Chapitre 3
Le traitement du langage naturel

1. Introduction

Le chapitre précédent a permis de préciser deux champs importants relatifs au traitement automatique du langage naturel : l'intelligence artificielle et la linguistique. Avant d'entrer concrètement dans le sujet, avec Python et les bibliothèques Python **spaCy**, **NLTK**, **Gensim** ou encore **scikit-learn**, ce chapitre est consacré au traitement automatique du langage naturel lui-même, c'est-à-dire aux outils et thématiques qui le composent. Le but est clairement de définir préalablement des notions et des éléments de vocabulaire, avant de les rencontrer dans les chapitres expérimentaux. D'autant que nombre d'éléments de vocabulaire ou de terminologies utilisées dans le TALN le sont en général et usuellement en langue anglaise.

Pour chaque sujet, nous prendrons éventuellement des exemples de bibliothèques Python que nous utiliserons par la suite. Les premières propositions d'implémentations des différents sujets arriveront dès le chapitre suivant.

2. Le nettoyage préalable d'un modèle

Avant de construire un modèle de traitement automatique du langage naturel, il est essentiel d'effectuer un nettoyage préalable des données textuelles. Ce processus vise à éliminer le bruit, à normaliser le texte et à préparer les données pour une analyse plus précise et efficace.

2.1 Nettoyage de base

Il est souvent nécessaire de procéder à un nettoyage selon plusieurs critères :

– la suppression des caractères spéciaux ;

– la conversion en minuscules ;

– la gestion des accents et des diacritiques (« Benoît » devient « Benoit ») ;

– plus généralement, la transformation des caractères non ASCII.

L'usage des expressions régulières est souvent très pratique pour cette étape parfois plus compliquée qu'il n'y paraît au premier abord. Par exemple, si vous vous intéressez à une base de documents traitant de langages informatiques, la suppression de caractères comme « + » ou « # » va entraîner l'assimilation des langages C++ et C# au langage C. Une méthode possible est alors de remplacer « + » par « plus » et « # » par « sharp », pour conserver l'information relative aux langages cplusplus et csharp.

2.2 Stopwords

L'une des premières étapes du nettoyage consiste à supprimer les mots vides, ceux qui « n'apportent rien », également connus sous le nom de ***stopwords***. Les stopwords sont des mots courants qui n'apportent pas beaucoup d'informations spécifiques à un texte donné, tels que « le », « la », « et », « mais », etc. La suppression de ces mots permet de réduire la dimensionnalité des données et de se concentrer sur les termes clés plus significatifs.

spaCy et NLTK, entre autres, proposent des listes de stopwords utilisables d'emblée et évidemment modifiables.

2.3 Stemming (racinisation)

Cette technique vise à réduire les mots à leur racine ou à leur forme de base. Par exemple, les mots « jouer », « jouant » et « joué » peuvent être racinisés en « joue ». Cela permet de regrouper différentes formes d'un mot et d'éviter la fragmentation des informations lors de l'analyse.

Mais il faut bien se rappeler que dès cette étape de nettoyage, l'objectif de la prédiction est fondamental. Dans une langue comme l'anglais, le ***stemming*** va assimiler « powerful » à « powerless », ce qui aurait des effets dévastateurs si on vise une analyse des sentiments. Si on cherche à classifier des documents, il est probable que cela ne pose pas forcément de problème. Cette remarque insiste sur le fait que l'entraînement du modèle dans ses objectifs de prédiction commence dès la phase de nettoyage, qui en ce sens est déjà fonctionnelle.

2.4 Lemmatisation

La **lemmatisation** est assez proche de la racinisation, mais d'une part utilise d'autres algorithmes, et d'autre part a pour réel objectif de retrouver le lemme d'un mot, par exemple l'infinitif pour les verbes.

Là aussi, cette technique peut s'avérer utile pour réduire le nombre de dimensions des données d'entrée.

Dans les faits, on cherche à faire cela :

– maisons => maison

– mangées, mangé, mangera, mange => manger

– bleues, bleue => bleu

3. La notion d'hyperparamètre

Lorsque l'on travaille avec un modèle d'intelligence artificielle, on a ce que l'on appelle les ***features***, les colonnes de notre modèle, les données qui participent fonctionnellement à l'apprentissage et donc à la prédiction future. Et il y a d'autres données qui sont, elles, dédiées spécifiquement au réglage de l'apprentissage. C'est ce que l'on appelle des **hyperparamètres**.

Lorsqu'il s'agit de modèles TALN, tels que les modèles de classification de texte, les modèles de génération de texte ou les modèles de traduction automatique, plusieurs hyperparamètres sont particulièrement importants. Par exemple, le choix du nombre de couches et de neurones dans un réseau neuronal est un hyperparamètre important à considérer. Il peut affecter la capacité du modèle à capturer des structures complexes et à généraliser efficacement. Un autre hyperparamètre essentiel est le taux d'apprentissage, qui contrôle la vitesse à laquelle un modèle est ajusté aux données. Un taux d'apprentissage trop élevé peut entraîner une convergence rapide mais instable, tandis qu'un taux d'apprentissage trop faible peut ralentir le processus d'apprentissage.

Dans le contexte du *topic modeling* (abordé à la suite dans ce chapitre), le nombre de sujets à extraire est un hyperparamètre crucial. Le choix d'un nombre approprié de sujets peut avoir un impact significatif sur la qualité de l'analyse thématique et sur la pertinence des résultats obtenus.

Il est important de noter que l'optimisation des hyperparamètres nécessite souvent une approche itérative. Il est courant d'utiliser des techniques comme l'optimisation bayésienne pour explorer différentes combinaisons d'hyperparamètres et trouver les valeurs qui optimisent les performances du modèle. Dans ce cas, on évalue les performances du modèle selon les hyperparamètres choisis. On peut ainsi déduire les valeurs d'hyperparamètres susceptibles de donner les meilleurs résultats en matière de prédiction.

4. La vectorisation et le bag of words

La **vectorisation** fait référence au processus de conversion du texte en une représentation vectorielle, où chaque mot ou terme est associé à un vecteur numérique. Cette transformation permet de capturer des informations sémantiques et syntaxiques du texte, ce qui facilite son traitement et son analyse par les modèles d'apprentissage automatique.

Le ***bag of words*** (ou sac de mots en français) est une méthode couramment utilisée pour la vectorisation du texte. Elle consiste à considérer chaque document comme une collection non ordonnée de mots, en ignorant donc l'ordre et la structure grammaticale du texte. Le sac de mots représente le texte sous forme d'un vecteur de comptage, où chaque dimension correspond à un mot unique et la valeur associée à chaque dimension indique le nombre d'occurrences de ce mot dans le document.

Par exemple, supposons que nous ayons deux phrases : « J'aime les pommes » et « Les pommes sont délicieuses ». En utilisant le sac de mots, nous pouvons construire un vecteur pour chaque phrase en comptant le nombre d'occurrences de chaque mot unique : [1, 1, 1, 0, 0] pour la première phrase et [0, 1, 1, 1, 1] pour la deuxième phrase. Chaque dimension du vecteur correspond à un mot unique (« J'aime », « les », « pommes », « sont », « délicieuses ») et la valeur associée indique le nombre d'occurrences.

Ces deux approches sont très souvent utilisées en TALN mais comportent des limites dès lors que l'on a besoin de comprendre le contexte, ou même seulement de s'intéresser à l'ordre des mots dans le texte.

5. Le topic modeling

Le ***topic modeling***, parfois traduit en « modélisation thématique » en français, est une technique qui vise à découvrir des sujets/thématiques (*topics*) dans un corpus de documents. En TALN, on utilise des algorithmes comme le ***Latent Dirichlet Allocation*** (**LDA**) pour identifier les sujets principaux d'un corpus de textes.

On peut par exemple utiliser Gensim pour accéder à une implémentation de LDA. Après prétraitement des données, on crée un dictionnaire de mots issus du corpus de documents, puis on entraîne un modèle dit LDA. Les sujets seront définis par une liste de mots. Ainsi, on peut obtenir ces deux topics :

– { football, stade, supporter, baseball } qui correspond vraisemblablement à un topic « sport ».
– { ministre, députée, chambre, loi } qui correspond vraisemblablement à un topic « politique ».

Avec LDA (et c'est le cas également avec d'autres algorithmes), le choix des hyperparamètres est important. Ici, il sera essentiel de définir correctement les deux hyperparamètres suivants :

– le nombre de topics ciblé ;
– le nombre de mots par topic.

6. Le word embedding

Le ***word embedding*** est une technique qui consiste à représenter les mots sous forme de vecteurs numériques dans un espace vectoriel. Cette technique est en général traduite en français par **plongement lexical**. Elle repose sur le fait que les mots qui apparaissent dans des contextes similaires ont tendance à avoir des significations équivalentes. C'est donc une approche distributionnelle et statistique qui transpose une éventuelle similarité sémantique de deux mots dans leurs contextes de phrases ou de textes, en une similarité vectorielle dans un espace vectoriel. Exprimé autrement : un mot se définit aussi par son contexte d'utilisation et donc par les mots qui l'entourent dans le texte.

L'algorithme central en matière de ***word embedding***, massivement utilisé en TALN, est l'algorithme d'entraînement de modèles **Word2Vec**. Cet algorithme, qui utilise des réseaux de neurones, se décline selon deux architectures différentes :

– l'architecture **CBOW**, dont l'objectif est de prédire un mot quand on a son contexte, c'est-à-dire le voisinage de ce mot dans le texte ;
– l'architecture **skip-gram**, dont l'objectif est inverse : prédire un contexte, les mots du contexte, quand on a en entrée un mot.

Nous aurons l'occasion d'utiliser cet algorithme, dont l'usage est disponible dans la bibliothèque Python **Gensim** de Google, justement orientée ***topic modeling***.

7. La fréquence d'apparition d'un mot

Cette question est centrale, notamment lors de la fouille de textes, c'est-à-dire lors de l'exploration d'un corpus de plusieurs dizaines ou centaines de documents par exemple.

Pour un mot donné, on va se poser plusieurs questions concernant :

– La fréquence d'apparition du mot M dans le document donné D.

– La fréquence d'apparition du mot M parmi tous les documents : est-il fréquemment utilisé au sein de l'ensemble des documents ? Quelle proportion de documents ne l'utilise pas ?

– La fréquence d'utilisation du mot M dans le document D en tenant compte de sa fréquence au sein de chaque document du corpus. Un mot absent d'un document est une information qu'il faut contextualiser aussi par la possibilité qu'il soit par exemple systématiquement présent dans tous les autres documents.

Exprimé autrement, on s'intéresse ici à la fréquence d'apparition d'un mot, non seulement par sa fréquence au sein d'un document, mais aussi et surtout par sa fréquence au sein de l'ensemble du corpus de documents.

Pour procéder à ce type d'analyse, on utilise en général la méthode **TF-IDF** (*Term Frequency - Inverse Document Frequency*). Pour ce faire, concrètement, on utilisera notamment l'algorithme **TfidfVectorizer**, issu de la bibliothèque Python de Machine Learning **scikit-learn**.

8. Les réseaux de neurones récurrents (RNN)

Les **RNN** (pour *Recurrent Neural Network*) représentent une architecture de réseau neuronal couramment utilisée dans le TALN. Ils sont conçus pour traiter des données séquentielles, telles que du texte, en conservant une mémoire interne des états précédents. Les RNN sont souvent utilisés pour des tâches telles que la classification de texte, la traduction automatique et la génération de texte.

Le fonctionnement des RNN repose sur l'utilisation de boucles récurrentes qui permettent aux informations d'être propagées d'une étape à l'autre dans la séquence. À chaque étape, le RNN prend en compte l'entrée courante et la sortie de l'étape précédente, ce qui lui permet de conserver une mémoire à court terme et de capturer les dépendances séquentielles.

Les RNN peuvent être utilisés dans différentes configurations. On peut notamment citer ces configurations :

– les réseaux de neurones récurrents à état caché simple (**Simple RNN**) ;
– les réseaux de neurones récurrents à longue mémoire (**LSTM** - *Long Short Term Memory*) ;
– les réseaux de neurones récurrents à mémoire à court terme (**GRU** - *Gated Recurrent Unit*).

Ces variantes de RNN offrent des capacités différentes en termes de mémoire et de modélisation des séquences.

spaCy et NLK intègrent des modèles pré-entraînés pour une langue donnée (français, anglais, etc.), y compris des modèles basés sur des RNN.