Programmation de spécialité (python)
TD 3 : acquisition de données

Julien Velcin

2025-2026*

L'objectif principal de cette suite de travaux dirigés est de mettre au point un moteur de recherche d'information maison. Il ne s'agit donc pas d'utiliser l'une des (nombreuses) librairies existantes mais d'implémenter sa *propre solution*. C'est ainsi l'occasion de mettre en œuvre les connaissances acquises dans le cours tout en acquérant une meilleure compréhension des techniques souvent cachées derrière les librairies "prêtes à l'emploi" (par ex. scikit-learn ou nltk). Il vous est donc **interdit** de recourir à ces dernières.

Partie 1 : chargement des données

Cette première partie consiste à extraire des documents à partir d'une source externe. Ces documents s'organisent en une collection qu'on appelle un *corpus*. On considère plusieurs sources d'information que l'on aimerait croiser sur un sujet de votre choix que nous appellerons **thématique**. Cette thématique devra pouvoir être simplement identifiée par un ou des mots clés, par exemple : "Coronavirus", "football", "climate" (nous travaillerons en anglais).

Nous allons d'abord simplement récupérer le contenu textuel. Commencez par créer une liste *docs*. Chaque entrée de la liste sera un document (donc un texte pour le moment). Les documents viendront de *deux* sources : Reddit et Arxiv. Vous récupérerez les documents liés à votre thématique au moyen de mots-clés, comme indiqué ci-dessous.

1.1 Reddit: à l'aide de la librairie `praw`, interrogez l'API comme expliqué dans le site web suivant. Quels sont les champs disponibles ? Quel est le champ contenant le contenu textuel ? Alimentez la liste *docs* avec ce texte uniquement. Vous pouvez déjà vous débarrasser des sauts de ligne ($\backslash n$) en les remplaçant par des espaces.

1.2 Arxiv: à l'aide de la librairie `urllib`, interrogez l'API d'Arxiv en vous aidant de l'aide en ligne. Parser les résultat grâce à la libraire `xmltodict`. Quels sont les champs disponibles ? Quel est le champ contenant le contenu textuel ? Alimentez la liste *docs*.

Partie 2 : sauvegarde des données

L'objectif est de ne pas avoir à interroger les APIs à chaque fois qu'on exécute notre programme. Pour cela :

*Édité par Tetiana Yemelianenko

2.1 Créez un tableau de type `DataFrame` de la librairie `pandas`. Instanciez ce tableau avec les textes qui ont été récupérés depuis les APIs. Vous utiliserez trois colonnes : une première colonne contenant l'identifiant unique du texte (par ex. un simple entier naturel), une deuxième colonne contenant le texte et une troisième colonne contenant son origine (reddit ou arxiv).

2.2 Sauvegardez ce tableau sur le disque dans un fichier de type `.csv` avec la tabulation `\t` comme séparateur.

2.3 Ajoutez le code permettant de charger directement ce tableau en mémoire, sans avoir à passer par l'interrogation des APIs.

Partie 3 : premières manipulation des données

Nous allons manipuler simplement notre corpus.

3.1 Affichez la taille de votre corpus, càd le nombre de documents.

3.2 Pour chaque document, affichez le nombre de mots et de phrases. Pour cela, vous utiliserez la fonction `split` en considérant que les mots sont séparés par des espaces et les phrases par des points. Il s'agit bien sûr d'une simplification, des techniques plus avancées ont été développées en TAL.

3.3 Supprimez de votre corpus les documents trop petits, ici qui contiennent moins de 20 caractères.

3.4 Créez une unique chaîne de caractère contenant tous les documents grâce à la fonction `join`. Cette chaîne de caractère vous sera utile dans la suite des TDs.

Programmation de spécialité (python)
TD 4 : première structuration du code

Julien Velcin

2025-2026*

L'objectif de ce TD est de commencer à structurer le code à l'aide de classes et de modules.

Partie 1 : première classe : le Document

Le contenu textuel n'est pas la seule ressource importante du corpus que vous avez constitué. Les documents sont souvent associés à de nombreuses méta-données comme le nom de l'auteur, l'année de publication, des tags, etc. Pour les gérer, nous allons utiliser le paradigme objet.

1.1 Dans un fichier (module) à part, intitulé `Document.py`, créez une classe `Document` contenant les attributs suivants :

- *titre* : le titre du document
- *auteur* : le nom de l'auteur
- *date* : la date de publication
- *url* : l'url source
- *texte* : le contenu textuel du document

1.2 Créez une première méthode, qui va afficher toutes les informations d'une instance. Implémentez ensuite la méthode `__str__(self)` qui affichera une version "digeste" de l'instance (par exemple son titre seulement). Cette méthode sera appelée quand vous exécuterez la fonction `print(doc)`, où `doc` est une instance de document.

1.3 Modifiez le code du TD précédent afin de pouvoir instancier des objets `Document`. Attention, il faut bien penser à **importer** le module dans le fichier principal. Pour la date de publication, vous pouvez la considérer comme une chaîne de caractère `str` mais le mieux serait de la traiter à l'aide de la librairie `datetime`.

Il faut créer une collection qui doit contenir toutes les instances que vous aurez créées, quelle que soit leur origine. Pour cela, à la place de la liste, vous utiliserez un *dictionnaire* qui va **indexer** les documents, c'est-à-dire les associer à un identifiant unique (leur *clef*) que vous générez au fur et à mesure. Vous appellerez ce dictionnaire d'indices *id2doc* dont les clés sont les identifiants et les valeurs les objets qui représentent les documents.

*Édité par Tetiana Yemelianenko

Partie 2 : Prise en compte des auteurs

Les auteurs aussi sont associés à des méta-données intéressantes. Vous allez implémenter une classe pour gérer ces informations.

2.1 Créez une classe *Author* contenant les attributs suivants:

- `name` : son nom
- `ndoc` : nombre de documents publiés
- `production` : un dictionnaire des documents écrits par l'auteur

Mettez cette nouvelle classe dans un fichier (module) à part, intitulé `Author.py`.

2.2 Créez une première méthode `add` propre à la classe `Author`, qui va permettre d'alimenter l'attribut `production`. Cet attribut permet de compiler l'ensemble des documents rédigés par un auteur. Implémentez ensuite les méthodes `__str__(self)`

2.3 Dans le fichier principal, créez un dictionnaire *id2aut* contenant les auteurs avec comme clef leur nom que l'on considèrera unique. Pour l'alimenter, vous devrez ajouter une instance d'auteur à chaque fois que vous observerez un nouvel auteur au moment d'alimenter le dictionnaire *id2doc*. Vous devrez vérifier si l'auteur est connu, c'est-à-dire si son nom ne fait pas déjà partie des clefs du dictionnaire.

2.4 Ajoutez le code nécessaire pour pouvoir demander quelques statistiques concernant un auteur entré par l'utilisateur : nombre de documents produits et taille moyenne des documents.

Partie 3 : Création du corpus

Il s'agit à présent de structurer le code du fichier principal à l'intérieur d'une classe dédiée à la gestion d'un corpus. Cela peut permettre de gérer *plusieurs* corpus ou un corpus qui évolue dans le temps.

3.1 Créez une classe `Corpus`, qui sera implémentée dans son propre fichier `Corpus.py`, à partir des méthode d'ajout des questions précédentes et contenant les attributs suivants :

- *nom* : le nom du corpus
- *authors* : le dictionnaire des auteurs
- *id2doc* : le dictionnaire des documents
- *ndoc* : comptage des documents
- *naut* : comptage des auteurs

3.2 Implémentez les méthodes permettant d'afficher les élément du corpus triés selon la date et le titre. Les méthodes prennent en paramètre le nombre de documents que vous voulez afficher. Implémentez la méthode `__repr__(self)`, qui permet un affichage plus digeste.

3.3 Créez enfin des méthodes *save* et *load* qui permettent d'enregistrer et charger le corpus sur le disque dur. Pour cela, enrichissez le code que vous avez réalisé dans le TD précédent afin de passer par une table `DataFrame`.

Il est bien sûr possible d'utiliser d'autres formats, comme par exemple `pickle` ou `json`. N'hésitez pas à les tester (optionnel).

3.4 Essayez de sauvegarder et charger vos données pour vous assurer que toute la chaîne de traitement est opérationnelle.

Programmation de spécialité (python)
TD 5 : héritage et première interface

Julien Velcin

2025-2026*

Les deux types de documents que vous avez intégrés à votre corpus (*post* de Reddit et articles d'Arxiv) possèdent de nombreux points communs : un titre, un auteur, une date, etc. Ils possèdent également un certain nombre de différences. Par exemple, les articles Reddit déclenchent des commentaires et l'API permet de récupérer (entre autres) de récupérer leur nombre. Les articles Arxiv sont le plus souvent écrits par *plusieurs auteurs* et il est possible d'en récupérer la liste. Dans cette partie, on vous demande de mettre en place un mécanisme d'héritage en implémentant les deux classes filles `RedditDocument` et `ArxivDocument`. On vous conseille d'implémenter ces deux nouvelles classes dans le fichier `Document.py`, à la suite de la classe mère.

Partie 1 : classe RedditDocument

1.1 Créez une première classe fille `RedditDocument` qui hérite de `Document`. Cette classe doit intégrer au moins une nouvelle variable typique aux documents Reddit et qu'on ne retrouve pas dans les autres sources que vous manipulez. Il s'agit par exemple du nombre de commentaires postés par les internautes en réaction au message. Il faudra notamment implémenter :

- un constructeur, en pensant à appeler le constructeur de la classe mère (en utilisant si possible le mot-clef `super()`),
- des accesseurs/mutateurs pour les champs spécifiques à la classe,
- une méthode spécifique pour s'occuper de l'affichage de l'objet, via la méthode `__str__`.

Partie 2 : classe ArxivDocument

2.1 De la même manière, créez une seconde classe fille `ArxivDocument` qui intègre la gestion des co-auteurs des articles comme élément spécifique à la classe.

Partie 3 : mise à jour de la classe Corpus

3.1 Testez votre objet corpus en y ajoutant des objets des deux classes que vous venez de créer. Le polymorphisme doit vous permettre de réaliser cette opération de manière simple, sans avoir à modifier les méthodes de la classe `Corpus`.

*Édité par Tetiana Yemelianenko

3.2 Vous devez pouvoir afficher la liste des articles contenus dans votre Corpus avec la *source* (Reddit ou Arxiv) d'où ils proviennent. Pour cela, il existe plusieurs solutions, mais vous implémenterez celle qui consiste à ajouter un champ `type` à la classe mère `Document` et à implémenter la méthode `getType()` directement dans les classes filles, comme vu en cours.

Partie 4 : Patrons de conception

4.1 Pour le moment, on considère qu'on ne traite qu'un unique corpus. Profitez-en pour tester l'utiliser d'un patron de type `Singleton`.

4.2 Créez un générateur de documents grâce à un patron de conception d'usine (*factory pattern*).

Programmation de spécialité (python)
TD 6 : analyse du contenu textuel

Julien Velcin

2025-2026*

Dans ce TD, nous allons aller plus loin dans l'analyse du contenu textuel de nos documents.

Partie 1 : travail sur les expressions régulières

1.1 Nous allons commencer par utiliser la libraire `re` (*regular expression*). Ajoutez une fonction `search` à la classe *Corpus*, qui retourne les passages des documents contenant le mot-clef entré en paramètre. Pour cela, il vous faudra travailler sur une unique chaîne de caractères qui concatène l'intégralité des chaînes (cf. TD 3). L'idéal serait de ne pas avoir à construire cette chaîne à chaque appel de la fonction `search`, mais une seule fois au moment du premier appel.

1.2 Ajoutez une fonction `concorde` à la classe `Corpus`, qui construit un concordancier pour une expression donnée. Il s'agit d'une légère modification de la fonction `search`. La taille du contexte est fixée par un paramètre en entrée. Vous ferez appel à la librairie *re* et à la librairie *panda* afin de stocker et retourner les résultats obtenus dans un tableau qui doit ressembler à ce qui suit :

contexte gauche	motif trouvé	contexte droit
...oilà un exemple de	texte	trouvé au milieu d...
...euxième exemple de	texte	trouvé ailleurs ma...

Partie 2 : quelques statistiques

Nous allons maintenant implémenter une méthode affichant plusieurs statistiques textuelles sur le corpus (appelée *stats*). Elle doit afficher:

- Le nombre de mots différents dans le corpus
- Afficher les n mots les plus fréquents (n est un paramètre)

Pour cela, vous devrez suivre les instructions suivantes :

2.1 Tout d'abord, vous devez implémenter une fonction `nettoyer_texte` qui prend une chaîne de caractères en entrée et lui applique une chaîne de traitements. Il faut à minima implémenter les traitements suivants : mise en minuscules (via la fonction `lower()`), remplacement des passages à la ligne $\backslash n$. Vous pouvez aussi remplacer les ponctuations et les chiffres à l'aide d'expressions régulières appropriées.

*Édité par Tetiana Yemelianenko

2.2 En bouclant sur les documents de votre corpus, vous devez construire vous même le *vocabulaire* qui sera utilisé pour décrire les textes de vos documents. Pour cela, vous utiliserez la fonction `split` en considérant plusieurs types de délimitation possible pour l'anglais (espace, tabulation, signe de ponctuation...). Le vocabulaire doit être stocké dans un dictionnaire, mais vous pouvez passer avant par un ensemble (`set`) afin d'éliminer facilement les doublons.

2.3 Pour finir, il faut compter le nombre d'occurrences de chacun des mots de votre vocabulaire en parcourant à nouveau la liste de vos documents. Parcourir deux fois le corpus n'est évidemment pas le plus efficace, donc n'hésitez pas à chercher une solution qui vous évite cette perte de temps. L'idéal est de construire un tableau `freq` avec la librairie `pandas`.

2.4 Vous pouvez enrichir le tableau `freq` en ajoutant une colonne indiquant le nombre de documents (*document frequency*) qui contiennent chacun des mots. Ce n'est pas la même chose que le nombre d'occurrences total des mots (*term frequency*).