Projet 5 OC : Catégorisez automatiquement des questions

Présenté par : NAMA NYAM Guy Anthony

Mentor : Julien Hendrick

22 Mars 2020

OPENCLASSROOMS

Sommaire

Mise en situation

- Stack Overflow est un site de questions-réponses liées au développement informatique.
- Pour poser une question sur ce site, il faut entrer plusieurs étiquettes liées la question.
- Pour les utilisateurs débutants, ils ne connaissent pas toujours les étiquettes associées à leur question.
- Il serait donc judicieux de développer un système qui assigne automatiquement plusieurs tags pertinents à une question.
- A côté de celà, il serait tout aussi pertinent de proposer les mots clés à ses posts.

Objectif

Développer un système de **suggestion de tag** pour le site.

Approches

- Pour ce faire, nous utilisons les algorithmes de machine learning :
  1. Une approche supervisée pour suggérer les étiquettes ou tags
  2. Une approche non supervisée pour suggérer les mots clés ou keywords.
- L'approche supervisée avec les algorithmes : Catboost, XGboost, LightGBM, Classification Naive Bayésienne, FAISS
- L'approche non supervisée avec l'algorithme : LDA.

Sommaire

Description

- Données importées depuis la plateforme "stackexchange explorer" proposé par Stack Overflow pour l'accès au post.
- Données sélectionnées :
  SELECT * FROM posts WHERE FavoriteCount $>$ ***50***.
- Le nombre d'enregistrement de la table de posts récupérés est de 20332 pour 22 colonnes.

Nettoyage de la donnée

- Des enregistrements retenus, nous avons uniquement gardé les champs **'Body'**, **'Title'**, **'Tags'**
- Aucune valeur manquante répertoriée pour ces 3 champs.
- Suppression des balises html pour le champs 'Body'

```
for i, row in df_tagged.iterrows():
    df_tagged.loc[i,"Body"] = BeautifulSoup(str(row["Body"]),'html.parser').get_text()
```

- Mise en minuscules de l'ensemble des champs.
- Remplacer les abréviations par leur forme longue :

```
paires_all = []
for i, row in df_tagged.iterrows():
    paires = schwartz_hearst.extract_abbreviation_definition_pairs(doc_text=str(row["Body"] + " " + row["Title"]))
    if paires:
        paires_all.append(paires)
paires_all
```

```
[{'lower_case': 'lower case separated by underscore'},
 {'kvo': 'key-value observation'},
 {'ioc': 'inversion of control'},
 {'cs+n': 'cs n =', 'sol 500': 'solution for n=500'},
 {'details': 'different way of handling the url structure i want to use'},
 {'crud': 'create, read, update and delete',
  "dto's": 'data transfer objects',
```

Nettoyage de la donnée

- Remplacer les formes contractées par leur forme longue : utilise un dictionnaire de correspondance.

```
text = re.sub(r"\'m", "am", text)
text = re.sub(r"\'re", "are", text)
text = re.sub(r"\'s", " ", text)
text = re.sub(r"n't", "not", text)
text = re.sub(r"\'ve", "have", text)
text = re.sub(r"\'d", "would", text)
text = re.sub(r"\'ll", "will", text)
```

- Suppression des stop words et ponctuations avec ***Spacy*** ; usage des attributs 'is_stop' et 'is_punct' sur les tokens

- Suppression des caractères *'bizarre'* : regex

```
# Remove special characters and bad symbols
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;-_+]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #]')
def clean_spec_bad(text):
    text = REPLACE_BY_SPACE_RE.sub(' ', text)
    text = BAD_SYMBOLS_RE.sub('', text)
    return text
```

Nettoyage de la donnée

- Suppression des chiffres y compris à l'intérieur des chaines.

```
def remove_numbers_from_string(text):
    document = nlp(text)
    words = []
    for token in document:
        words.append(''.join([i for i in token.text if not i.isdigit()]))
    return ' '.join(words)
```

- Suppression des espaces multiples et les mots de tailles faibles(inférieur à 2)
- Lemmatisation : attribut 'lemma_'

```
# Lemmatization
def lemmatization(text):
    document = nlp(text)
    words = [token.lemma_ for token in document]
    return ' '.join(words)
```

- Suppression des mots non fréquents(apparition moins de 3 fois)

Nettoyage de la donnée

- Visualisation du champs 'Body' après épuration

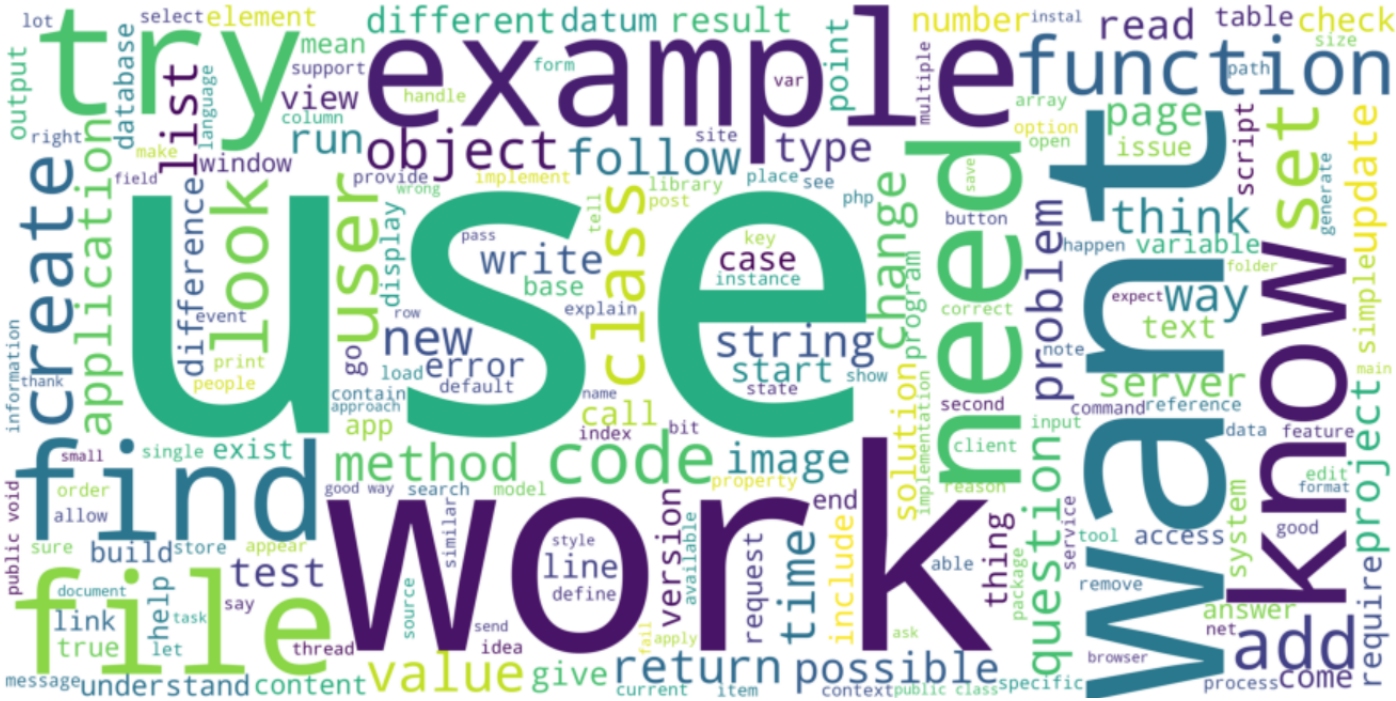
- L'ensemble des traitements textuels précédents étaient pour les suggestions des tags(approche supervisée) et des mots clés(approche non supervisée).

Nettoyage de la donnée

- Un traitement supplémentaire pour la suggestion des keywords.

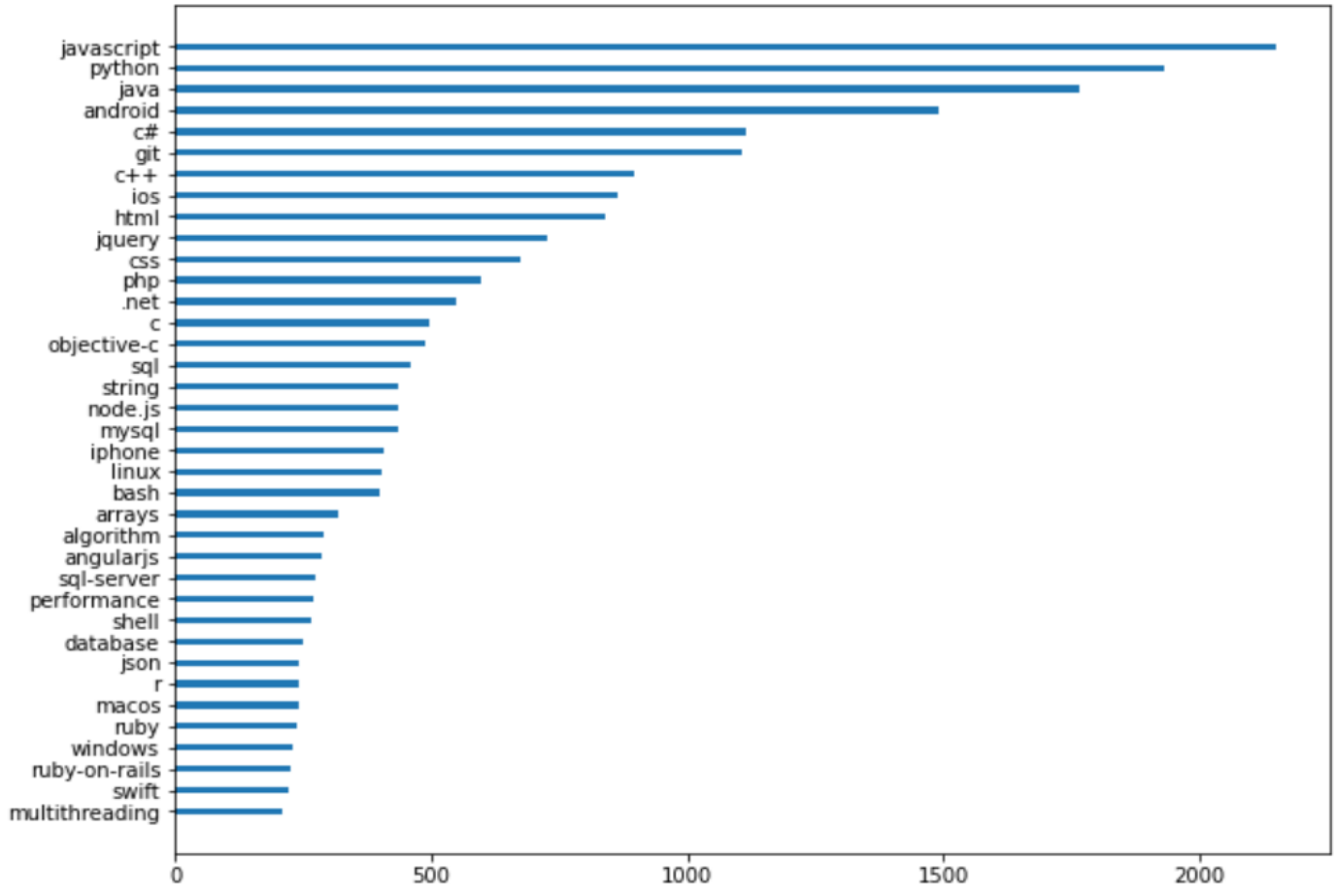
```
def save_potential_keywords(text):
    document = nlp(str(text))
    words = [token.text for token in document if token.pos_ in ['NOUN', 'VERB', 'ADJ']]
    return ' '.join(words)
```

- Visualisation du champs 'Body'



Nettoyage de la donnée

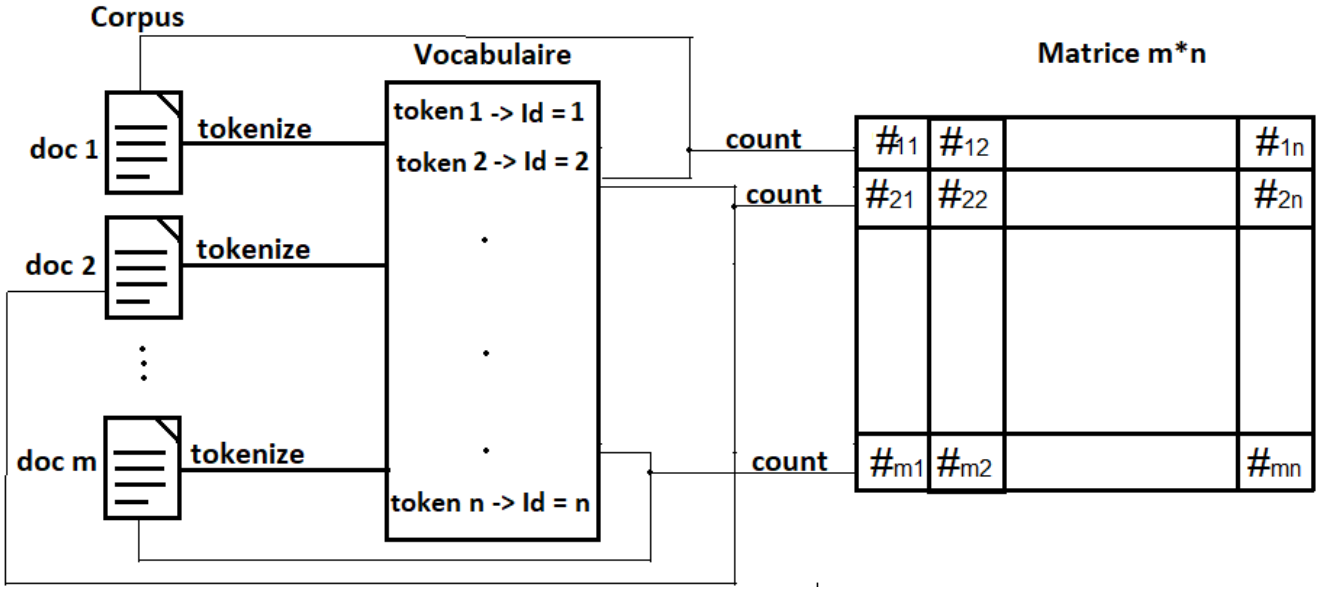
- Le traitement des étiquettes concerne le nombre d'apparition dans notre dataset.
- Nous avons conservé ceux qui apparaissent 200 fois et plus.



Sommaire

Tf

- Comptage de mots



- $\#_{ij}$ représente le nombre d'occurrence du token j dans le document i

```
# Common Vectorizer usage
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()  # Implements both tokenization and occurrence counting
corpus = [
          'Voici le premier document.',
          'Voici le second document.',
          'Et the troixième.'
          ]
  X = vectorizer.fit_transform(corpus)
```

Tf-Idf

- Le but est de réduire l'importance relative de certains mots très présents mais peu significatifs.
- Les occurrences $\#_{ij}$ sont remplacées par $tf\text{-}idf(t_j, d_i)$ normalisée par la norme 2 où :

$$tf\text{-}idf(t,d) = tf(t,d) * idf(t), \text{ où :}$$

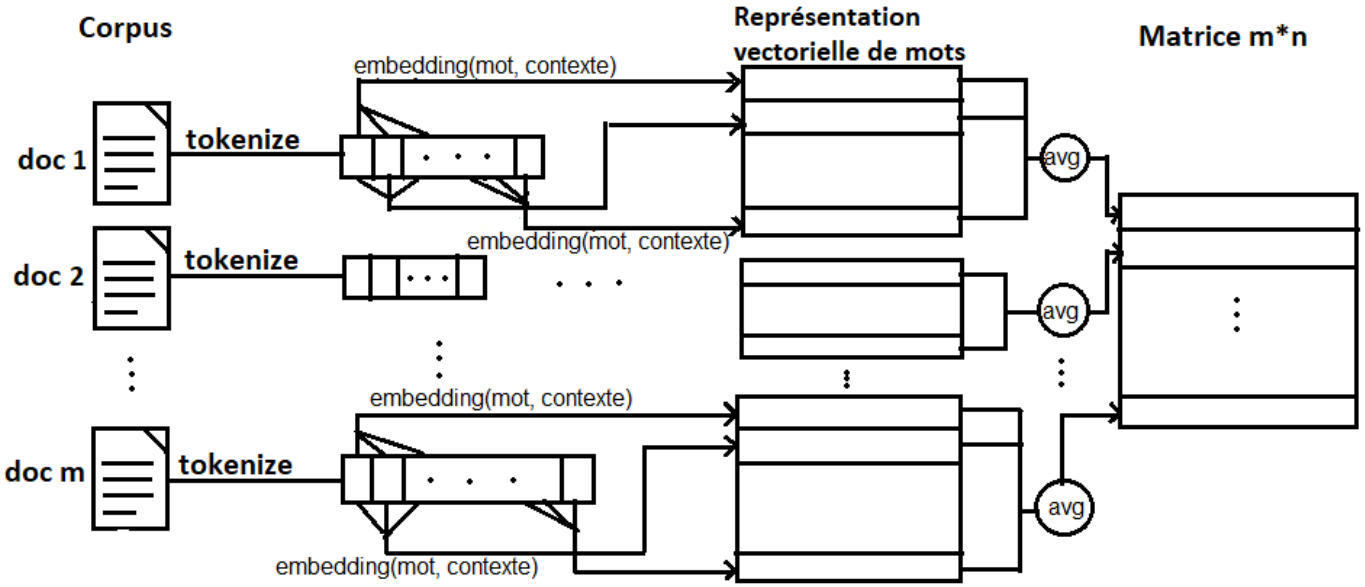
$tf(t,d)$ correspond à la fréquence du terme ou token t dans le document d.

$idf(t) = \log \frac{1+m}{1+df(t)} + 1$, où m est le nombre total de documents et $df(t)$ est le nombre de documents dans lequel apparait le terme ou token t.

- L'utilisation est identique à la précédente transformation en utilisant la classe ***TfidfVectorizer***
- ***Principale limite BOW*** : ignore la dépendance de l'ordre des mots.

Aperçu général

- Plongements de mots.



- *avg* représente la moyenne des représentations vectorielles de mots.
- L'hypothèse principale(***distributional hypothesis***) est de prendre en compte le contexte dans lequel le mot est trouvé.
- Les méthodes de plongements de mots utilisées : Word2vec(gensim), Glove(gensim), DistilBer(HuggingFace) et fastText.

Sommaire

Algorithmes multiclasses

- Nous avons commencé par définir notre métrique d'évaluation.
- La plupart des données à notre disposition contenait 1 à 5 étiquettes.
- Calculer l'exactitude sur les 3 premières étiquettes(médiane).

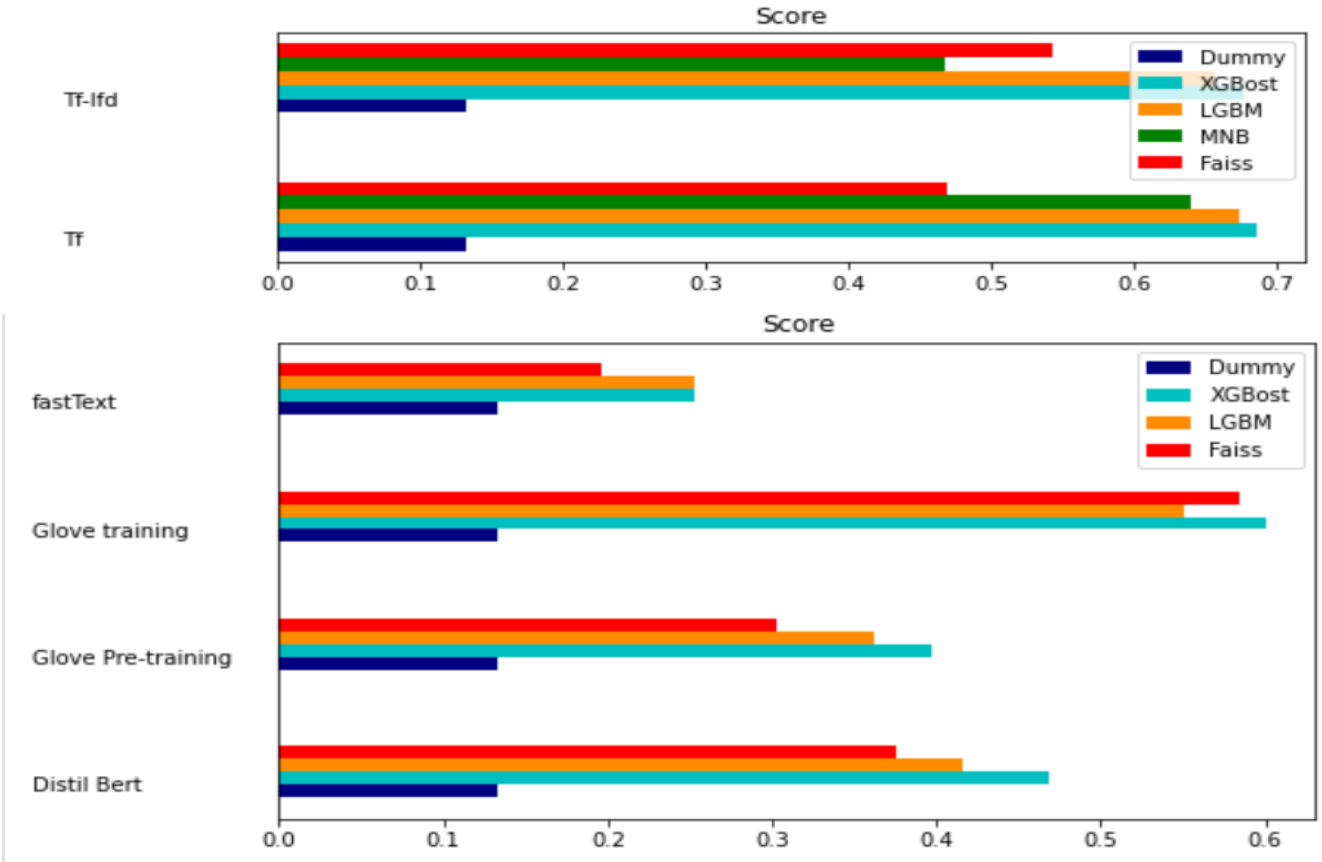
Algorithme 1 Own metric

Require: classifier, X_test, y_true
Ensure: accuracy

```
count = 0
for chaque entrée de X_test do
    Récupérer les 3 meilleurs labels
    if la première étiquette trouvée correspond à la vraie étiquette : count += 1
    if la deuxième étiquette trouvée correspond à la vraie étiquette : count += 0.5
    if la troixième étiquette trouvée correspond à la vraie étiquette : count += 0.25
end for
Retourner count/nombre d'entrée(X_test)
```

Sélection de représentations pour l'hyper-paramétrage

- La sélection se fait sur les modèles suivants : MultinomialNB, Faiss, DummyClassifier, XGBClassifier, LGBMClassifier, CatBoostClassifier.
- L'idée est de conserver la meilleure représentation Bag of words et la meilleure représentation word embedding



Hyper-paramétrage : RandomizedSearchCV

- L'hyper-paramétrage est réalisée sur la représentation Bag of words Tf et Word2vec pour la représentation vectorielles de mots.
- Les algorithmes utilisés disposent dans leur documentation les meilleures paramètres à optimiser(base de notre sélection).

```
# XGBoost
params_xgb = {'gamma': sp_rand(0, 10),
              'max_depth': sp_randint(3, 10),
              'min_child_weight': sp_randint(1, 6)
              }
grid_xgb = RandomizedSearchCV(xgb_model, params_xgb, cv=5, n_iter=10, n_jobs=-1)

# LightGBM
params_lgbm = {'num_leaves':sp_randint(3, 10),
               'max_depth': sp_randint(3, 10)
               }
grid_lgbm = RandomizedSearchCV(lgbm_model, params_lgbm, n_iter=10, cv=5, n_jobs=-1)

# Naive_bayes MultinomialNB
params_lMNB = {'alpha': sp_rand()}
grid_MNB = RandomizedSearchCV(MultinomialNB_model, params_lMNB, n_iter=10, cv=5, n_jobs=-1)
```

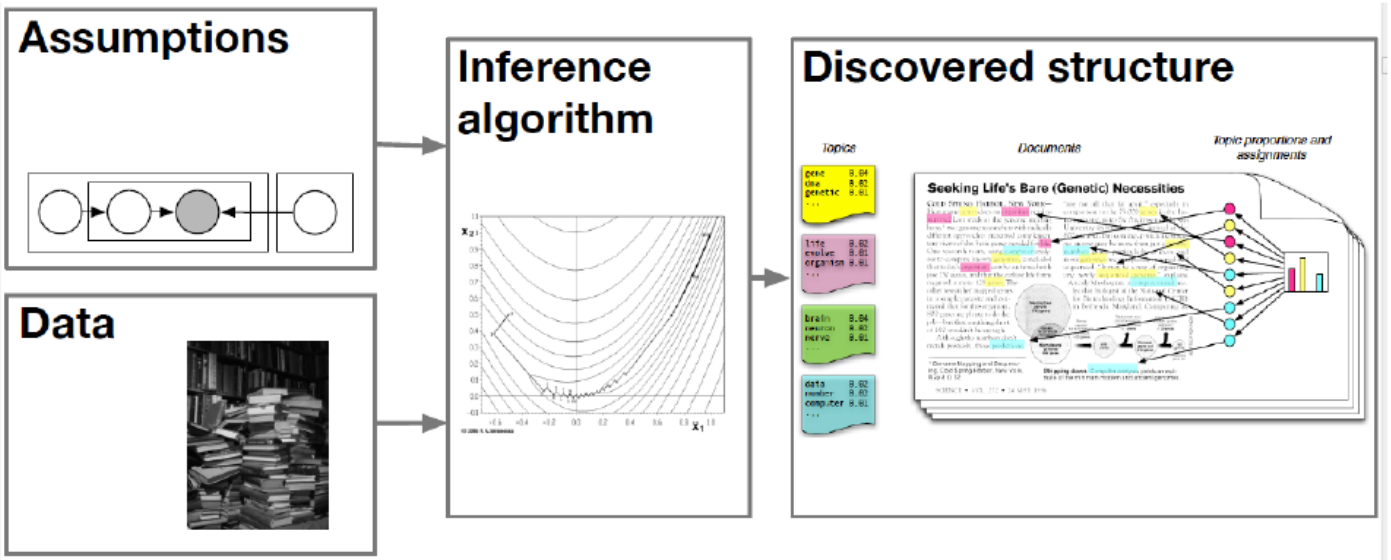
Meilleur modèle

- La meilleure représentation est celle de Tf.
- Le meilleur algorithme est celui de XGBoost
- Le score du meilleur modèle :
- Gain d'environ 0.2(0.685 à 0.707)
- Meilleurs paramètres :
  {'gamma' : 8.407984783100105,
  'max_depth' : 8,
  'min_child_weight' : 3
  }
- Temps d'exécution : 10666s

Sommaire

Algorithme d'apprentissage non supervisé : LDA

- Nous avons utilisé l'algorithme LDA(Latent Dirichlet Allocation) qui est une méthode non supervisée générative pour les mots clés.
- Aperçu général.



- Comme algorithme alternative NMF.

LDA : pratique

- Inférence : déterminer les thèmes, les distributions de chaque mot sur les thèmes, la fréquence d'apparition de chaque thème sur le corpus.
- Pour effectuer l'inférence de ce modèle, nous avons utilisé la librairie LDA de ***gensim***.

```
from gensim.models.ldamodel import LdaModel
from gensim.corpora.dictionary import Dictionary
```

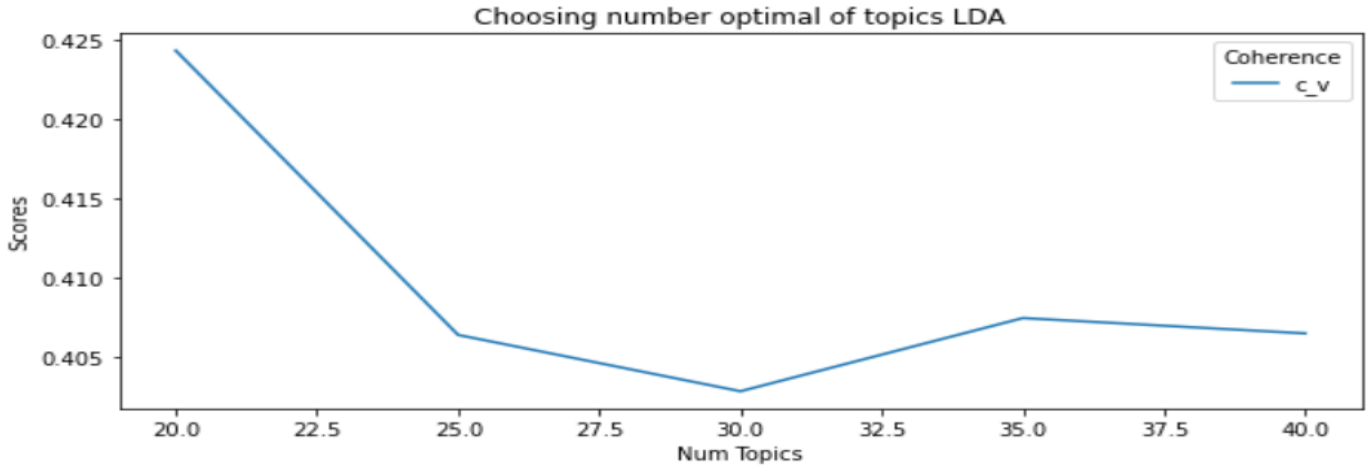
```
corpus = ['Voici le premier doc', 'Voici le deuxieme doc', 'Voilà le troixième']
docs_bow = [doc.split() for doc in corpus]
dictionary = Dictionary(data)
corpus_lda = [dictionary.doc2bow(doc_bow) for doc_bow in docs_bow ]
```

```
n_topics = 20
ldaModel = LdaModel(corpus_lda, num_topics=n_topics, id2word=dictionary)
```

- Évaluations du modèle LDA pour l'hyperparamètre n_topics optimal.
  - L'évaluation quantitative telle que la ***perplexité*** qui mesure la capacité du modèle probabiliste à généraliser.
  - L'évaluation qualitative telle que la ***PMI*** qui évalue automatique la cohérence du modèle basé sur le corpus d'entrée.

LDA : évaluation

- La cohérence du modèle.



- Premiers sujets.

```
display_topics(ldaModel, no_top_words, -1)
```

```
------------------------------------------
Topic 0:
error server file try build project run system connection studio
------------------------------------------
------------------------------------------
Topic 1:
string double date swift time format contribution type way include
------------------------------------------
------------------------------------------
Topic 2:
database key row datum table value store null type data
------------------------------------------
------------------------------------------
```

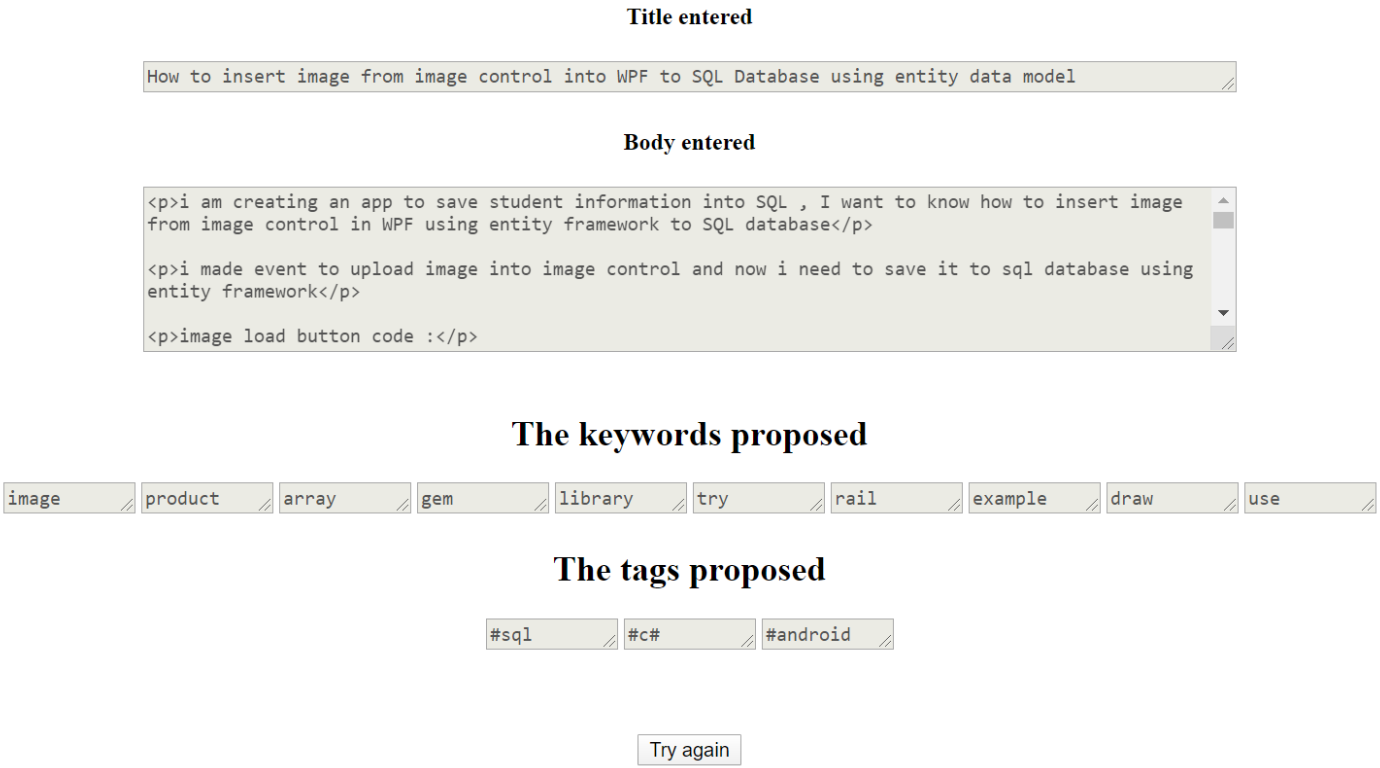
Sommaire

Prédiction : api

- Plusieurs objets sauvegardés pour le déploiement :
  1. Le dictionnaire LDA
  2. Le modèle LDA
  3. La classe CountVectorizer
  4. Le modèle XGBoost
  5. Les tags d'entrainement
  6. Les abréviations d'entrainement
- La fonction de pré-traitement est bien définie
- Le déploiement en local avec Flask : framework d'application web.
- L'api répond à l'adresse 127.0.0.1 :5000/predict à la méthode POST. Les champs doivent porter les noms(name) respectivement 'title' et 'body' pour le titre et le corps du post.

Prédiction : exemple

- Test sur les données de la requête :
  SELECT * FROM posts WHERE FavoriteCount < 50



Gestion de version

- Dépôt disponible sur Github : https ://github.com/AnthonyNama/Categoriser-automatiquement-des-questions
- Cloner : git clone https ://github.com/AnthonyNama/Categoriser-automatiquement-des-questions.git
- Après modification des fichiers : suivre l'ensemble d'instruction :
  1. git status
  2. git add -a
  3. git commit -m “message”
  4. git push origin master

Sommaire

Conclusion

- L'étape d'analyse textuelle est critique dans les problèmes NLP que toute autre application.
- Le plongement de mots apportent une amélioration aux représentations bag of words. Néanmoins, ces dernières ont toujours leur place dans la résolution de problème NLP.
- L'algorithme XGBoost semble offrir les meilleures performances dans l'ensemble des représentations des corpus.
- Le temps d'apprentissage ou de fine tuning des modèles NLP est assez important pour avoir un modèle des plus performants et nécessite de nombreuses ressources.

Perspectives

- Manipuler les 'emoji' : **import emoji**
- Ajouter les bi-grammes ou tri-grammes à la représentation Bag of words pour vérifier l'amélioration du modèle.
- Améliorer le modèle avec des features linguistiques(POS, NER, etc.).
- Revoir le nettoyage de la donnée pour améliorer les résultats pour la méthode distilBert qui nécessite des phrases corrects.
- Utiliser la méthode SVD pour le word embedding.
- Utiliser l'algorithme NMF pour l'extraction de sujets.