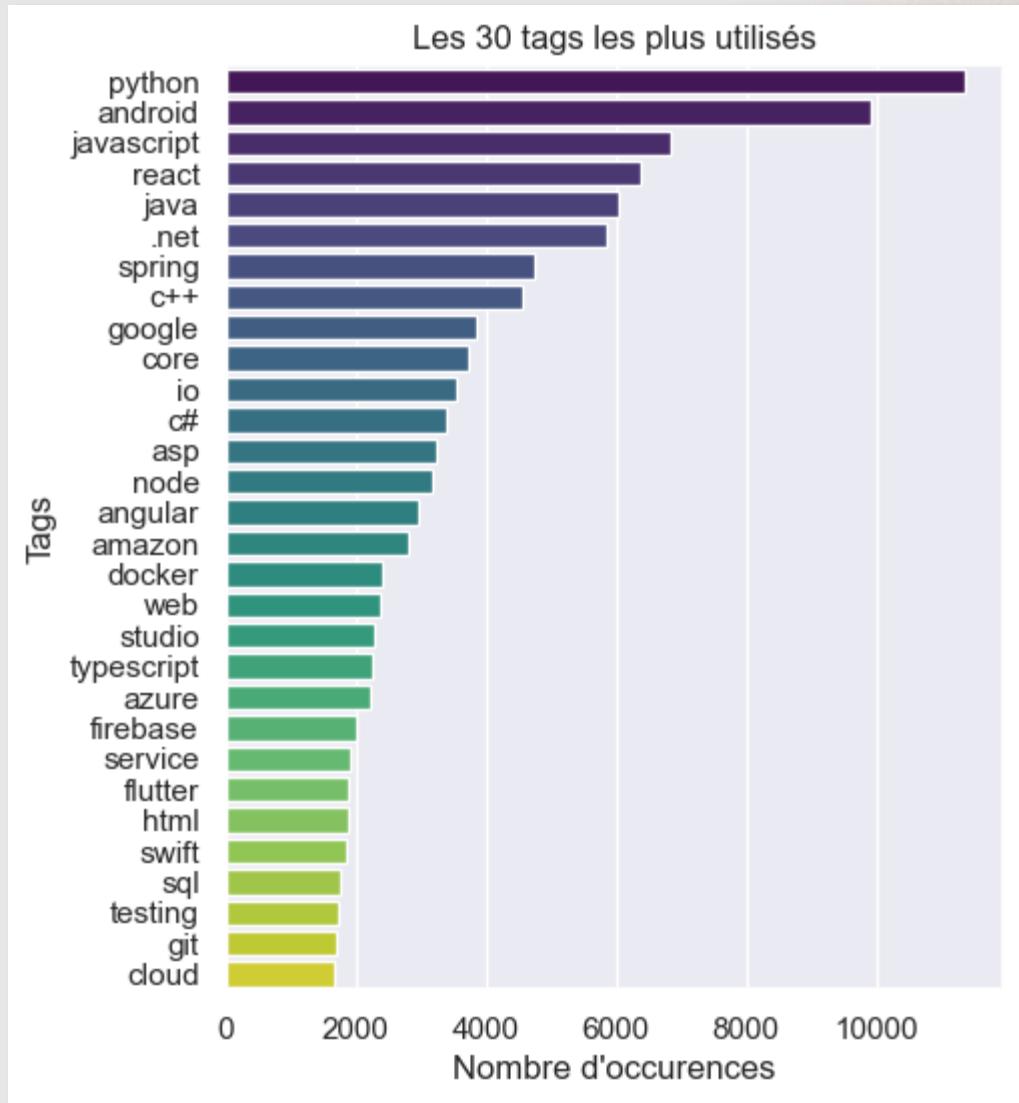
Cette démarche visait à identifier les mots couramment utilisés dans le corpus qui n'apportent pas d'information supplémentaire pour la catégorisation des questions.

L'objectif était de créer une liste de mots communs que nous exclurons lors de la tokenisation des documents, tout comme les mots vides ou ceux présents une seule fois.



Nettoyage des données .4/4

Les tags



Nous avons choisi de limiter le nombre de tags à 20 pour optimiser les performances du modèle tout en couvrant une large part des questions posées.

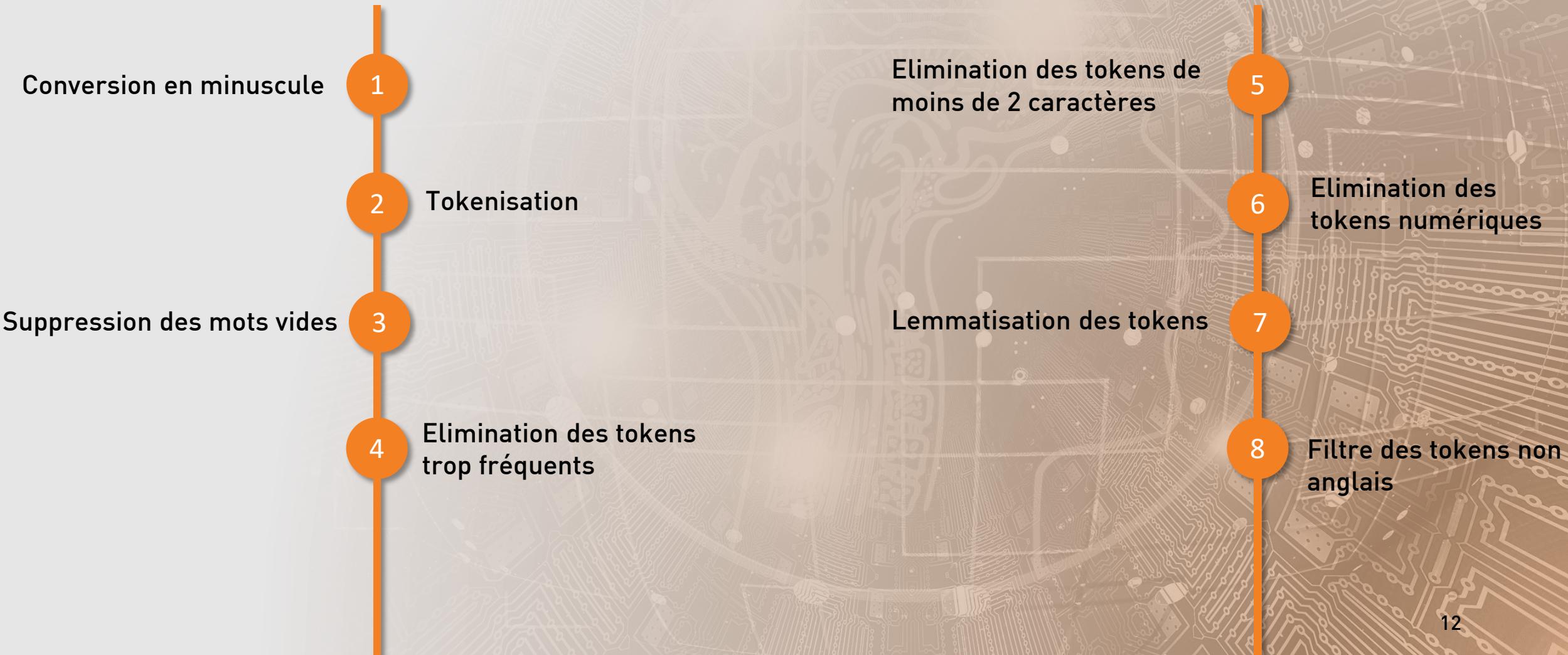
Cette approche nous permet de maintenir un modèle léger et efficace, tout en conservant une capacité de catégorisation robuste.



Prétraitement des données

Fonction de processing des documents

Les étapes suivantes résument la construction de la fonction de nettoyage des documents et des tags :





Feature engineering

Matrices de type bag of words

Notre objectif est de transformer chaque token unique du corpus en une variable distincte. Ainsi, nous pourrons encoder ces variables dans une matrice afin de prédire les différents tags à l'aide de données numériques.

CountVectorizer

Nous avons simplement compté le nombre d'occurrences de chaque token dans le document.

```
# Création d'une matrice document-token Bag of Words
bow_vectorizer = CountVectorizer(analyzer = 'word',
                                 max_df = 0.7,
                                 min_df = 0.05,
                                 stop_words = 'english')

data_vectorized_bow = bow_vectorizer.fit_transform(df.clean_title_body.values)
```

TF-IDF

Nous pouvons déterminer les mots importants de la manière suivante :

- TF mesure l'importance du token dans le document.
- IDF mesure la pertinence du token en fonction de sa distribution et de son utilisation dans l'ensemble du corpus.

```
# Création d'une matrice document-token TF-IDF
tfidf_vectorizer = TfidfVectorizer(analyzer = 'word',
                                    max_df=0.7,
                                    min_df=0.05,
                                    stop_words = 'english')

data_vectorized_tfidf = tfidf_vectorizer.fit_transform(df.clean_title_body.values)
```

Dans les deux cas, les mots (tokens) qui apparaissent dans plus de 70% des documents sont ignorés (max_df=0.7). Bien que nous ayons déjà éliminé les mots communs au préalable, certains mots peuvent encore ne pas caractériser le sujet. Le choix de 70% a été déterminé après plusieurs tests et vise à ne conserver que les mots contribuant véritablement à la compréhension des catégories des questions.

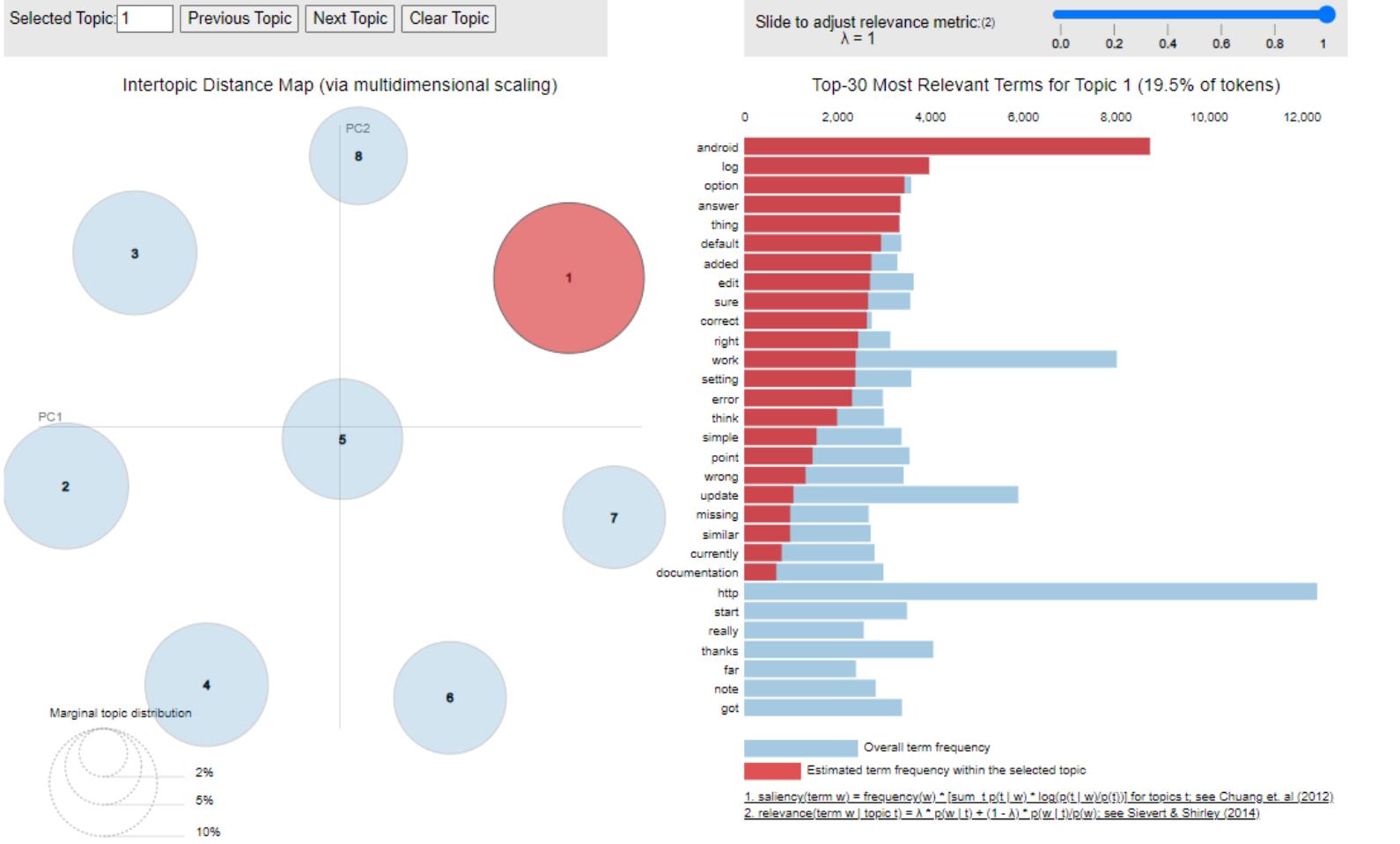
De même, certains mots sont très peu fréquents et ne permettent pas de catégoriser efficacement les questions. Après plusieurs tests, nous avons décidé de ne pas conserver les tokens qui apparaissent dans moins de 5% des documents.



Méthode non supervisée .1/2

LDA avec Countvectorizer

LDAvis (2D) du topic modeling (Countvectorizer) :



Nous avons identifié 8 sujets principaux :

- Topic 1 : Android
- Topic 2 : Windows
- Topic 3 : Python
- Topic 4 : Api (io, net, core)
- Topic 5 : Json
- Topic 6 : Package
- Topic 7 : Data, java
- Topic 8 : Application

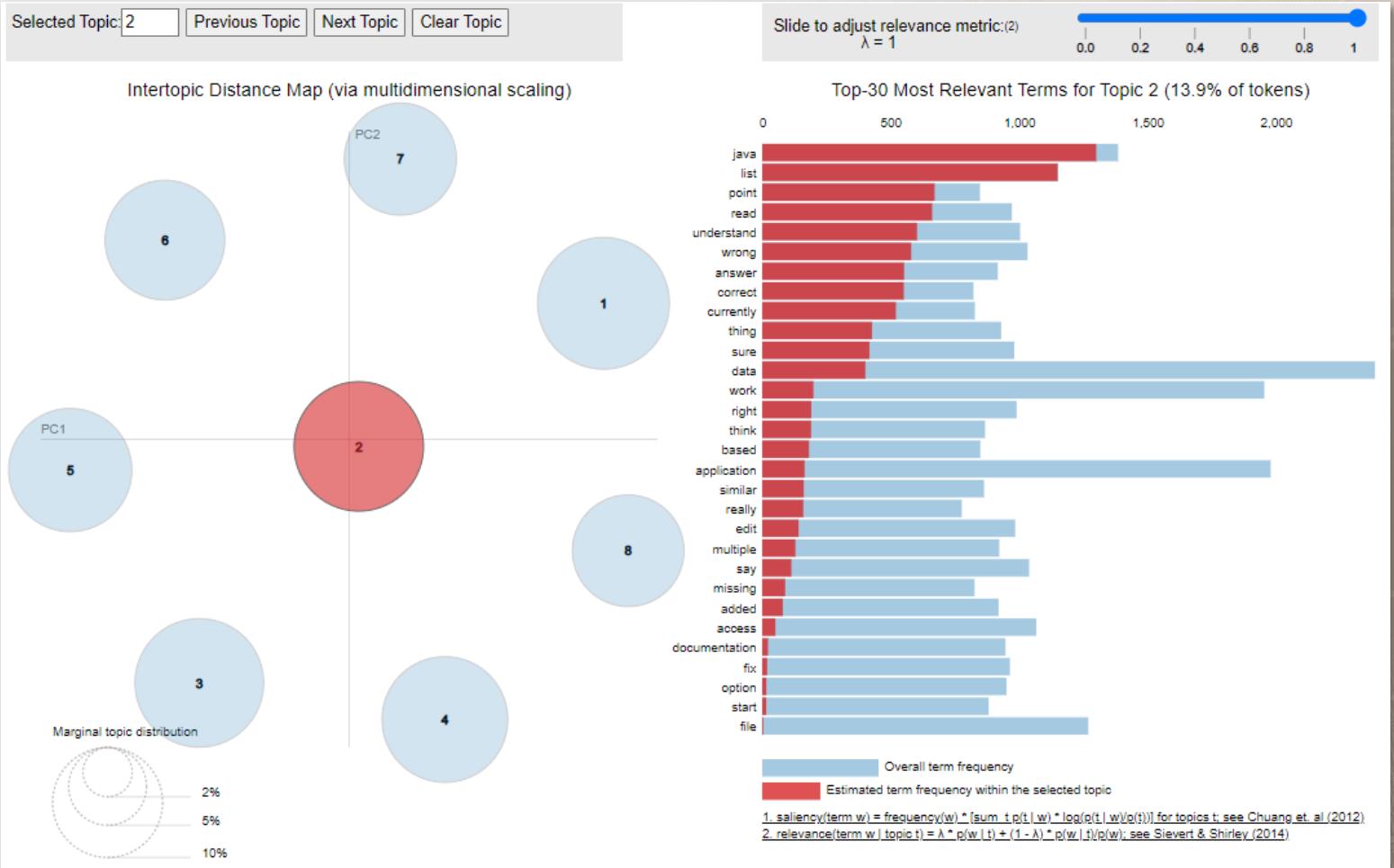
Score de cohérence : 0.41



Méthode non supervisée .2/2

LDA avec TF-IDF

LDAvis (2D) du topic modeling (TF-IDF) :



Nous avons identifié également 8 sujets principaux :

- Topic 1 : Net, json
- Topic 2 : Java
- Topic 3 : Html
- Topic 4 : Api (io, access)
- Topic 5 : Python
- Topic 6 : Inside, update
- Topic 7 : Android
- Topic 8 : Instead, documentation

Score de cohérence : 0.425



Méthode supervisée .1/2

Word embedding

Le word embedding va nous permettre d'améliorer la recherche d'information.

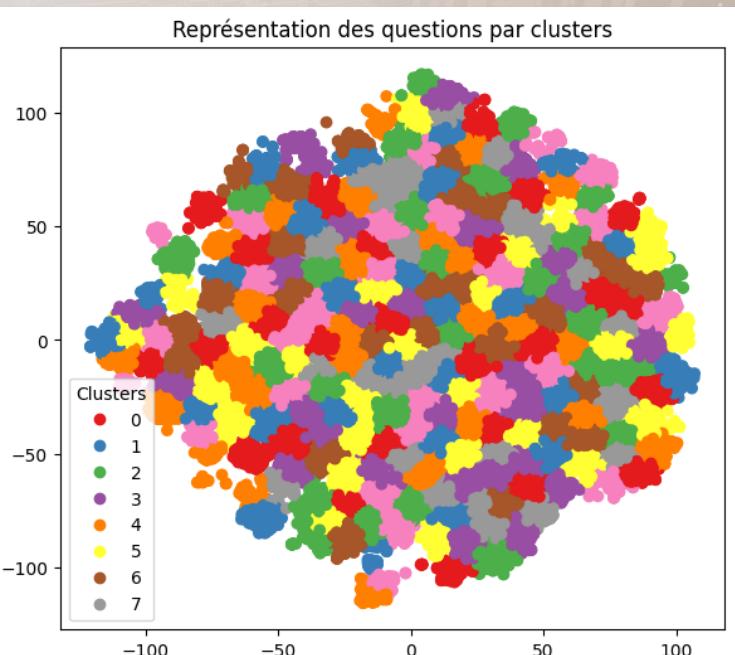
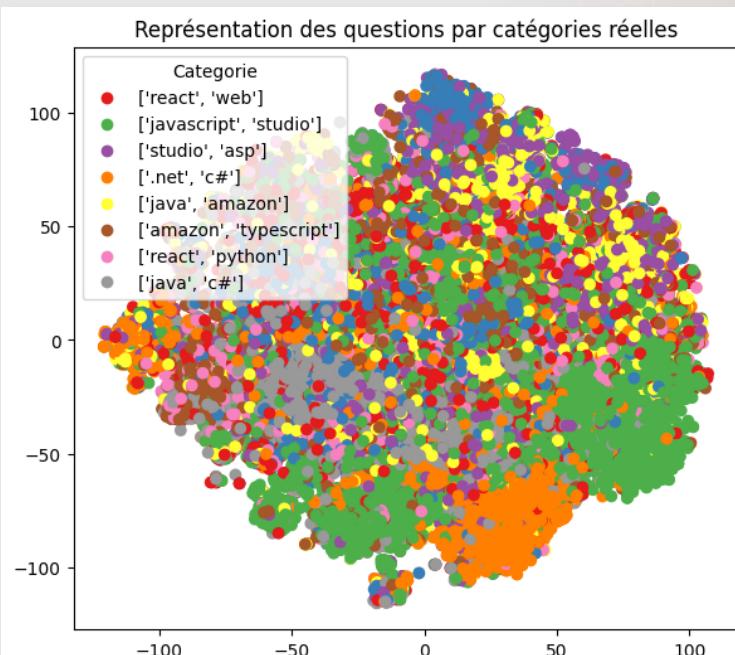
Le but ici, est d'avoir une méthode plus représentative du sens des mots en rapprochant les mots similaires.

L'idée est donc de pouvoir représenter les mots sous forme de vecteur dans un espace vectoriel et de pouvoir comprendre leur similarité (ou leur sens) grâce à la distance qui les sépare dans ce même espace.

Pour cela, nous avons utilisé Word2Vec, Bert et USE.

La finalité est qu'on va pouvoir entraîner les modèles sur différentes features représentatives des documents afin de pouvoir les caractériser.

Visualisation TSNE de la matrice USE :

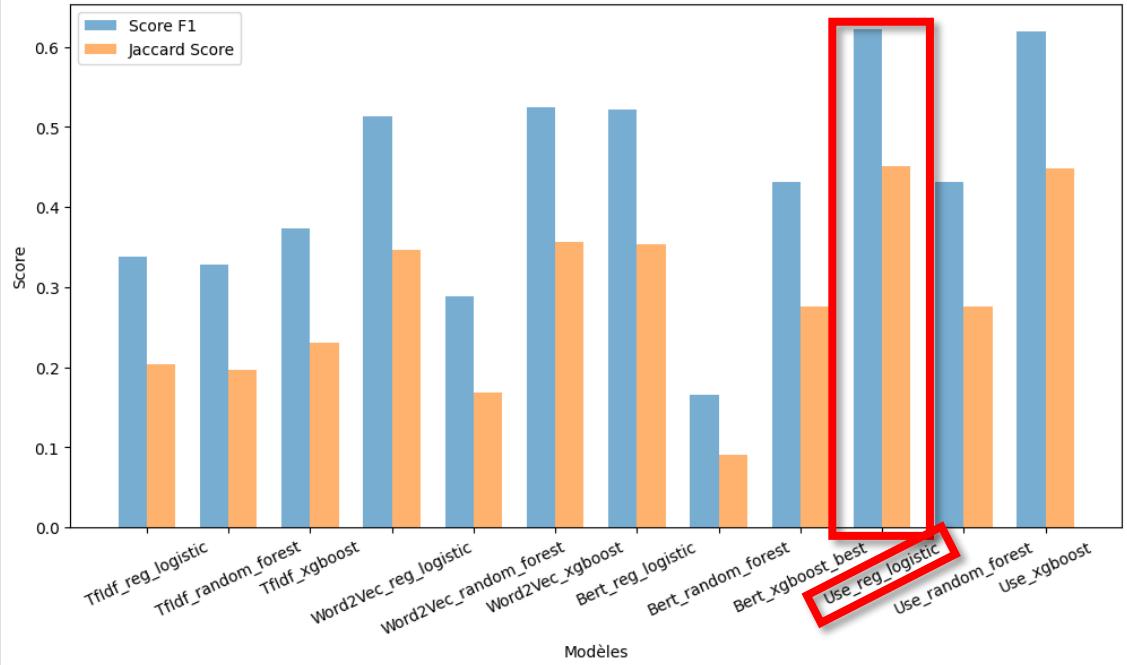




Méthode supervisée .2/2

Scores et choix du modèle

Comparaison des scores des modèles



Enregistrement des runs dans MLFlow Tracking :

Run Name	Created	Duration	accuracy	f1_score	jaccard_score	precision	recall
Use_reg_logistic_best_params	18 days ago	12.2s	0.41085719...	0.62215801...	0.45154525...	0.75656683...	0.52830188...
Use_xgboost_best_params	17 days ago	8.7min	0.41126969...	0.61909620...	0.44832682...	0.82298226...	0.49617384...
Word2Vec_xgboost_best_params	19 days ago	1.4min	0.32901575...	0.52478360...	0.35573330...	0.78204534...	0.39488287...
Bert_reg_logistic_best_params	19 days ago	53.7s	0.31053543...	0.52209412...	0.35326615...	0.73971722...	0.40341141...
Word2Vec_reg_logistic_best_params	20 days ago	4.5s	0.31235046...	0.51396861...	0.34586659...	0.74520031...	0.39225422...
Use_random_forest_best_params	17 days ago	1.7h	0.25740450...	0.43186769...	0.27540258...	0.90379605...	0.28371984...
Bert_xgboost_best_params	18 days ago	21.7min	0.24280174...	0.43175378...	0.27530994...	0.85085910...	0.28926923...
Tfidf_xgboost_best_params	20 days ago	28.8s	0.19924098...	0.37392222...	0.22995346...	0.75717085...	0.24826216...
Tfidf_reg_logistic_best_params	20 days ago	5.1s	0.17630558...	0.33742445...	0.20295285...	0.82646322...	0.21198668...
Tfidf_random_forest_best_params	20 days ago	6.3min	0.16863295...	0.32771793...	0.19597048...	0.87329286...	0.20170570...
Word2Vec_random_forest_best_params	20 days ago	18.1min	0.17069548...	0.28812402...	0.16830893...	0.87091069...	0.17261522...
Bert_random_forest_best_params	19 days ago	50.3min	0.08052140...	0.16544469...	0.09018245...	0.91426893...	0.09095157...

Nous avons utilisé une validation croisée pour entraîner trois types de classifiers pour chaque matrice de données (TF-IDF, Word2Vec, BERT et USE).

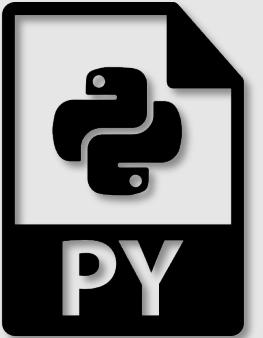
Tous les scores ont été enregistrés sur MLFlow Tracking pour assurer un suivi précis et la reproductibilité du projet.

Le modèle sélectionné est la régression logistique utilisant la matrice d'embedding USE.



Mise en place de l'API

Pour la création de l'API, nous avons opté pour le framework Flask et avons effectué des tests avec Pytest.



Code_API.py



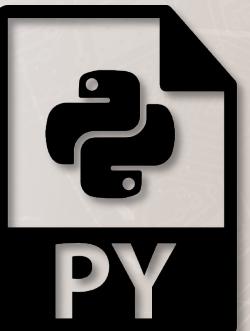
Test de l'API



Flask

Backend

Nous avons choisi Streamlit pour développer une interface utilisateur conviviale.



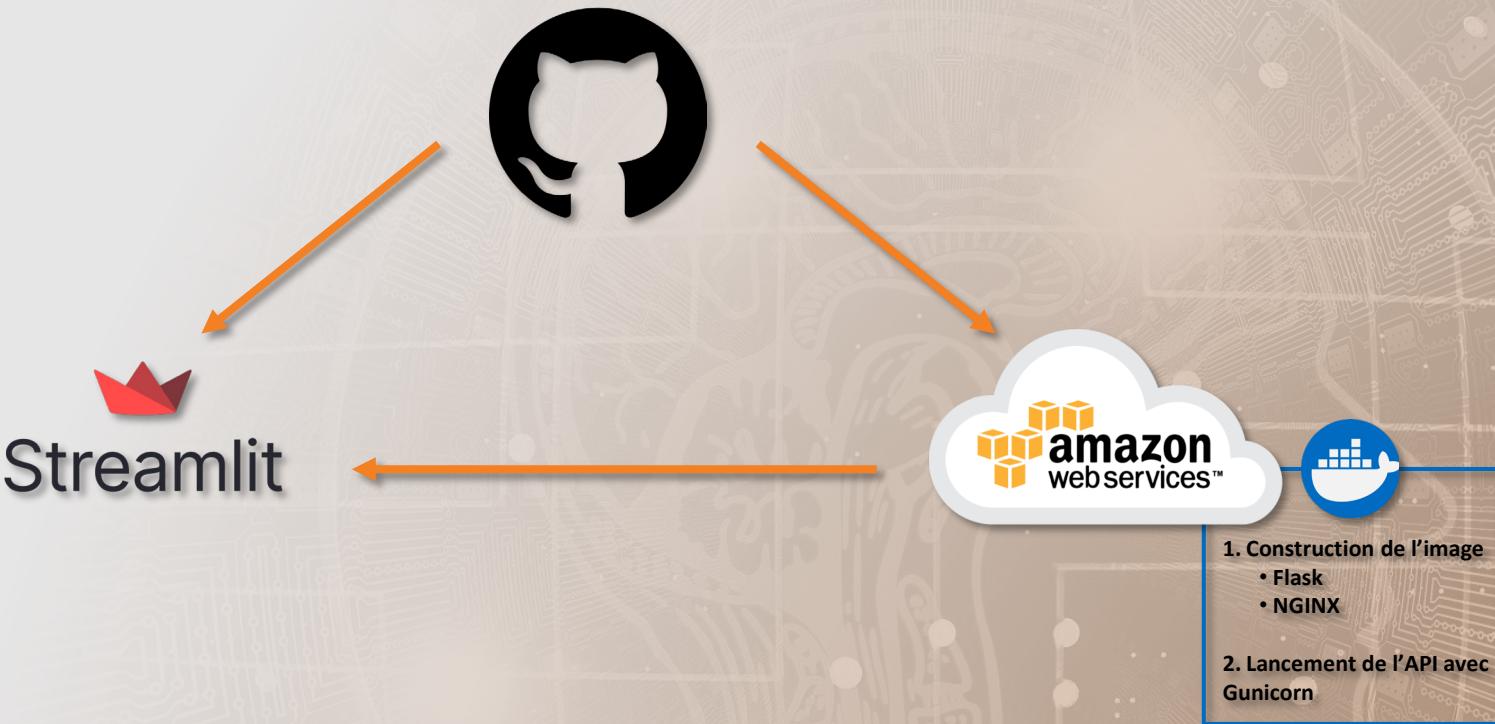
Point_entree_API.py



Frontend



Déploiement continu de l'API



Nous avons utilisé Git et GitHub pour mettre en place un déploiement continu.

Le code du point d'entrée de l'API est directement intégré à Streamlit, tandis que le code de l'API Flask a été déployé sur AWS à l'aide de Docker. Notre objectif était de pouvoir exécuter l'API dans un environnement de production via NGINX et Gunicorn.

L'interface utilisateur côté frontend est ainsi connectée au code côté backend via l'adresse IP du serveur EC2 d'AWS.



Démonstration de l'API

Le modèle sélectionné est disponible pour l'application en local.

Cependant, l'instance EC2 gratuite d'AWS ne permettait pas d'héberger le modèle.

Par conséquent, nous avons utilisé la régression logistique avec TF-IDF pour faire la démonstration de l'API en production.

API pour prédire les Tags d'une question StackOverFlow

Entrez le titre de votre question ici:

Entrez le corps de votre question ici:

Prédiction

Lien pour accéder à l'API :

<https://categoriser-automatiquement-des-questions.streamlit.app/>



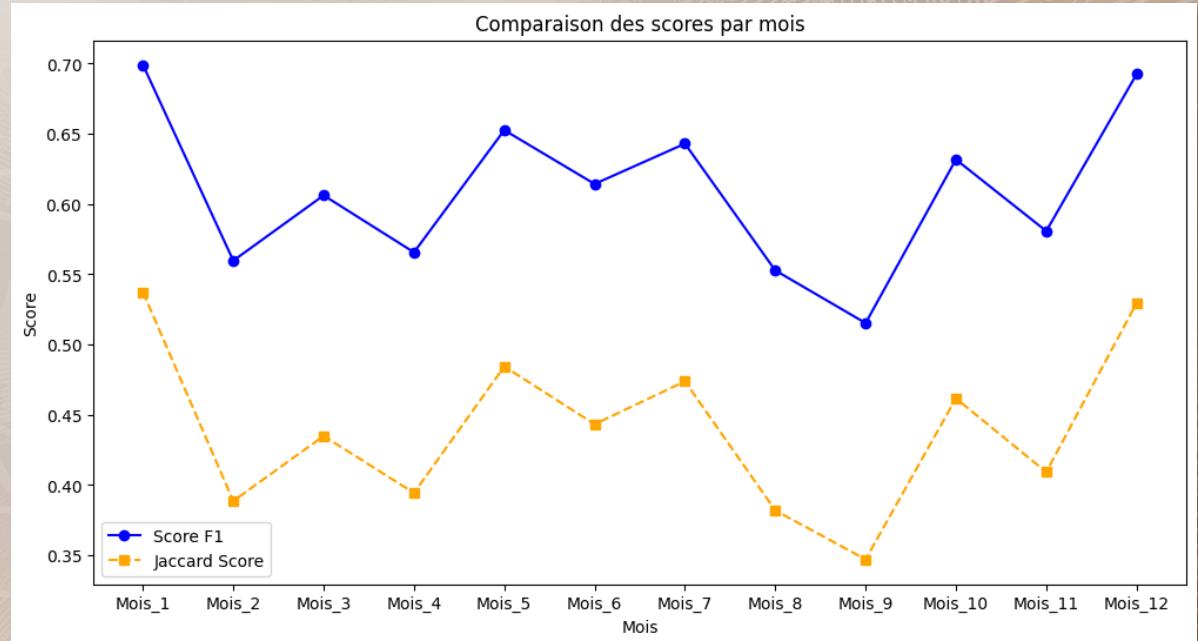
Conclusion

Stabilité du modèle sur un an

Enregistrement des runs dans MLFlow Tracking :

Table Chart Evaluation Experimental

	Run Name	Metrics				
		Created	Duration	f1_score	jaccard_score	
	Mois_12	18 hours ago	2.7s	0.69230769...	0.52941176...	
	Mois_11	18 hours ago	2.5s	0.58064516...	0.40909090...	
	Mois_10	18 hours ago	2.5s	0.63157894...	0.46153846...	
	Mois_9	18 hours ago	2.6s	0.51515151...	0.34693877...	
	Mois_8	18 hours ago	2.6s	0.55263157...	0.38181818...	
	Mois_7	18 hours ago	2.6s	0.64285714...	0.47368421...	
	Mois_6	18 hours ago	2.4s	0.61417322...	0.44318181...	
	Mois_5	18 hours ago	2.4s	0.65248226...	0.48421052...	
	Mois_4	18 hours ago	2.4s	0.56551724...	0.39423076...	
	Mois_3	18 hours ago	2.4s	0.60606060...	0.43478260...	
	Mois_2	18 hours ago	2.5s	0.55958549...	0.38848920...	
	Mois_1	18 hours ago	3.2s	0.69892473...	0.53719008...	



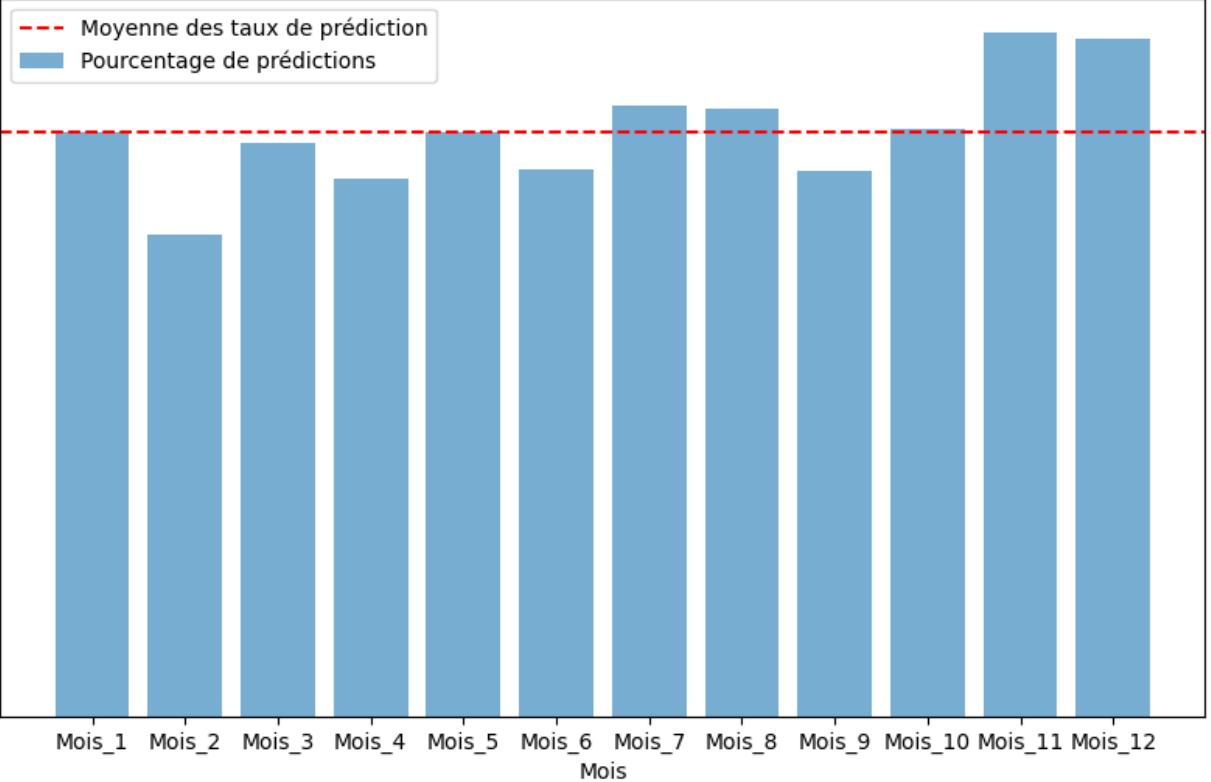
Il est observé que le modèle ne diminue pas en termes de score au fil du temps. Cependant, il est moins précis pour les mois 2, 4 et 9.



Conclusion

Stabilité du modèle sur un an

Comparaison des pourcentage de prédition réalisés par le modèle au fil des mois



En examinant de plus près les mois où le modèle est moins précis, il apparaît que cela est dû au fait que le modèle prédit beaucoup moins de tags.

Il semble donc que le modèle soit confronté à des questions pour lesquelles il ne connaît pas les tags associés.

Pour résoudre ce problème, il serait nécessaire de déterminer quels sont ces tags manquants et éventuellement de ré-entraîner le modèle en tenant compte de ces tags supplémentaires.