Projet 5

Catégorisez automatiquement des questions

Plan de présentation

1. Présentation de problématique
2. Cleaning
3. Feature engineering
   - BOW
   - TF-IDF
   - Word2vect
4. Exploration
   - Analyse exploratoire
   - Réduction de dimension
5. Modélisation non-supervisée
   - LDA
   - Clustering
   - Approche basée sur les fréquences
6. Modélisation supervisée
7. Conclusion

1. Présentation de problématique

- Stack Overflow est une des plus grandes plateformes de questions – réponses pour les développeurs

- Les utilisateurs peuvent poser des questions relatives au développement de code, mais souvent il faut mieux chercher dans les questions qui existent déjà

➡ **Aujourd'hui, la BDD contient :**
  - **47 millions de questions dont**
  - **4 millions de questions qui datent de 2019.**

- Pour faciliter la classification des questions et la recherche, les utilisateurs sont censés de saisir jusqu'à 5 tags => mots clés qui caractérisent la question

- Les tags sont un texte libre (système d'aide automatique existe, mais pas de restriction sur le chaîne de caractères saisie)

1. Présentation de problématique

- **But de projet :**
  - Trouver un algorithme de machine learning qui aidera l'utilisateur à saisir les tags appropriés en faisant une prédiction de tags à partir de données texte renseignées

- **Intérêt de projet :**
  - Mieux classifier des questions
  - Proposer des questions déjà existantes plus pertinentes
  - Eviter des questions en doublons
  - Fidéliser les utilisateurs en proposant un fonctionnement simple, rapide et efficace

- **Données :**
  - Analyses descriptives + modélisation : 50 000 questions de 2019
  - Création d'un modèle word2vect : 500 000 questions de 2019

2. Cleaning

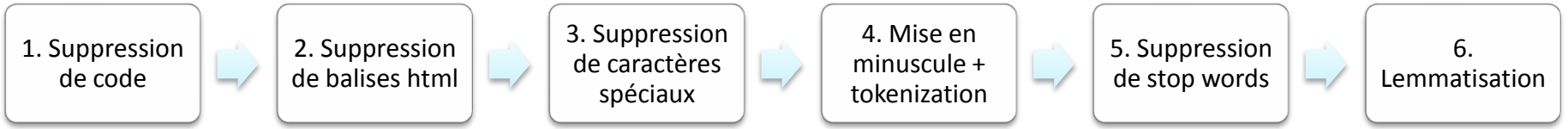
Features :

- Chaque feature est composées de titre et corps de question
- Exemple de corps :

```
<p>I want to use an executor to run the two controllers. The reader controller will need to be invoked every X
minutes (and X is greater than the time it takes the controller to run).</p>

<p>Right now, I am creating a list of <code>Callables</code>, sending them to an ExecutorService that is:</p>

<pre><code>List&lt;Future&lt;Void&gt;&gt; futures = ExecutorService es = new Executors.newFixedThreadPoll(2);
for(Future&lt;Void&gt; future: futures) {
    try {
        future.get();
    } catch (Exception e) {
        // log the error
    }
}
</code></pre>
```

- Particularités :
  - Contient du code
  - Contient des balises HTML

- Workflow de cleaning :



2. Cleaning

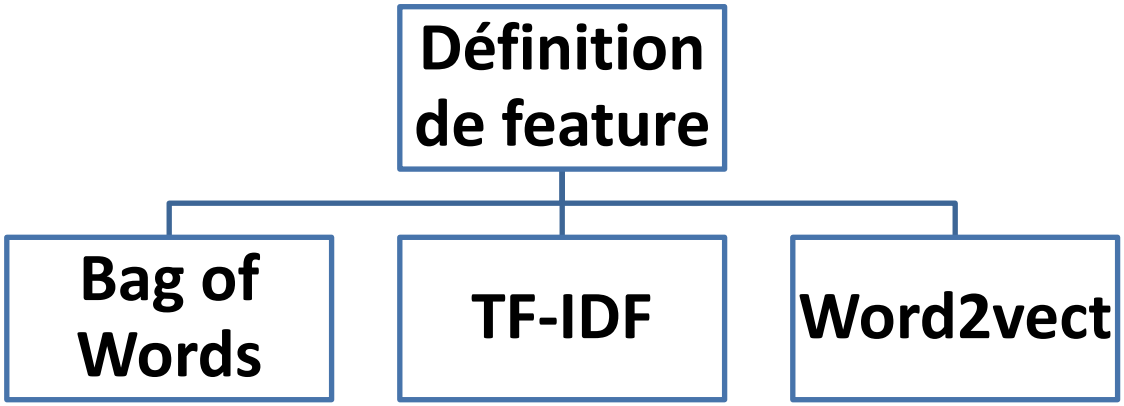
Cible :

- Composé de plusieurs tags séparés par <>
- Si le tag contient plusieurs mots, ceux-ci sont séparés par « - »
- Exemple de cible :

```
'<java><executorservice><java-threads>'
```

=> Le cleaning consiste en suppression de <> et remplacement par l'espace

3. Features engineering Workflow



Plusieurs façons de **définition de feature** ont été testés :

- Titre uniquement
- Titre + corps
- Titre + corps + bigrams titre + bigrams corps
- Titre pondéré + corps
- Titre pondéré + bigrams titre + corps

3. Features engineering
Bag of Words

- Méthode utilisée pour créer une matrice creuse à partir d'un corpus de texte

- Dimensions de la matrice :
    - *n* lignes, où *n* représente le nombre de questions
    - *k* colonnes, où *k* représente le nombre de n-grams de corpus

- Les éléments de la matrice représentent le comptage de nombre de fois le mot apparaît dans la question

- Création BOW de cible : matrice binaire, car chaque tag n'est présent qu'une fois dans chaque question => avantage pour la modélisation (par exemple pour appliquer une régression logistique)

3. Features engineering
TF-IDF

- **TF- IDF** = Term-Frequency – Inverse Document Frequency

- Equivalent de la méthode Bag of Words

- Les mêmes dimensions de la matrice :
  - *n* lignes, où *n* représente le nombre de questions
  - *k* colonnes, où *k* représente le nombre de n-grams de corpus

- **Différence :**
  - les termes de la matrice sont calculés comme fréquence pondérée :
  - Fréquence de terme au sein d'un document / fréquence inversée de terme dans le corpus

3. Features engineering
Word2vect

- Méthode de word embedding (plongement de mots) pour créer une matrice dense

- Les mots sont représentés par vecteurs => possibilité de calculer les distances, les similarités entre les mots, ainsi que des opérations arithmétiques

- Il est possible d'utiliser des dictionnaires qui existent déjà et qui sont disponible en ligne, par exemple des vecteurs pré-entraîné sur Google News dataset (100 billion mots)

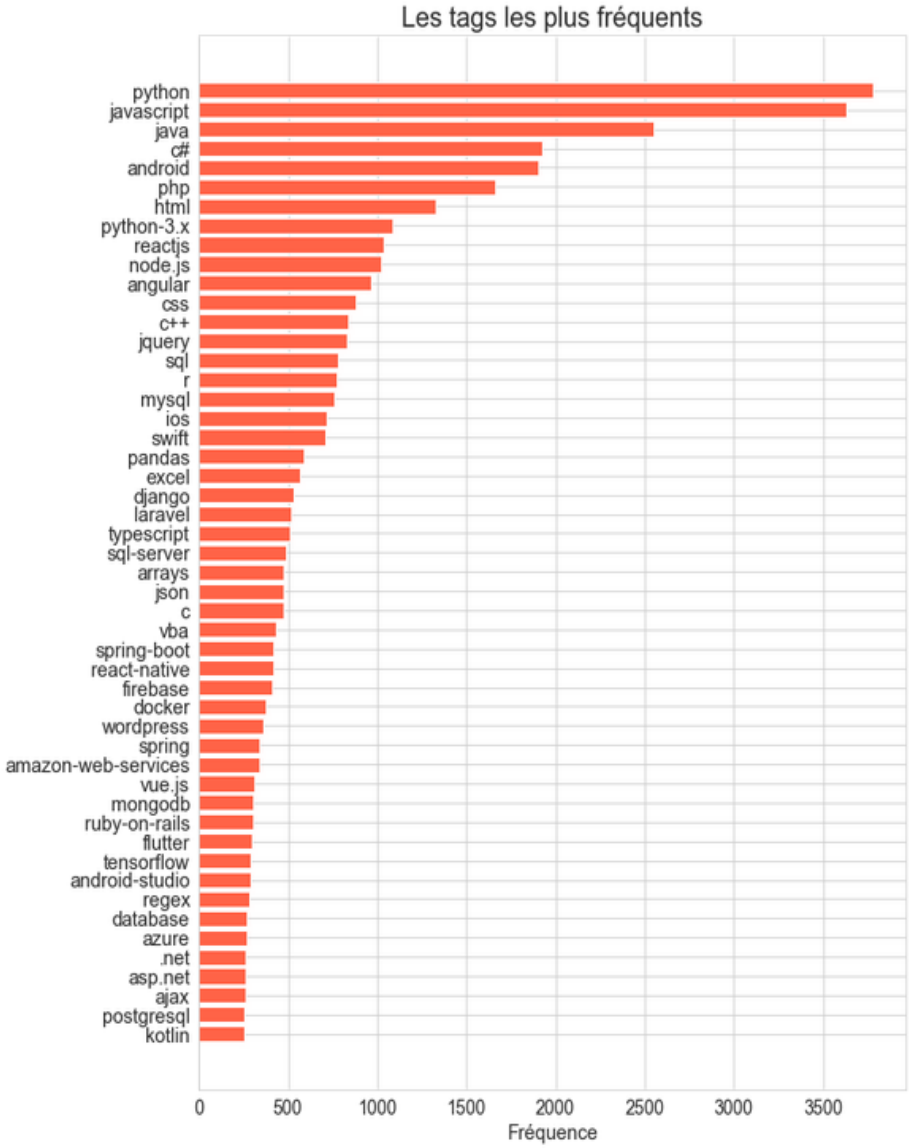
- Etant donné que notre problématique est spécifique dans le domaine d'informatique, j'ai créé mon propre modèle à partir de 500 000 questions publiées en 2019

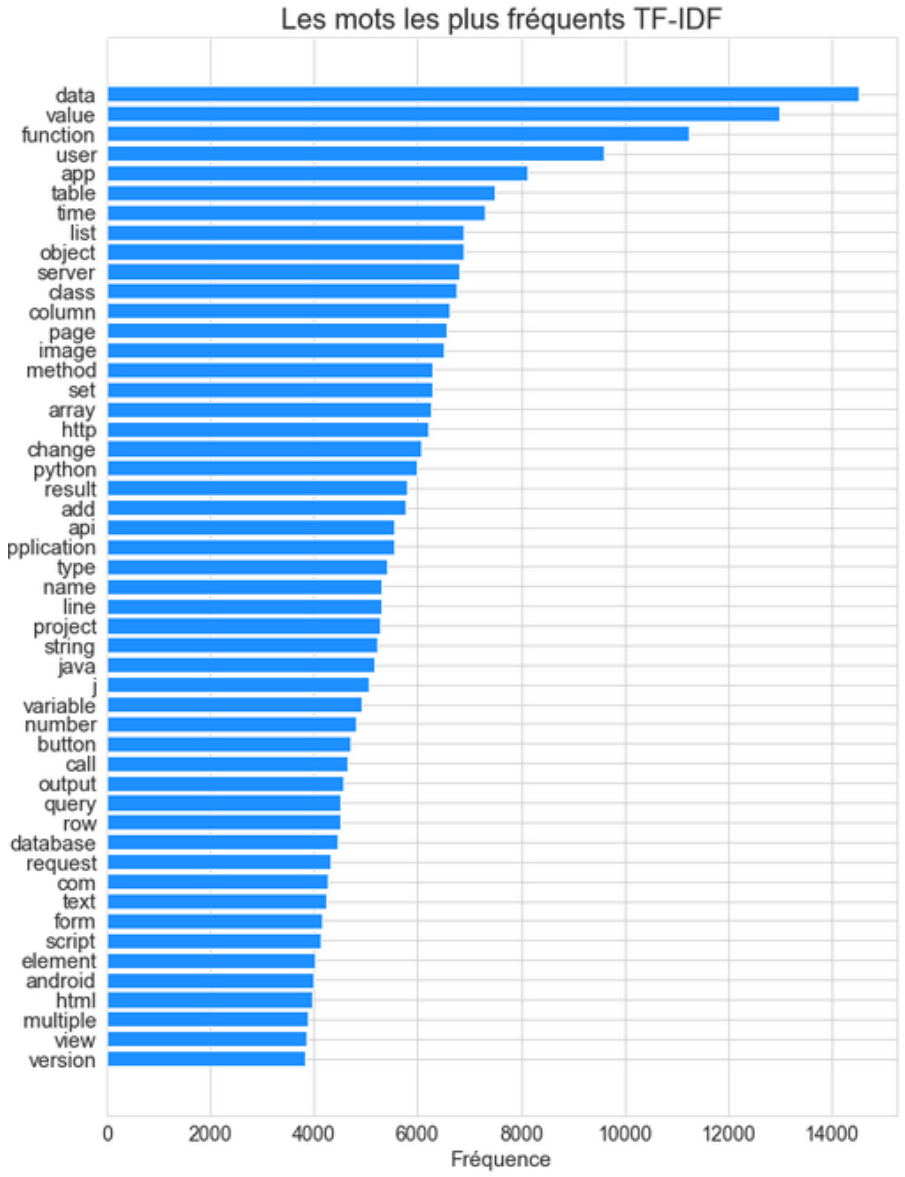
- 2 modèles ont été testés : vecteurs de taille 300 et 400

- Exemple de tests effectués sur le modèle :

```
Entrée [327]: model1.doesnt_match("python javascript jupyter anaconda".split())

Out[327]: 'javascript'

Entrée [331]: model1.most_similar("python")

Out[331]: [('py', 0.5850811004638672),
           ('anaconda', 0.5812091827392578),
           ('miniconda', 0.5759146213531494),
           ('venv', 0.5366100072860718),
           ('pypy', 0.5239589214324951),
           ('pyenv', 0.49613383412361145),
           ('envs', 0.493344247341156),
           ('virtualenvs', 0.47054028511047363),
           ('pycharmprojects', 0.4705374836921692),
           ('pycharm', 0.4689590036869049)]
```

4. Exploration
Analyse exploratoire

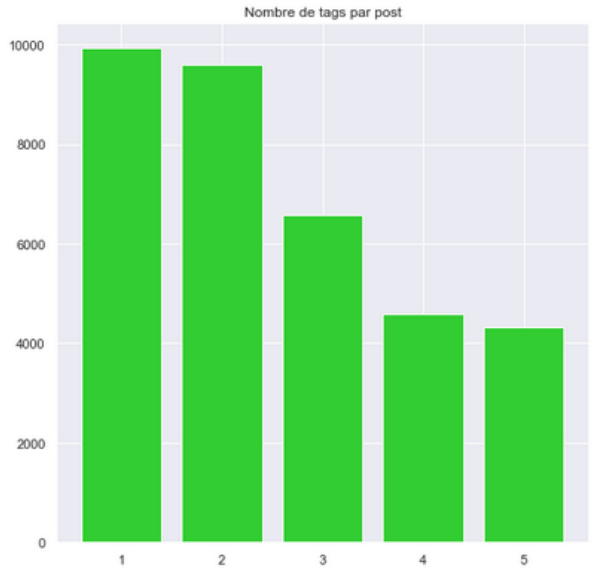




4. Exploration
Analyse exploratoire - cible

Nombre de tags par post :
- min = 1 / max = 5
- médian = 3
- moyen = 3

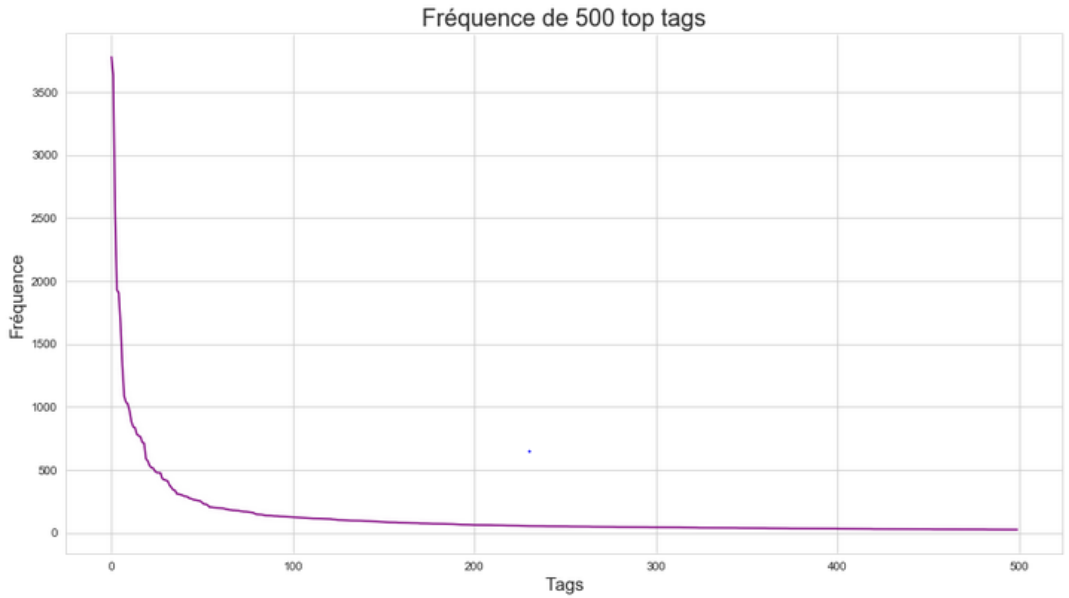
Nombre de labels distinctes :
- 11,5 k tags



Les tags les plus fréquents :

	Word	Frequence	Pourcentage	Pourc cum
3511	python	3777	4.021508	4.021508
2136	javascript	3632	3.867121	7.888629
2113	java	2548	2.712947	10.601576
816	c#	1927	2.051746	12.653322
164	android	1905	2.028322	14.681644

La première expression est présente 3777 fois, ce qui représente 4 % de posts.



Les tags les moins fréquents :

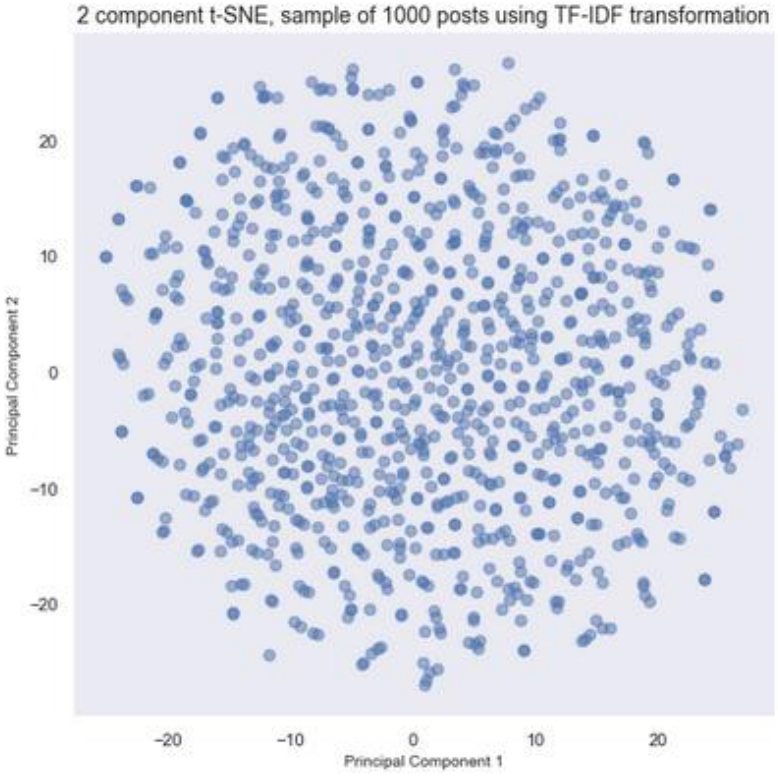
	Word	Frequence	Pourcentage	Pourc cum
413	appsettings	2	0.002129	99.991482
414	appsflyer	2	0.002129	99.993612
415	appveyor	2	0.002129	99.995741
848	callgrind	2	0.002129	99.997871
313	angular-ng-if	2	0.002129	100.000000

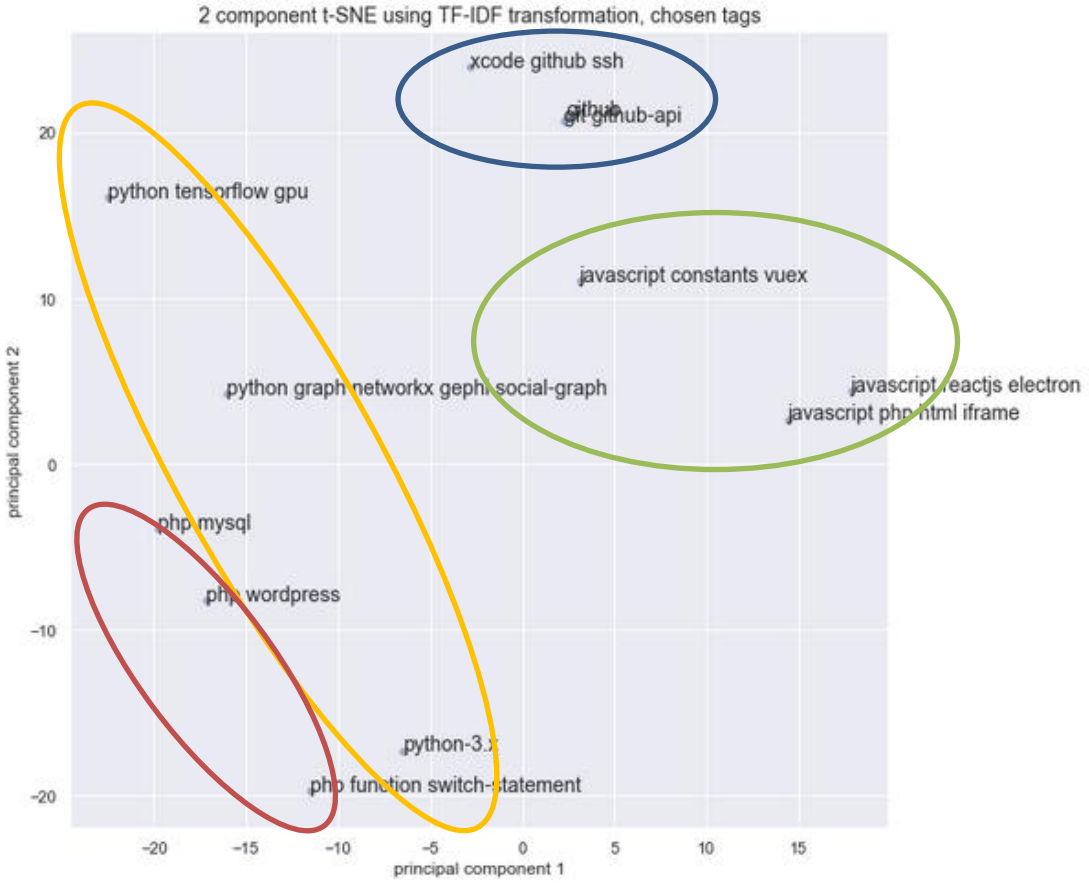
Les dernières expressions sont présentes seulement 2 fois.

Le défi : Réduire le nombre de tags
- ➢LDA
- ➢Clustering

4. Exploration
Réduction de dimension

- Réduction de dimension d'un échantillon de 1000 questions, matrice TF-IDF
- Méthodes testées : ACP et t-SNE
- Affichage d'un sous-ensemble de tags contenant mot spécifiques: python, php, github, javascript

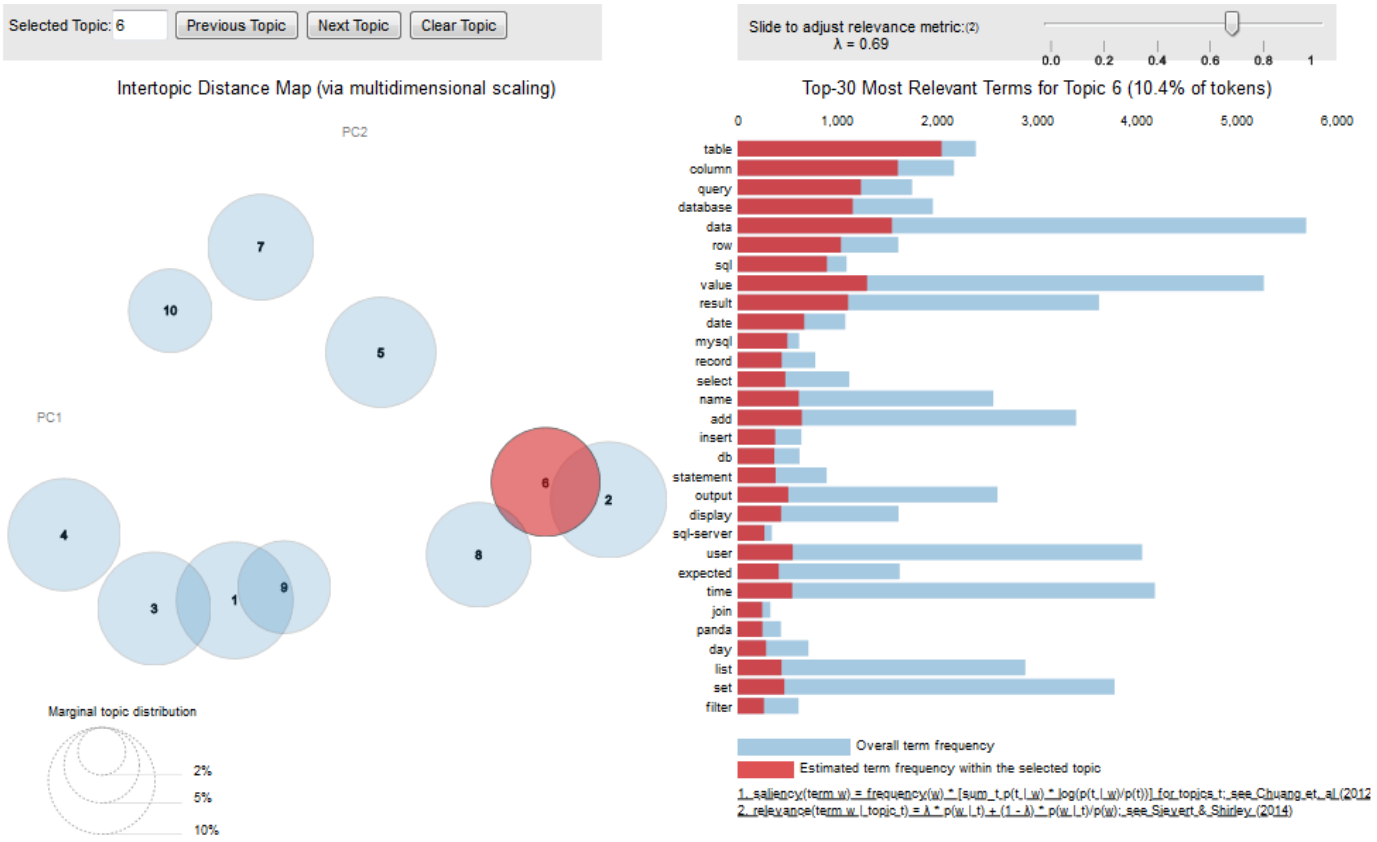




5. Modélisation non-supervisée
Latent Dirichlet Allocation

LDA est une méthode non-supervisée générative qui nous permets d'évaluer un nombre donné de thèmes présents dans le corpus et de retourner les mots clés les plus pertinents pour chaque sujet

- **Objectif :** Récupérer les thèmes potentielles pour reformuler les tags et réduire leur nombre
- **Désavantage de la méthode** : Difficile de trouver des thème précis, distincts et interprétable.
- **Workflow :** Test de plusieurs types de définition de features (BOW, TF-IDF avec titre, titre pondéré, bigrams etc.), puis tuning de paramètres de modèle
- **Interprétation de sujets :** dans le « meilleur » modèle – visualisation interactive à l'aide de LDAvis :
- **Conclusion :** Certains sujets sont interprétable, d'autre non. Difficile à utiliser pour reformuler la cible

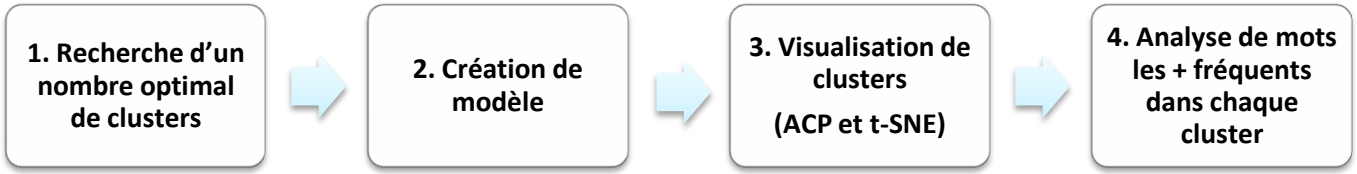


5. Modélisation non-supervisée
Clustering

- **Méthode utilisée:** Kmeans, données transformées par TF-IDF

- **But :** Trouver des clusters avec des questions similaires, déterminer les sujets et réduire le nombre de tags

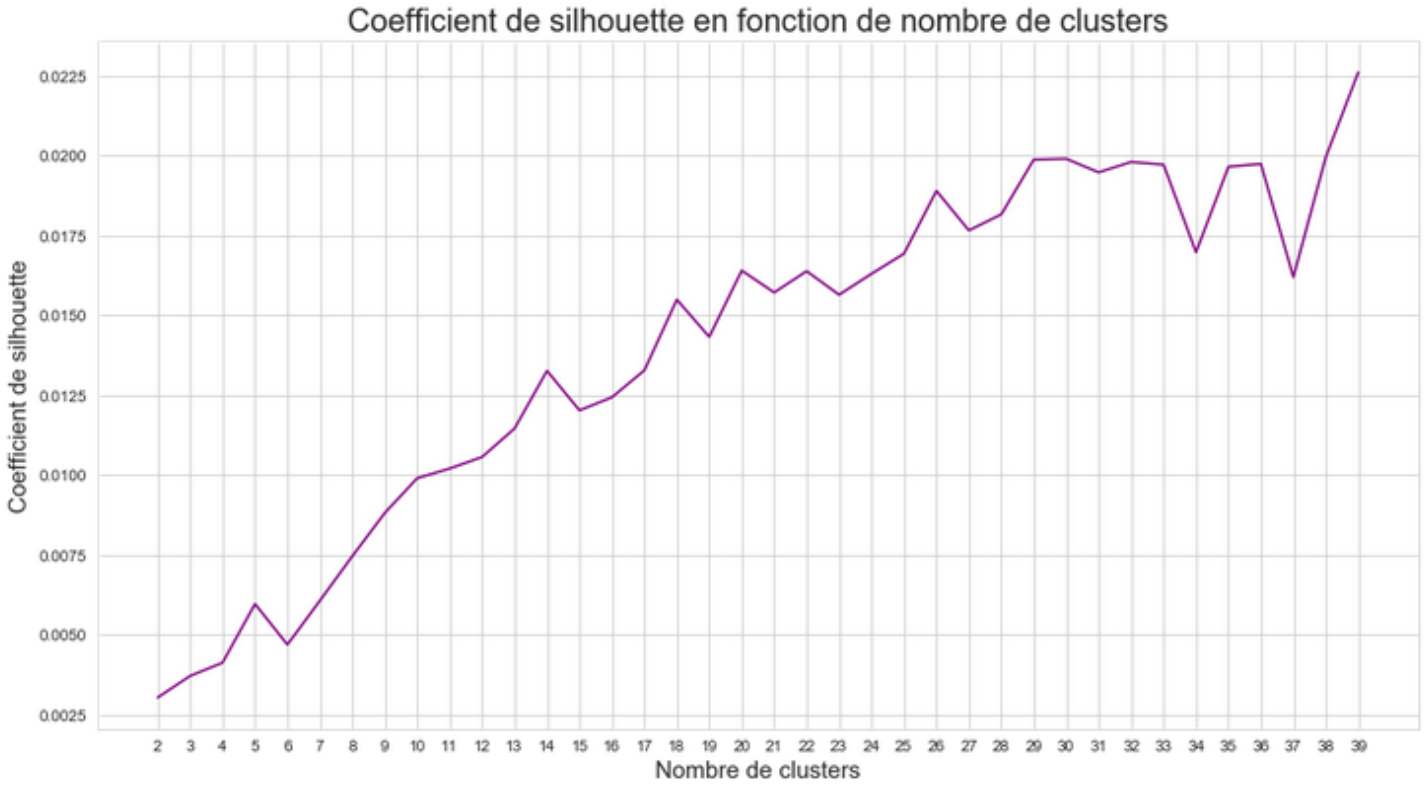
- **Workflow :**



- Fonction utilisée dans sci-kit learn : MiniBatchKMeans

5. Modélisation non-supervisée Clustering

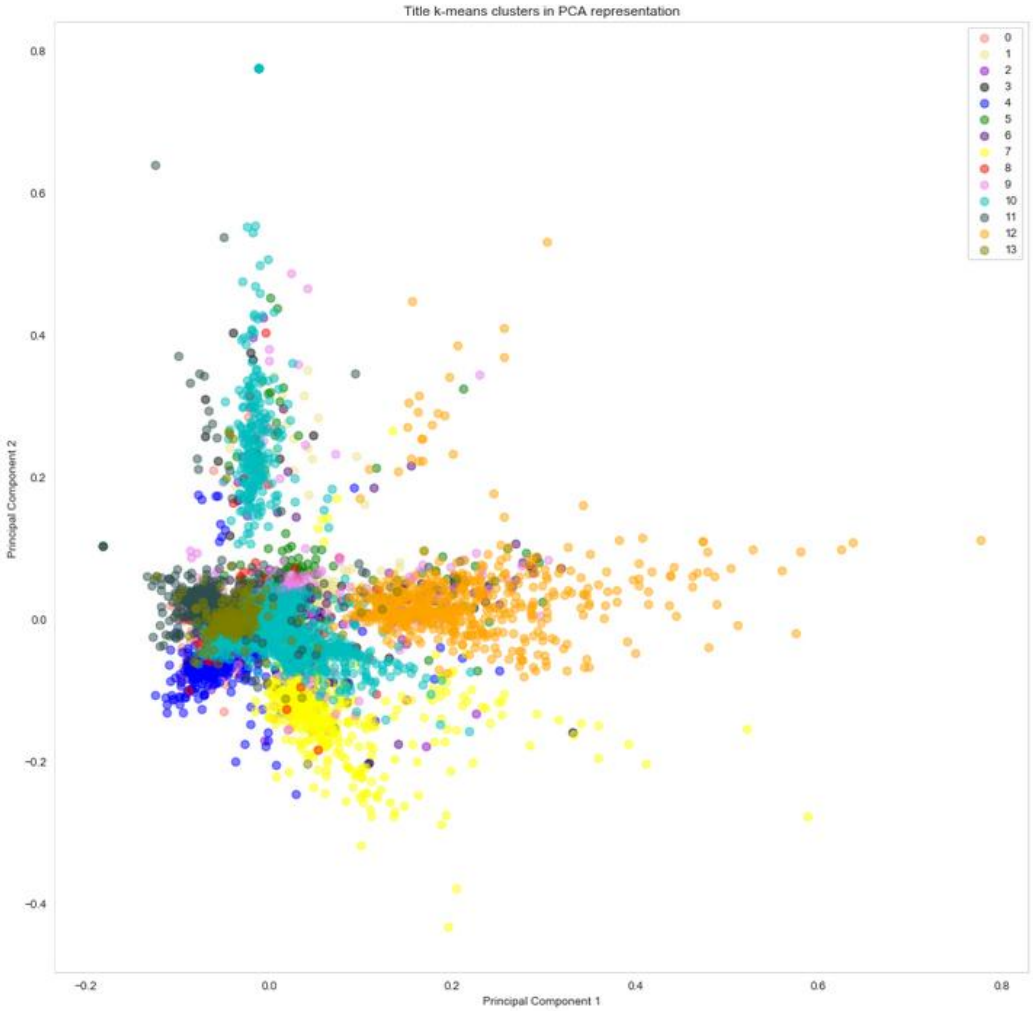
Résultats pour le titre codé avec TF-IDF :



Coefficient de silhouette est faible, mais augmente avec le nombre de clusters. Nous avons choisi un nombre « raisonnable » de clusters qui forme un pic sur la courbe : 14 clusters

5. Modélisation non-supervisée
Clustering

Visualisation à l'aide de ACP:



Exemple de mots clés:

```
Cluster7:
method number send failed insert swift creating selenium ui calling
```

```
Cluster10:
value data function python array object column multiple string table
```

```
Cluster12:
api java j add php name date result version within
```

Conclusion : Les clusters ne sont pas très nets et difficiles à interpréter.

5. Modélisation non-supervisée
Approche basée sur les fréquences

Méthode inspirée par un approche non-supervisée proposée par Ha-Cohen-Kerner en 2003

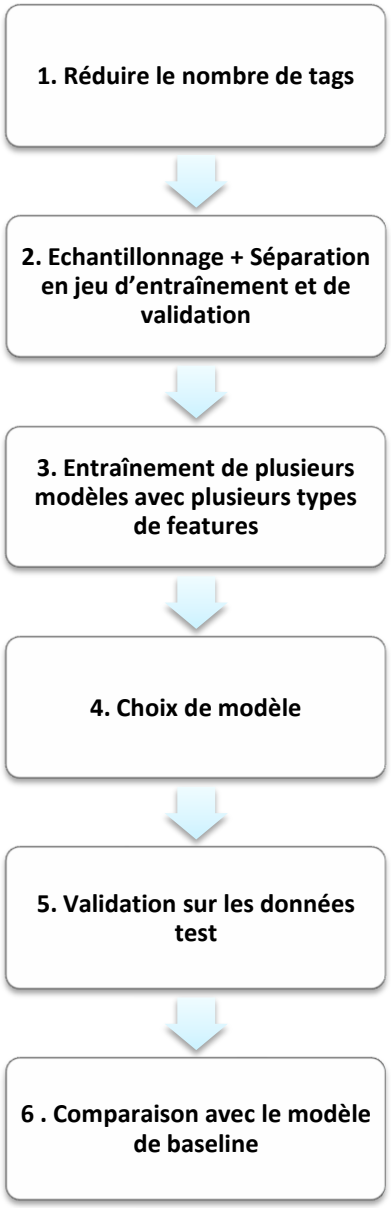
- **Principe de la méthode :**
  1. Créer un dictionnaire de tags
  2. Créer un dictionnaire d'expressions de questions composée par titre, corps et leur bigrams
  3. Comparer les expressions communes au dictionnaire de tags et au dictionnaire de questions
  4. Calculer les fréquences TF IDF d'expressions de la question et classer par ordre décroissant
  5. Sortir 3 tags les plus fréquents

- **Avantages :**
  - Rapidité de calcul
  - Propose au moins 1 tags dans 99,99 % de cas

- **Inconvénients :**
  - Ne fait aucune estimation de tags qui ne figurent pas parmi les expressions écrites par utilisateur

- **Conclusion :**
  - Modèle est simple et donne des résultat intéressants. Utilisation comme modèle de « baseline » pour comparer avec la modélisation supervisée

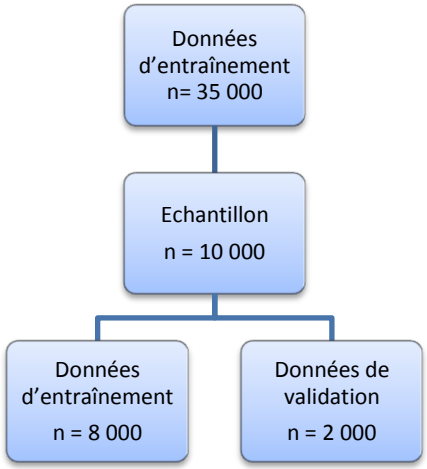
6. Modélisation supervisée



1. Réduction de nombre de tags

- Pas de méthode appropriée parmi les modèles non-supervisés testés => réduction de nombre de tags par rapport à leur fréquences.
- Suppression de tags avec moins de 0.1 % d'effectif
- Top tags = 130 tags les plus fréquents
- Après la suppression 15 % de question se retrouve sans aucun tag

2. Données pour choisir le modèle



6. Modélisation supervisée

3. Entraînement de plusieurs modèles avec plusieurs types de features

- Feature = titre pondéré + corps
  - Bag of words
  - TF-IDF
  - Word2vect (2 modèles différents)
- Cible = matrice binaire de 130 labels
- Modèles de classification multilabel
  - Régression logistique
  - KNN
  - Arbres de décision
  - Forêts aléatoires

4. Choix de modèle

- Critère de choix :
  - **Score F1 micro avg**, qui calcule le score en se basant sur le nombre de TP et FN et FP global
  - Score F1 est une moyenne harmonique de précision et recall => prend en considération les tags prédits par erreur ET les tags qui ne sont pas prédits par erreur

- Le meilleur modèle :
  - **Régression logistique avec TF-IDF**, F1 = 0.47, Précision = 0.75, Recall = 0.35

6. Modélisation supervisée

5. Validation sur les données test

- Le jeu de données test (15 000 questions / 50 000) mise de côté tout au début

- **Résultat :**
  - F1 = 0.56
  - Précision = 0.76
  - Recall = 0.44

6. Modélisation supervisée

6. Comparaison avec le modèle de baseline

	Critère	Modèle de baseline	Modèle supervisé
I.	Temps d'exécution	0.15 sec	0.06 sec
II.	Nombre de questions avec au moins 1 prédiction	99.99 %	59.46 %
III.	Nombre de questions avec au moins 1 prédiction correcte	36.89 %	50.41 %

Prédiction de tags pour une question test aléatoire :

Titre :

```
IPV4 traffic not working with AWS egress only internet gateway
```

Corps :

```
<p>I have assigned an egress only internet gateway to my private subnet. Now I can connect with IPV6 websites bu
t not with IPV4 addresses.</p>

<p>Do I need NAT gateways to access IPV4 address from my EC2 machine? (Only outgoing traffic)</p>
```

Tags prédits modèle de baseline :

```
private
access
gateway
--- 0.15600037574768066 seconds ---
```

Tags prédits modèle supervisé :

```
amazon-web-services
--- 0.06240034103393555 seconds ---
```

7. Présentation d'API

2 Saisir le texte

Entrée [*]:
```
# User's title input
title = input("Title: ")
```

Title:

Entrée [*]:
```
# User's body input
body = input("Body: ")
```

3 Retourner les tags prédits

Entrée [*]:
```
# Cleaning
body_clean = post_to_words(body)
title_clean = post_to_words(title)
post_w = (title_clean + " ") *3 + " " + body_clean
post_w = Series(post_w)

# Formatting
new_question_final = tf_idf_final.transform(post_w)
new_question_final = new_question_final.toarray()

# Fit the sample
predicted_tags_new_quest = model_final.predict(new_question_final)

# Print the tags
for freq, word in zip(predicted_tags_new_quest[0], y_train_final_vocab):
    if freq > 0:
        print(word)
```

8. Conclusion

Afin de prédire les tags à partir de données texte, la régression logistique est le meilleur modèle parmi les algorithmes testés

- **Avantages:**
  - Rapidité de prédiction,
  - Précision

- **Désavantage :**
  - Le modèle donne des prédictions positifs dans seulement 60 % de cas

- **Améliorations proposées:**
  - Réduire le seuil de prédiction positive pour obtenir plus de tags prédits
  - Créer un modèle hybride qui prend en compte et les tags prédit par le modèle supervisé et le modèle de baseline

9. Liens & ressources

- Le code de projet, les documents, ainsi que le code final d'API est accessible ici : https://github.com/Lenka-St/Projet5

- **Ressources :**
  - https://www.kaggle.com/c/word2vec-nlp-tutorial#part-1-for-beginners-bag-of-words
  - http://www.nltk.org/book/ch01.html
  - https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24
  - https://towardsdatascience.com/using-word2vec-to-analyze-news-headlines-and-predict-article-success-cdeda5f14751
  - https://www.kaggle.com/jbencina/clustering-documents-with-tfidf-and-kmeans
  - https://www.kdnuggets.com/2019/09/overview-topics-extraction-python-latent-dirichlet-allocation.html
  - https://medium.com/datadriveninvestor/predicting-tags-for-the-questions-in-stack-overflow-29438367261e
  - https://towardsdatascience.com/improving-the-stack-overflow-search-algorithm-using-semantic-search-and-nlp-a23e20091d4c