Année 2020 – 2021
Équipe(s) : OpenCalssRooms
Encadrant(s) : B.Beaufils

Machine Learning

CATÉGORISEZ AUTOMATIQUEMENT DES QUESTIONS

Khellouf Leila

23 mai 2021

Résumé

Ce rapport présente les techniques fondamentales du traitement automatique de la langue (*Naturel Language Processing NLP en anglais*) et l'enjeu métier de ce projet est de proposer au site de **StackOverflow** (un forum d'aide pour les développeurs) **un modèle de suggestion automatique de tags**

Mots clés : NLP, OnevsRest, Logistic regression, LDA, api

Sommaire

1 Introduction

L'objectif de ce rapport est de comprendre à quoi sert le traitement du langage naturel et comment s'en servir. Commencons par définir ce concept.

1.1 Qu'est ce que NLP ?

Le **NLP**(Naturel Language Processing. traitement automatique du langage naturel en francais) est une branche de l'intelligence artificielle qui consiste à relier les machines à comprendre les humains dans leur langage naturel. Le langage naturel peut être sous forme de texte ou de son, qui sont utilisés pour que les humains se communiquent entre eux. La NLP peut permettre aux humains de communiquer avec les machines de manière naturelle.

1.2 Quelques application NLP :

Le traitement du langage naturel (NLP) permet de répondre à différents besoins, à savoir :

- Traduction de texte
- Résumé automatique de texte
- Classification de texte (ce qui permet de filter les spams dans nos boites mail par example)
- Analyse des sentiments / opinions/ discours

1.3 Les Frameworks les plus utilisés avec Python

Une librairie NLP est une boîte à outils qui traite des données non structurées de différentes sources, ce qui nous permet de les comprendre et d'obtenir des informations précieuses. Les bibliothèques NLP les plus utilisées en Python sont :

- **NLTK** : Le Natural Language Toolkit est relativement mature (il est en développement depuis 2001) et s'est positionné comme l'une des principales ressources en matière de traitement du langage naturel. NLTK possède des outils pour presque toutes les tâches NLP.
- **Gensim** : Gensim est une bibliothèque hautement optimisée pour la modélisation sémantique non supervisée (Topic modeling : LDA, LSI. . .) et la similitude de documents.
- **spaCy** : est une librairie plus récente (2015) qui fournit la plupart des fonctionnalités standard (tokenisation, analyse, reconnaissance d'entités nommées. . .). Elle est conçue pour être très rapide.

1.4 Problématique

Pour poser une question sur le site de Stack Overflow, il faut entrer plusieurs tags de manière à retrouver facilement la question par la suite. Il est judicieux de suggérer quelques tags relatifs à la question posée pour les utilisateurs(généralement les débutants) qui ne savent les étiquettes associées à leur question. Il est donc question de développer un système qui assigne automatiquement plusieurs tags pertinents à une question.

1.5 Les principales méthodes utilisées en NLP

Globalement, nous pouvons distinguer deux aspects essentiels à tout problème de NLP :

- La **partie "linguistique"**, qui consiste à prétraiter et transformer les informations en entrée en un jeu de données explotable.
- La **partie "apprentissage automatique"** ou "Data science", qui porte sur d'application de modèles de Machine Learning ou Deep Learning à ce jeu de données.

2 La phase de prétraitement : du texte aux données

Avant qu'un texte donné soit utilisable, il est indispsensable de transformer les données brutes en des données exploitables.

Les données : Stack Overflow propose un outil d'export de données stackexchange explore, qui recense un grand nombre de données authentiques de la plateforme d'entraide.

2.1 Tokenisation

La tokenisation cherche à transformer un texte en une série de tokens individuels. Dans l'idée, chaque token représente un mot.

```
import nltk
nltk.download('punkt')

from nltk.tokenize import word_tokenize

question= " I'd like to be able to introspect a C++ class for its name"

token= word_tokenize(question)
print(token)
```

```
['I', "'d", 'like', 'to', 'be', 'able', 'to', 'introspect', 'a', 'C++', 'class', 'for', 'its', 'name']
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\PC\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

2.2 Suppression des tags HTML

L'utilisation de la bibliothèque **BeautifulSoup** permet de supprimer les balises HTML.

	Body	Title	Tags	Text	id
0	<p>I found an example in the <a href="http://m...	Dynamic LINQ OrderBy on IEnumerable<T> / IQuer...	<c#><linq><linq-to-objects>	dynamic linq orderby on ienumerable<t> / iquer...	0
1	<p>RequireJS seems to do something internally ...	Prevent RequireJS from Caching Required Scripts	<javascript><jquery><requirejs>	prevent requirejs from caching required script...	1
2	<p>Inspired by <a href="https://devblogs.micro...	How do you rotate a two dimensional array?	<algorithm><matrix><multidimensional-array>	how do you rotate a two dimensional array?_<p>...	2
3	<p>What is the best way to get <code>IDENTITY<...	Best way to get identity of inserted row?	<sql><sql-server><tsql>	best way to get identity of inserted row?_<p>w...	3
4	<p>I am trying to synchronise a project that I...	How do you synchronise projects to GitHub with...	<android><github><intellij-idea><android-studio>	how do you synchronise projects to github with...	4

```
from bs4 import BeautifulSoup
from urllib import request
for i, row in data_tag.iterrows():
    data_tag.loc[i, "Body"]= BeautifulSoup(str(row["Body"]), 'html.parser').get_text()
    data_tag.loc[i, "Text"]= BeautifulSoup(str(row["Text"]), 'html.parser').get_text()
```

```
data_tag.head()
```

	Body	Title	Tags	Text	id
0	I found an example in the VS2008 Examples for ...	Dynamic LINQ OrderBy on IEnumerable<T> / IQuer...	<c#><linq><linq-to-objects>	dynamic linq orderby on ienumerable / iqueryab...	0
1	RequireJS seems to do something internally tha...	Prevent RequireJS from Caching Required Scripts	<javascript><jquery><requirejs>	prevent requirejs from caching required script...	1
2	Inspired by Raymond Chen's post, say you have ...	How do you rotate a two dimensional array?	<algorithm><matrix><multidimensional-array>	how do you rotate a two dimensional array?_ins...	2
3	What is the best way to get IDENTITY of insert...	Best way to get identity of inserted row?	<sql><sql-server><tsql>	best way to get identity of inserted row?_what...	3
4	I am trying to synchronise a project that I ha...	How do you synchronise projects to GitHub with...	<android><github><intellij-idea><android-studio>	how do you synchronise projects to github with...	4

2.3 Mise en minuscules et suppression des nombres, links...

Cette phase consiste à réaliser des taches de la suppression d'urls, suppression des nombres et transformation des majuscules en minuscules.

```
def clean_text(text):
    # Make text lowercase
    text = str(text).lower()
    # remove text in square brackets,
    text = re.sub('\[.*?\]', '', text)
    # remove links
    text = re.sub('https?://\S+|www\.\S+', '', text)
    # remove words containing numbers.
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

2.4 Stopwords

Les stopwords sont des mots couramment utilisés en anglais qui n'ont aucune signification contextuelle dans une phrase. Nous les supprimons donc avant la classification.

```
stop_words= stopwords.words('english')
more_stopwords= ['don', 'im', 'o']
stop_words= stop_words + more_stopwords

def remove_stopwords(text):
    text= ' '.join(word for word in text.split(' ') if word not in stop_words)
    return text
```

2.5 Les forme contractées

L'utilisation de la bibliothèque re de Python pour chercher et remplacer les différents patterns.

```
def remove_contract_form(text):
    document = nltk.word_tokenize(text)
    words = []
    for token in document:
        text = token
        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)
        words.append(text)
    return ' '.join(words)
```

2.6 La ponctuation et les symboles spéciaux

Les ponctuations et symboles spéciaux dans le traitement du texte de base sont comme des mots qui n'apportent pas d'information. Les expressions régulières permettent de les filtrer efficace. Les patterns retrouvés sont remplacés par un espace(' ') avec une conditions sur les targets utiliser dans ce projet.

```
# 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
```

```
def clean_punct(text):
    ''' Remove all the punctuation from text, unless it's part of an important
    tag (ex: c++, c#, etc)

    Parameter:

    text: text to remove punctuation from it
    '''

    words = token.tokenize(text)
    punctuation_filtered = []
    regex = re.compile('[%s]' % re.escape(punct))
    remove_punctuation = str.maketrans(' ', ' ', punct)

    for w in words:
        if w in top_tags:
            punctuation_filtered.append(w)
        else:
            w = re.sub('^[0-9]*', " ", w)
            punctuation_filtered.append(regex.sub('', w))

    filtered_list = strip_list_noempty(punctuation_filtered)

    return ' '.join(map(str, filtered_list))
```

2.7 Lemmatisation

Se réfère généralement à faire les choses correctement avec l'utilisation d'un vocabulaire et une analyse morphologique des mots, visant normalement à supprimer uniquement les fins flexionnelles et à renvoyer la forme de base et de dictionnaire d'un mot.

Il existe plusieurs algorithmes de dérivation implémentés dans la bibliothèque NLTK Python :

— PorterStemmer
— SnowballStemmers
— LancaterStemmer

```python
stemmer = nltk.SnowballStemmer("english")

def stemm_text(text):
    text= ' '.join(stemmer.stem(word) for word in text.split(' '))
    return text

```

2.8 Suppression des espaces multiples et des mots de taille faible

Après les opérations précédentes, des espaces multiples peuvent être créés.

Les mots de taille 1 ne sont pas porteurs d'informations ; d'où leurs suppressions, dans le cadre de ce projet, nous avons gardé certains mots avec les étiquettes tels que : c, r, etc.

```python
def delete_multiple_space(text):
    return ' '.join(text.split())
```

```python
# Delete string of length = 1 and not in tags
def low_length(text):
    document = nlp(text)
    words = []
    for token in document:
        if len(token.text) > 1:
            words.append(token.text)
        else:
            if token.text in top:
                words.append(token.text)
    return ' '.join(words)
```

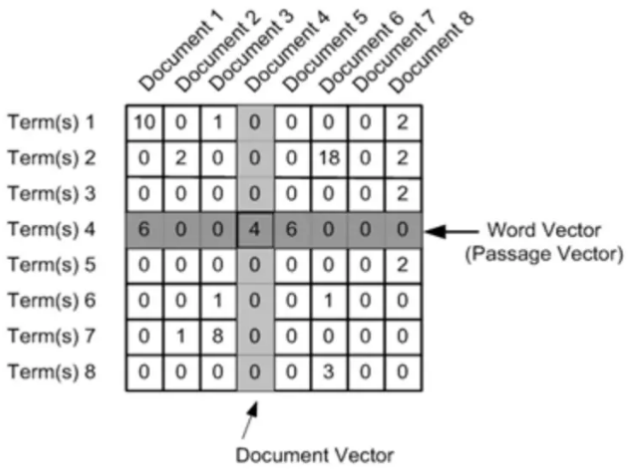
3 Extraction des features

Afin de pouvoir appliquer les méthodes de Machine Learning aux problèmes relatifs au langage naturel, il est indispensable de transformer les données textuelles en données numériques. Il existe plusieurs approches dont les principales sont les suivantes

— Comptage des mots TF
— Pondération des mots TF-IDF
— word embeddings (Glove)

3.1 Bag-Of-Words

Cette méthode consiste à compter le nombre d'occurrences des tokens présents dans le corpus pour chaque texte. Chaque texte est alors représenté par un vecteur d'occurrences. On parle généralement de Bag-Of-Word, ou sac de mots en français.




```python
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
corpus = [
     'This is the first document.',
     'This is the second second document.',
     'And the third one.',
     ]

X = vectorizer.fit_transform(corpus)
print(X)
```

```
(0, 8)        1
(0, 3)        1
(0, 6)        1
(0, 2)        1
(0, 1)        1
(1, 8)        1
(1, 3)        1
(1, 6)        1
(1, 1)        1
(1, 5)        2
(2, 6)        1
(2, 0)        1
(2, 7)        1
(2, 4)        1
```

Néanmoins, cette approche présente un inconvénient majeur : certains mots sont par nature plus utilisés que d'autres, ce qui peut conduire le modèle à des résultats erronés.

3.2 Term Frequency-Inverse Document Frequency (TF-IDF)

Cette méthode consiste à compter le nombre d'occurrences des tokens présents dans le corpus pour chaque texte, que l'on divise ensuite par le nombre d'occurrences total de ces même tokens dans tout le corpus.

Pour le terme x présent dans le document y, on peut définir son poids par la relation suivante :

$$w_{x,y} = tf_{x,y} \cdot log(\frac{N}{df_x})$$

Où :

- $tf_{x,y}$ est la fréquence du terme x dans y ;
- df_x est le nombre de documents contenant x ;
- N est le total de documents.

```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [
     'This is the first document.',
     'This is the second second document.',
     'And the third one.',
 ]
vectorizer = TfidfVectorizer()
vect= vectorizer.fit_transform(corpus)
print(vect)
```

```
(0, 1)        0.43306684852870914
(0, 2)        0.5694308628404254
(0, 6)        0.33631504064053513
(0, 3)        0.43306684852870914
(0, 8)        0.43306684852870914
(1, 5)        0.8108387095324792
(1, 1)        0.3083318691673373
(1, 6)        0.23944720188599447
(1, 3)        0.3083318691673373
(1, 8)        0.3083318691673373
(2, 4)        0.546454011634009
(2, 7)        0.546454011634009
(2, 0)        0.546454011634009
(2, 6)        0.3227445421804912
```

L'éfficacité de ces méthodes diffère selon le cas d'application. Toutefois, elles présentent deux principales limites :

— Plus le vocabulaire du corpus est riche, plus la taille des vecteurs est grande, ce qui peut représenter un problème pour les modèles d'apprentissage utilisées dans l'étapes suivante.
— Le comptage d'occurence des mots ne permet pas de rendre compte de leur agencement et donc du sens des phrases.

3.3 Global Vectors for Word Representation (Glove)

Cette méthode consiste à construire des vecteurs de taille fixe qui prennent en compte le contexte dans lequel se trouvent les mots. Ainsi, deux mots présents dans des contextes similaires auront des vecteurs plus proches (en terme de distance vectorielle). Cela permet alors de capturer à la fois similarités sémantiques, syntaxiques ou thématiques des mots.

4 La phase d'apprentissage : des données au modèle

De manière globale, on peut distinguer **3 principales approches NLP :** les **méthodes basées sur des règles**, modèles classiques de **Machine Learning** et des modèles de **Deep Learning**

4.1 Modèles classiques de Machine Learning

Les représentations précédentes sont utilisés dans nos algorithmes afin de pouvoir effectuer différentes tâches de classification à la fois supervisée pour la prédiction des étiquettes(Logistic Regression) et non supervisée pour modéliser des sujets(LDA).

4.1.1 Approche supervisée

Puisque, dans ce projet on à classifier une observation dans une catégorie parmi 20 (L'étiquette $Y \in \{0, 1, ..., 19\}$ donc on va parler de **Multi-class classification**, pour cela on doit d'abord définir l'algorithme de regression logistique et One-Versus-All.

La **régression logistique** est un modèle de classification linéaire qui est le pendant de la régression linéaire , quand Y ne doit prendre que deux valeurs possibles (0 ou 1). Comme le modèle est linéaire, la fonction hypothèse pourra s'écrire comme suit :

$$S(X^{(i)}) = \phi_0 x_0 + \phi_1 x_1 + \phi_2 x_2 + ... + \phi_n x_n = \sum_{i=0}^{n+1} (\phi_i x_i) \tag{1}$$

avec :

- $X^{(i)}$: une observation (que ce soit du Training Set ou du Test Set) cette variable est un vecteur contenant $x_1, x_2, ..., x_n$
- x_i : est une variable prédictive (feature) qui servira dans le calcul du modèle prédictif.
- ϕ_i : est un poids/ paramètre de la fonction hypothèse. Ce sont ces ϕ_i qu'on cherche à calculer pour obtenir notre fonction de prédiction.
- ϕ_0 est une constante nommée le bias (biais)

On a définit précédemment $X^{(i)}$ comme étant un vecteur de $x_1, x_2, ..., x_n$. Faisant la même chose pour les $\phi_{(i)}$:

soit grand phi ϕ le vecteur contenant $\phi_0, \phi_1, ..., \phi_n$,.

Notre fonction hypothèse peut- être vue comme le produit de deux vecteurs ϕ et X :

$$S(X) = \phi X \tag{2}$$

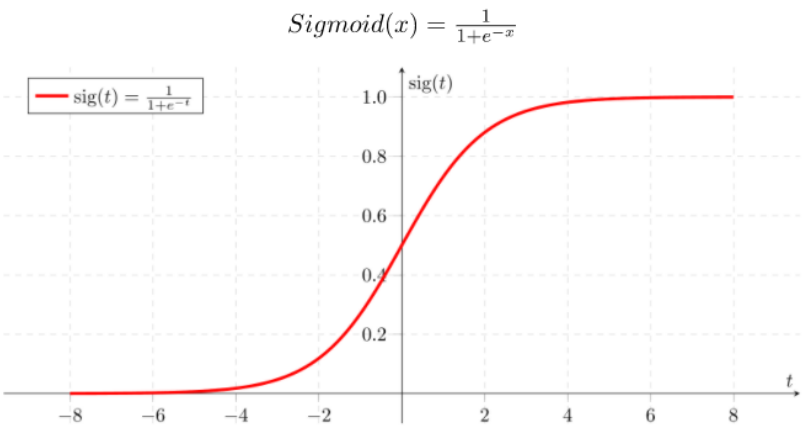
On appelle cette fonction hypothèse : la fonction score. L'idée est de trouver des coefficients $\phi 0, \phi 1, \ldots, \phi n$ de sorte que :

- $S(X^{(i)}) > 0$ quand la classe (étiquette) vaut 1
- $S(X^{(i)}) < 0$ quand la classe (étiquette) vaut 0

Sigmoid function pour calculer la probabilité d'une classe

La fonction score qu'on a obtenue intègre les différentes variables prédictives (les x_i). A cette fonction, on appliquera la fonction sigmoid (Sigmoid Function). Cette fonction produit des valeurs comprises entre 0 et 1. Le résultat obtenu par la fonction sigmoid est interprété comme **la probabilité que l'observation X soit d'un label (étiquette) 1**.

La fonction Logistique (autre nom pour la fonction Sigmoid), est définie comme suit :



En appliquant cette fonction sigmoid sur notre fonction score, on obtient notre fonction hypothèse pour la régression logistique :

$$H(X) = sigmoid(S(X)) = sigmoid(\phi X) = 1 \div (1 + \exp(\phi X)) \tag{3}$$

L'agorithme **One-Versus-All**(One vs All) permet d'utiliser **Logistic Regression** pour la classification mutli-classes. Le principe est simple : il consiste à découper le problème de classification multi-classes en une **multitude de problème de classification** binaires.

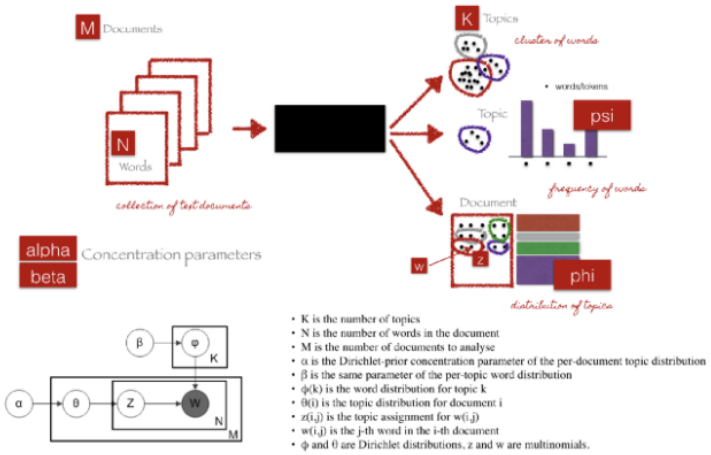
Supposant que l'étiquette *C#* correspond à la classe 1, $javascript$ à la classe 2 et *.net* à la classe 3. L'algorithme *One-vs-All* va procéder comme suit :

- **Etape 1** : On considère que les étiquettes *C#* sont la classe positive (étiquette 1) et le reste comme la classe négative (dans ce cas, javascript et .net serant dans le meme groupe de la classe négative (étiquette 0), et on entraine la régression logistique sur cette configuration de données. Ce qui produira une fonction de prediction $H^1(x)$
- **Etape 2** : On considère l'étiquette $javascript$ comme la classe positive et le reste comme la classe négative, et on entraine la régression logistique pour obtenir une deuxième fonction de prédiction :$H^2(x)$
- **Etape 3** : on considère les étiquettes .net comme la classe positive et le reste comme la classe négative, et on entraine la régression logistique pour obtenir $H^3(x)$

Chacun des ces fonctions de prédiction $H^1(x)$, $H^2(x)$, $H^3(x)$ nous dennera la **probabilité que x soit de la classe** Y_i . La bonne classe de l'obseravtion x est celle pour laquelle on a obtenu la **plus grande probabilité** En d'autres termes, la classe de x et le $maxH^i(x)$, avec K le nombre de classes (étiquettes) possibles.

4.1.2 Approche non supervisée

Latent Dirichlet Allocation (LDA) est un processus probabiliste génératif, conçu dans le but spécifique de découvrir la structure de sujet latente dans les corpus de texte.



http://chdoig.github.io/pytexas2015-topic-modeling/#/3/4

- Nous pouvons décrire le processus génératif de LDA comme, étant donné le nombre M de documents, le nombre N de mots et le nombre K préalable de sujets, le modèle s'entraîne à produire :

— ψ, la distribution des mots pour chaque sujet K
— ϕ, la répartition des sujets pour chaque document i

L'inférence de ce modèle est donc de 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**. Les deux principales entrées du modèle LDA topic sont le dictionnaire (id2word) et le corpus.

```
corpus_2= []
for i, row in data_tag.iterrows():
    obs= str(row["Text_clean"])
    corpus_2.append(obs)
```

```
data_lemmatized = [text.split() for text in corpus_2]

# Create Dictionary
id2word= corpora.Dictionary(data_lemmatized)

# create Copus
texts= data_lemmatized

# Term Document Frequency
corpus_= [id2word.doc2bow(text) for text in texts]

# view
print(corpus_[:1][0][:])
```

```
2021-05-16 00:27:11,666 : INFO : adding document #0 to Dictionary(0 unique tokens: [])
2021-05-16 00:27:13,630 : INFO : built Dictionary(50479 unique tokens: ['a', 'age', 'allow', 'an', 'ani']...) from 8763 documents (total 946031 corpus positio
2021-05-16 00:27:13,644 : INFO : Dictionary lifecycle event {'msg': "built Dictionary(50479 unique tokens: ['a', 'age', 'allow', 'an', 'ani']...) from 8763 do
946031 corpus positions)", 'datetime': '2021-05-16T00:27:13.634698', 'gensim': '4.0.1', 'python': '3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (
form': 'Windows-10-10.0.19041-SP0', 'event': 'created'}

[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 2), (7, 1), (8, 2), (9, 2), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1), (16, 1), (17, 1), (18,
0, 2), (21, 1), (22, 1), (23, 3), (24, 1), (25, 1), (26, 1), (27, 1), (28, 1), (29, 1), (30, 1), (31, 2), (32, 1), (33, 1), (34, 2), (35, 1), (36, 1), (37, 1)
1), (40, 1)]
```

Gensim crée un identifiant unique pour chaque mot du document. Le corpus produit illustré ci-dessus est un mappage de (word_id, word_frequency).

Par exemple, (0, 1) ci-dessus implique, le mot id 0 apparaît une fois dans le premier document. De même, le mot id 1 apparaît une fois et ainsi de suite.

Ceci est utilisé comme entrée par le modèle LDA.

```
# Human readable format of corpus (term-frequency)
[[(id2word[id], freq) for id, freq in cp] for cp in corpus_[:1]]
```

```
[[('a', 1),
  ('age', 1),
  ('allow', 1),
  ('an', 1),
  ('ani', 1),
  ('desc', 1),
  ('dynam', 2),
  ('eg', 1),
  ('exampl', 2),
  ('for', 2),
  ('found', 1),
  ('function', 1),
  ('get', 1),
  ('ienumer', 1),
  ('ienumerablet', 1),
  ('in', 1),
  ('includ', 1),
  ('iqueryablei', 1),
  ('iqueryablet', 1),
  ('is', 1),
  ('linq', 2),
  ('method', 1),
  ('name', 1),
  ('on', 3),
```

Nous avons tout ce qu'il faut pour former le modèle LDA de base. En plus du corpus et du dictionnaire, on doit également fournir le nombre de Topics. En dehors de cela, alpha et eta sont des hyperparamètres qui affectent la sparsity des topics. Selon la documentation Gensim, les deux valeurs par défaut sont 1.0 / num_topics avant (nous utiliserons la valeur par défaut pour le modèle de base).

'chunksize' contrôle le nombre de documents traités à la fois dans l'algorithme d'apprentissage. L'augmentation de la taille des morceaux accélérera la formation, au moins tant que le morceau de documents rentrera facilement dans la mémoire.

'passes' contrôle la fréquence à laquelle nous entraînons le modèle sur l'ensemble du corpus (défini sur 10). Un autre mot pour les passes pourrait être «epochs». les itérations sont quelque peu techniques, mais elles contrôlent essentiellement la fréquence à laquelle nous répétons une boucle particulière sur chaque document. Il est important de définir le nombre de «passes» et d '«itérations» suffisamment élevé.

```
# Build LDA model
lda_model = gensim.models.LdaMulticore(corpus=corpus_,
                                       id2word=id2word,
                                       num_topics=10,
                                       random_state=100,
                                       chunksize=100,
                                       passes=10,
                                       per_word_topics=True)
```

```
2021-05-16 00:27:14,696 : INFO : using symmetric alpha at 0.1
2021-05-16 00:27:14,699 : INFO : using symmetric eta at 0.1
2021-05-16 00:27:14,741 : INFO : using serial LDA version on this node
2021-05-16 00:27:14,863 : INFO : running online LDA training, 10 topics, 10 passes over the supplied corpus of 8763 documents, updating every 300 document
~3000 documents, iterating 50x with a convergence threshold of 0.001000
2021-05-16 00:27:14,886 : INFO : training LDA model using 3 processes
```

Le modèle LDA ci-dessus est construit avec 10 topics différents où chaque topic est une combinaison de mots-clés et chaque mot-clé apporte une certaine pondération au topic.

- Nous pouvons voir les mots-clés pour chaque sujet et la pondération (importance) de chaque mot-clé en utilisant la fonction lda_model.print_topics ()

```
no_top_words = 10
n_topics = 10
def display_topics(model, no_top_words, num_topics):
    for idx, topic in model.show_topics(formatted=False, num_topics=num_topics, num_words= no_top_words):
        print("-----------------------------------------")
        print("Topic %d:" % (idx))
        print(" ".join([w[0] for w in topic]))
        print("-----------------------------------------")

display_topics(lda_model, no_top_words, n_topics)
```

```
-----------------------------------------
Topic 0:
file test build py librari lib version python instal packag
-----------------------------------------
-----------------------------------------
Topic 1:
file error use get request tri server run user script
-----------------------------------------
-----------------------------------------
Topic 2:
use would like one way question know work want look
-----------------------------------------
-----------------------------------------
Topic 3:
imag size line number locat formula color print rang get
-----------------------------------------
-----------------------------------------
Topic 4:
git branch commit node js file chang repositori push remot
-----------------------------------------
```

Comment interpréter cela ?

Le sujet 0 est représenté par

$(0,'0.024*"file"+0.021*"test"+0.021*"build"+0.021*"py"+0.020*"librari"'+0.017*"lib"+0.017*"version"+0.017*"python"+0.016*"instal"+'0.015*"packag"')$

Cela signifie que les 10 principaux mots clés qui contribuent à ce sujet sont : 'div', 'class', 'px' .. et ainsi de suite et que le poids de 'div' sur le sujet 0 est de 0,088.

Les pondérations reflètent l'importance d'un mot-clé pour ce sujet.

En regardant ces mots clés, on peut deviner ce que pourrait être ce topic , on peut le résumer soit «c #» ou «python», «JAVA», ...etc.

De même, pouvez-vous parcourir les mots-clés de sujet restants et juger quel est le sujet ?

```
# compute Perplexity - lower is the better
perplexity= lda_model.log_perplexity(corpus_)
print('\nPerplexity: ', perplexity)
```

```
2021-05-16 00:29:09,494 : INFO : -7.664 per-word bound, 202.8 perplexity estimate based on a held-out corpus of 8763 documents with 946031 words
```

La perplexité du modèle et la [cohérence du sujet] fournissent une mesure pratique pour juger de la qualité d'un modèle de sujet donné. Il se trouve que, le score de cohérence du sujet, en particulier, a été plus utile.

```
from gensim.models import CoherenceModel

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized, dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('Coherence Score: ', coherence_lda)
```

```
2021-05-16 00:29:09,537 : INFO : using ParallelWordOccurrenceAccumulator(processes=3, batch_size=64) to estimate probabilities from sliding windows
2021-05-16 00:29:17,420 : INFO : 5 batches submitted to accumulate stats from 320 documents (-5298 virtual)
2021-05-16 00:29:17,645 : INFO : 7 batches submitted to accumulate stats from 448 documents (-4280 virtual)
```

Pour visualiser les bon topics, on a utiliser la bibiothéque pyLDAvis, ou le modèle qui donnera un bon topic aura des bulles assez grosses et non superposées dispersées.

```
pyLDAvis.enable_notebook()
import pyLDAvis.gensim_models as pgm

lda_vis= pgm.prepare(lda_model, corpus_, id2word)
lda_vis
```

5 Evaluation

Il s'agit ici pour nous de présenter la métrique utilisée ainsi que de décrire le processus d'évaluation pour nos algorithmes multi-classes.

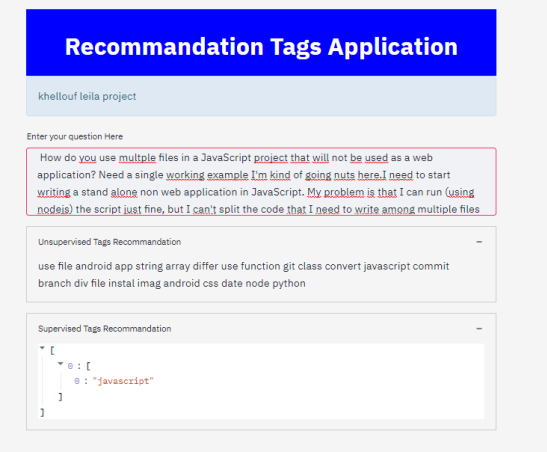
La métrique utilisée est le score du coefficient de similarité Jaccard ce score est calculé pour chaque instance, et il trouve leur moyenne en pourcentage.

```
:
def avg_jaccard(y_true, y_pred):
    ''' It calculates Jaccard similarity coefficient score for each instance,and
    it finds their average in percentage

    Parameters:

    y_true: truth labels
    y_pred: predicted labels
    '''
    jacard = np.minimum(y_true, y_pred).sum(axis=1) / \
        np.maximum(y_true, y_pred).sum(axis=1)

    return jacard.mean()*100
```

6 Déploiement

Le déploiement s'est fait avec **Streamlit** qu'est un framework d'application open-source pour les équipes d'apprentissage automatique et de science des données. l'api répond à l'adresse : Voir l'api

Lors de la création de l'api, plusieurs objets sont intégrés à savoir l'extracteur de feature (CountVectorizer) et le modèle de prédiction d'étiquettes entrainés. Les étapes de pré-traitement de la donnée on été réspecter avant toutes prédictions.



7 Conclusion

Dans ce document, après une brève vue d'ensemble du NLP, nous avons évoqué tour à tour les processus de pré-traitement, d'extraction de features, les algorithmes de modélisation, l'évaluation et le déploiement. Nous pouvons dire qu'il y'a des étapes plus critiques que d'autres qui doivent faire l'objet d'une attention particulière à savoir :

- **Data set** : entrainer les modèles sur une grande base de donnée.
- **le prétraitement** : Manipulation des URL, emoji par example,..etc
- **Apprentissage des features** : Utiliser les modèles de Deep Learning (LSTM) avec le word embedding
- Trouver de nouvelle **métrique d'évaluation**