

Chapitre 1-4

Installer son environnement de travail

1. Introduction

Il ne s'agit ici que de CPython, l'implémentation de référence de Python, et non de PyPy ou Jython.

Quel que soit votre système d'exploitation, vous pouvez installer Python en lisant ce chapitre puis, dans un second temps, installer des bibliothèques tierces au gré de vos besoins (cf. section Installer une bibliothèque tierce) et vous pourrez créer des environnements virtuels (cf. section Créer un environnement virtuel).

Si vous souhaitez installer d'un seul coup Python ainsi que Jupyter (anciennement IPython) et la plupart des bibliothèques scientifiques ou d'analyse de données, vous pouvez aller directement à la section Installer Anaconda, pour installer celui-ci en lieu et place de Python. Vous disposerez alors d'autres méthodes pour gérer les environnements virtuels et pour installer des bibliothèques tierces.

2. Installer Python

2.1 Pour Windows

Le système d'exploitation Windows requiert usuellement l'utilisation d'un installateur pour pouvoir installer un logiciel quel qu'il soit. Si vous disposez de Windows, vous devriez en avoir l'habitude. Python ne déroge pas à la règle.

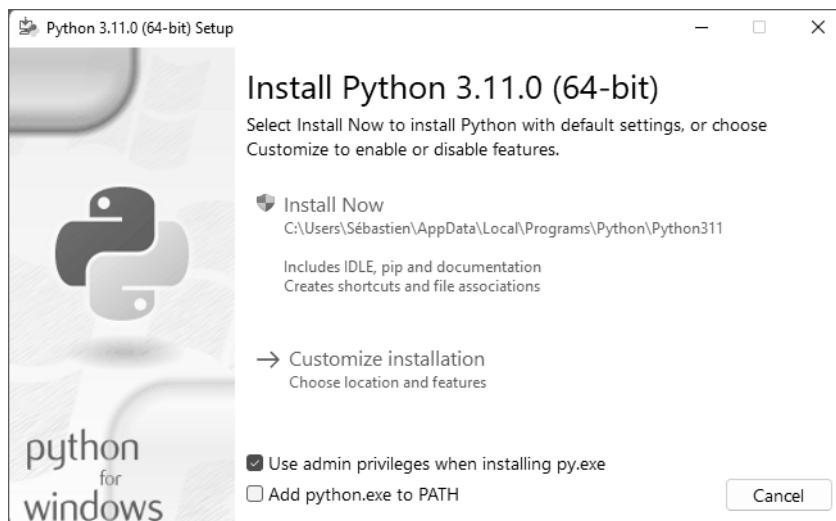
Pour installer Python, vous devez donc aller sur le site officiel (<https://www.python.org/downloads/>) pour télécharger l'installateur adéquat. Comme vous pourrez le constater, on vous met en avant un accès rapide à la dernière version (au moment où ces lignes sont écrites, la 3.11.0), puis un accès aux dernières versions encore actives (actuellement la version 3.10 qui reçoit encore des corrections d'anomalies, puis les versions 3.9 à 3.7 qui reçoivent des corrections de sécurité uniquement).

Il est également possible de télécharger la toute dernière version de la branche 2.7 qui est en fin de vie (elle n'est plus mise à jour), car il existe encore de nombreux projets n'ayant pas encore migré.

Le support correctif dure 2 ans après la première sortie de la version et le support de sécurité dure 5 ans.

Pour notre part, nous vous conseillons la dernière 3.x, mais vous êtes libre d'installer celle que vous souhaitez ou même d'en installer plusieurs suivant vos contraintes, il n'y a pas d'objection à cela.

Une fois le téléchargement effectué, vous devez lancer l'installateur (et éventuellement passer quelques protections de votre système qui vous demande d'accorder votre confiance à cet installateur), pour observer l'écran suivant :

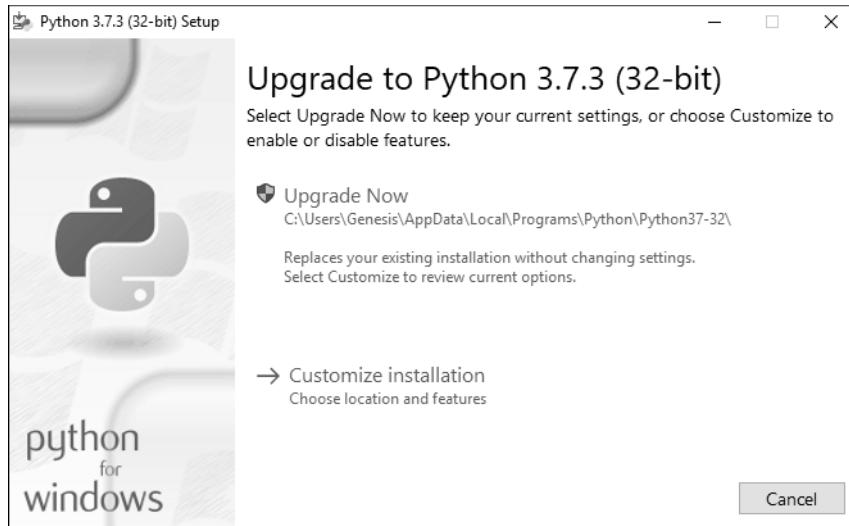


Comme vous pouvez le constater, il est possible de personnaliser l'installation en choisissant le chemin d'installation du logiciel ou en choisissant de ne pas sélectionner quelques fonctionnalités, mais nous ne le conseillons pas.

Nous vous recommandons en revanche de cocher la case **Add python.exe to PATH** afin de configurer la variable PATH du terminal pour rendre Python accessible plus facilement.

Chapitre 1-4

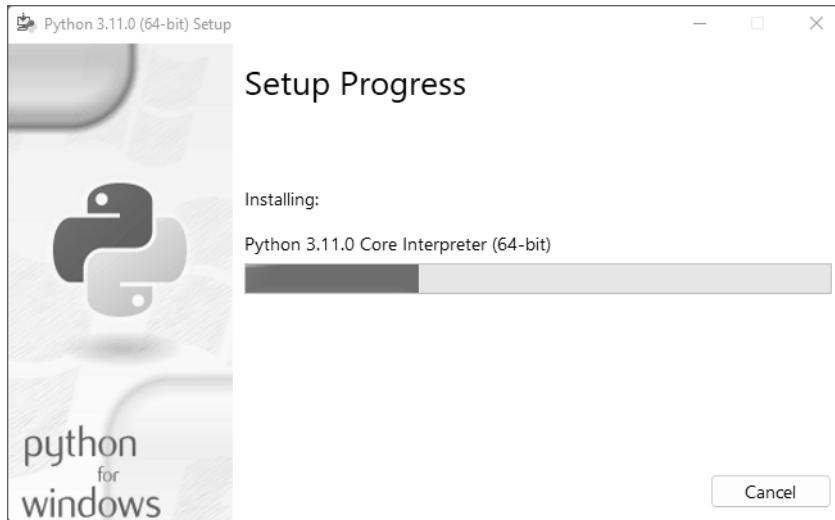
Si vous avez déjà une ancienne version de Python installée de la même branche (dans cet exemple, Python 3.7.2 est déjà installé), vous pourrez la mettre à jour à l'aide du même installeur :



Par contre, si vous avez déjà la version 3.7.1 et que vous installez la version 3.11, cette dernière ne viendra pas remplacer la précédente, mais s'installera à côté. Si vous souhaitez remplacer, il vous faudra donc désinstaller proprement la toute dernière version installée, ce qu'il est possible de faire en relançant l'installeur d'origine.

Nous vous encourageons à garder les installateurs sur votre PC, car ils pourraient devenir indisponibles au téléchargement si trop vieux.

Quel que soit le scénario, vous arriverez devant un écran vous montrant la progression de l'installation et vous n'aurez qu'à fermer la fenêtre une fois celle-ci terminée :



Vous êtes maintenant prêt à utiliser Python.

2.2 Pour Mac

Il faut savoir qu'une version de Python est déjà préinstallée sur Mac, car Mac OS X l'utilise pour ses propres besoins et Python est intégré à son propre cycle de développement. Cependant, si vous souhaitez une version différente de celle qui est déjà présente, vous pouvez l'installer, sachant qu'il n'y a pas de contre-indication à posséder plusieurs versions de Python sur la même machine.

Pour installer Python sur Mac OS X, la procédure à suivre est similaire à celle pour Windows. Il faut donc se rendre sur le site officiel (<https://www.python.org/downloads/mac-osx/>), télécharger un installateur correspondant à sa configuration et suivre les étapes.

Pour les utilisateurs de Mac, sachez que Python dispose d'une bonne intégration de ses spécificités, en particulier vis-à-vis de Objective-C, le langage de programmation avec lequel est développé Mac OS X, et Cocoa, interface de programmation de Mac OS X.

2.3 Pour GNU/Linux et BSD

Les différentes distributions libres utilisent nativement Python, notamment pour des parties sensibles. Python y est donc tout naturellement déjà installé, généralement sous la dernière version de la branche 2.x. Cependant, ici comme ailleurs, il n'y a pas d'objections à utiliser plusieurs versions de Python.

Le plus simple reste d'utiliser votre gestionnaire de paquets, ce qui peut se faire via un outil graphique, comme Synaptic pour Debian :



Il suffit alors de faire une recherche sur le mot-clé **python** pour voir les différentes versions (sur une ancienne Debian, ce sont Python 2.6, 2.7 et 3.2).

Par contre, tous les paquets **python3-xxxxx** que vous pouvez voir ici sont des bibliothèques tierces et non Python lui-même. Nous en parlerons plus tard dans ce chapitre.

Une fois les paquets souhaités sélectionnés, il ne manque plus qu'à les installer en cliquant sur le bouton **Appliquer**.

Notez que tout ceci peut se faire par la simple ligne de commande, toujours en utilisant votre gestionnaire de paquets qui peut être `apt-get`, `aptitude`, `yum`, `emerge`, `pkg_add` ou autre.

Voici par exemple pour une distribution Debian ou Ubuntu :

```
█ $ sudo aptitude install python3
```

Ceci ne permet cependant pas de choisir la version que l'on souhaite, à moins d'aller trouver des sources alternatives. Si l'on veut avoir la toute dernière version de Python, il faudra la plupart du temps passer par la compilation.

2.4 Par la compilation

Compiler Python n'est pas en soi une tâche très complexe. C'est par contre souvent une tâche imposée lorsque l'on ne travaille pas avec des conteneurs. En effet, en entreprise, on développe souvent des applications qui sont destinées à être hébergées. Il est alors impératif de travailler sur votre propre poste avec une version de Python qui soit la même que celle existante sur la machine de production.

Sous GNU/Linux, mais aussi sous d'autres systèmes, il est possible de compiler la version de Python que l'on souhaite. Après tout, Python n'est rien d'autre qu'un programme écrit en C. Pour ce faire, il faut aller télécharger le code source (<https://www.python.org/downloads/source/>), qui prend la forme d'une archive, puis décompresser celle-ci, se placer dans le répertoire ainsi obtenu et taper ces quelques commandes :

```
$ ./configure --prefix=/path/to/my/python/directory  
$ make  
$ sudo make altinstall
```

Notez que dans cette dernière ligne, nous n'utilisons pas la commande **make install**, qui aurait pour effet de remplacer votre Python système par le Python que vous compilez, ce qui pourrait avoir des conséquences indésirables voire désastreuses.

Notez également que vous choisissez lors de la configuration le chemin dans lequel vous placerez vos bibliothèques Python. En général, l'usage veut que l'on utilise **/opt**, mais il n'y a pas de règle, tout dépend des pratiques de votre entreprise ou votre expérience en la matière.

Si vous venez d'installer Python 3.5 par cette méthode, vous aurez alors maintenant accès à ce programme en l'appelant ainsi, depuis votre terminal :

```
$ python3.5
```

Par cette même méthode, vous pouvez installer les dernières versions (<https://www.python.org/download/pre-releases/>) de Python qui ne sont pas encore sorties (alphas ou betas), ce qui vous permet de les tester en avant-première !

Notons que, par cette méthode, toutes les bibliothèques de Python ne fonctionneront pas. En effet, lorsqu'elles ont besoin d'autres bibliothèques C, il faut effectuer des compilations croisées et utiliser les différents en-têtes de ces bibliothèques. C'est le cas par exemple pour faire fonctionner Curses, ReportLab (génération de fichiers PDF) ou encore PyUSB (accès aux ports matériels USB).

Dans ce cas-là, la commande **./configure** devra recevoir des arguments supplémentaires et vous devrez trouver un tutoriel en ligne pour vous indiquer la démarche, laquelle peut être plus ou moins complexe.

2.5 Pour un smartphone

Installer une machine virtuelle Python sur un smartphone est possible. Pour Android, la procédure est assez simple puisqu'il existe un produit dédié (<http://qpython.com/>), tout comme sur Windows Phone (<https://apps.microsoft.com/store/detail/python-39/9P7QFQMJRFP7>). Pour iOS, c'est une autre paire de manches (<https://github.com/linusyang/python-for-ios>) étant donné que l'utilisateur est enfermé dans un système sur lequel il n'a aucun contrôle.

3. Installer une bibliothèque tierce

■ Remarque

Si vous abhorrez le terminal, sachez que vous pouvez installer une bibliothèque tierce depuis votre IDE, ce qui sera probablement plus aisé pour vous.

3.1 À partir de Python 3.4

Pour installer une bibliothèque tierce, vous devez simplement connaître son nom. Celui-ci est généralement assez intuitif. Par exemple, la bibliothèque permettant de communiquer avec un serveur Redis s'appelle `redis`.

Il peut y avoir des variations. Par exemple, la bibliothèque de référence pour traiter du XML est `lxml` et, plus complexe, celle pour BeautifulSoup est `bs4`. En recherchant comment répondre à un besoin sur le Net ou sur PyPi (<https://pypi.python.org/pypi>), vous trouverez rapidement une bibliothèque de référence.

Sur des sujets plus confidentiels, il vous arrivera de trouver plusieurs petites bibliothèques. Vous pourrez alors les tester et choisir celle que vous utiliserez pour votre projet.

Sachez que vous pouvez aussi conduire une recherche directement depuis votre terminal :

```
■ $ pip search xml  
■ $ pip search soup
```

Cela vous donnera une liste de bibliothèques accompagnée d'une courte description, à la manière de ce que font les gestionnaires de paquets sous Linux (lesquels sont écrits en Python, au passage).

Sachez que **pip** existe quel que soit votre système d'exploitation (vous devez être familier avec le terminal de votre système, cependant) et que depuis la version 3.4 de Python, il est installé automatiquement avec celui-ci. Si ce n'est pas votre cas, consultez la section suivante : Pour une version inférieure à Python 3.4.

pip est un outil formidable. Si vous utilisez une version de Python qui est celle du système, vous utiliserez alors la commande **pip** pour gérer les bibliothèques. Si vous utilisez une autre version, telle que Python 3.5, alors vous utiliserez la commande **pip-3.5**. Pour Python 3.3, ce sera **pip-3.3**. Dans les exemples suivants, il vous faudra prendre en compte cette particularité.

Cet outil vous permettra d'installer une bibliothèque à sa dernière version ainsi que toutes les bibliothèques dépendantes. En effet, il n'est pas rare qu'une bibliothèque de Python ait besoin d'une autre bibliothèque (ou de plusieurs) pour fonctionner. Par exemple, l'installation de `redis` se fait par cette commande :

```
■ $ pip install redis
```

On peut aussi choisir la version à installer :

```
■ $ pip install -Iv redis==2.10.5
```

Ou mettre à jour la bibliothèque à une version précise :

```
■ $ pip install -U redis==2.10.5
```

Ou à la dernière version :

```
■ $ pip install -U redis
```

Et on peut la désinstaller :

```
■ $ pip uninstall redis
```

Une fonctionnalité très importante permet d'obtenir la liste des bibliothèques déjà installées (quelle que soit la manière dont elles ont été installées) :

```
■ $ pip freeze
```

Ce que l'on peut mettre dans un fichier :

```
■ $ pip freeze > requirements.txt
```

Pour installer tous les paquets ainsi listés, il faut procéder ainsi :

```
■ $ pip install -r requirements/base.txt
```

Cette méthode est particulièrement utile dans le cadre d'un environnement virtuel ; nous y reviendrons.

Il est possible de retrouver des informations sur un paquet déjà installé :

```
■ $ pip show django-redis
---
Name: django-redis
Version: 4.3.0
Location: /path/to/my/env/lib/python3.4/site-packages
Requires: redis
```

On voit ici que le paquet **django-redis** a une dépendance vers **redis** : en l'installant, on installe automatiquement **redis**.

Mettre à jour ce paquet met à jour automatiquement les dépendances :

```
■ $ pip install -U django-redis
```

Si on ne veut pas mettre à jour les dépendances, on peut procéder ainsi :

```
■ $ pip install -U --no-deps django-redis
```

On peut aussi installer plusieurs bibliothèques en même temps :

```
■ $ pip install django-redis==4.3.0 bs4 lxml
```

Cette commande installera donc automatiquement redis s'il n'est pas installé, car il est déclaré comme dépendance.

Cette commande a cependant des limites. En effet, si vous installez une bibliothèque tierce qui utilise une bibliothèque C, vous devrez disposer des en-têtes C correspondants (paquets **dev** pour Debian ou **devel** pour Fedora). Il faut donc avoir un peu de pratique dans ce genre de situation pour savoir déjouer ces pièges.

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