**NLP - Projet 4 – Reconnaissance de références juridiques**

Ce projet vise à développer une solution de traitement automatique du langage naturel (NLP) permettant d’identifier et de baliser les références à d'autres textes juridiques dans un corpus de documents HTML. L'objectif est de détecter avec précision les mentions de lois, décrets, arrêtés, circulaires et autres textes réglementaires, puis de les encadrer avec des balises <a> et </a> pour faciliter leur identification et leur navigation.

Pour cela, deux approches ont été considérées :

1. Utiliser des LLMs pré-entrainés, et les fine-tuner pour notre tâche de token classification.
2. Développer des RegEx pour extraire les références à des documents.

Le code est disponible sur mon GitHub : [AlexiaALLAL/nlp\_doc\_recognition\_project](https://github.com/AlexiaALLAL/nlp_doc_recognition_project/)

# Utilisation de LLM préentraînés et fine-tuning

## Constitution du jeu de données :

La plus grande difficulté rencontrée lors de ce projet a été l’absence de documents annotés pour entrainer les modèles. Dans un premier temps, j’ai donc tenté d’annoter (de rajouter les balises <a> et </a>) à la main quelques documents. Ces documents annotés sont accessibles dans le fichier **data\_annoted**.

Cependant, cela prenait beaucoup de temps, et seuls quelques documents ont été annotés. Ce problème sera réglé par la suite en utilisant les fichiers annotés par regex, avec la méthode décrite dans la partie 2. Cela m’a permis d’entrainer les LLMs sur le jeu de données complet.

## Pré-traitement et tokenisation

J’ai tout d’abord commencé par créer des fonctions pour :

* **get\_labels\_from\_annotation** : Récupère les labels (0 ou 1) à partir d’un texte contenant des balises <a> et </a>
* **read\_annoted\_data** : Lit tous les fichiers html du fichier data, ne garde que la partie entre les balises <header class="dsr-header"> et </header> (qui contient toutes les références à identifier), récupère les labels correspondants, et les séparer en chunks de 100 ou 250 mots/labels (pour la tokenisation)
* **build\_data** : Récupère tous les chunks de mots/labels, et les sépare en train, validation et test datasets. Retourne un objet **datasets.DatasetDict**, contenant 3 objets **dataset.Dataset**, dont les features sont ['id', 'tokens', 'ner\_tags'].

J’ai ensuite utilisé plusieurs modèles différents, chacun associés à leur tokenizer.

## Modèles pré-entrainés

Le prétraitement et tokenisation à partir de là a été trouvée dans ce tutoriel de Hugging Face : [Token classification](https://huggingface.co/docs/transformers/tasks/token_classification#token-classification), que j’ai adapté à notre base de données.

Trois LLMs pré-entrainés ont été utilisés :

* DistilBERT : [distilbert/distilbert-base-uncased · Hugging Face](https://huggingface.co/distilbert/distilbert-base-uncased) (67M paramètres). Malgré une taille réduite de paramètres, l’implémentation n’a pas donné de bons résultats. Le modèle ayant été entrainé en anglais, il semble nécessaire de trouver d’autres modèles.
* XLM-RoBERTa : [FacebookAI/xlm-roberta-base · Hugging Face](https://huggingface.co/FacebookAI/xlm-roberta-base). Ce modèle a été entrainé dans 100 languages, comprenant le français (279M paramètres)
* CamemBERT : [almanach/camembert-base · Hugging Face](https://huggingface.co/almanach/camembert-base) : Un modèle entrainé en français (110M paramètres)

Aucun de ces modèle n’a donné de résultat concluant, à cause du manque de données sur lesquelles les entrainer (je n’ai annoté à la main que quelques fichiers).

J’ai décidé de garder le modèle CamemBERT de par son entrainement spécialement en français, et son nombre de paramètres plus faible que XLM-RoBERTa.

### Entrainement sur les fichiers annotés par Regex

L’approche avec les LLMs ne donnant pas de résultats concluants, je me suis rabbatue sur une méthode par regex, décrite dans la partie II. Une fois les documents annotés convenablement par cette méthode, j’ai pu réentraîner le LLM CamemBERT.

Au bout d’une seule epoch, il prédit déjà autre chose que des zéros. Voici les performances au bout de 25 epochs :

* Precision : 0.872117
* Recall : 0.773234
* F1-Score : 0.819704
* Accurracy : 0.957103

Le test sur le premier chunk d’un fichier pris au hasard donne ceci :

<header class="dsr-header"> <div class="dsr-entity"> <div> Préfecture </div> <div> Direction des Collectivités Locales et des Produits Publics </div> <div> Bureau des Foyers Publics et Installations Classées </div> </div> <div class="dsr-identification"> <h1> **<a>ARRÊTÉ du 20 AVR. 2020</a>**\*\* portant autorisation d'exploiter une unité de valorisation énergétique de combustibles solides de récupération (CSR), de déchets d'activité économique (DAE) et d'ordures ménagères (OM) sur le territoire de la commune de Bantzenheim à la société B+T ÉNERGIE France Sas en référence au titre VIII du livre I et au titre I° du livre V du **<a>code de l</a>**'environnement </h1> </div> <div class="dsr-visa"> VU le **<a>code de</a>**

Ca a l’air de très bien fonctionner ! Mais il y a tout de même quelques erreurs, et la mise en page n’est pas conservée par rapport aux regex. C’est donc l’approche par regex que j’ai choisi de garder, car elle me semble plus robuste. L’approche par LLM est cependant très intéressante, et pourrait être très prometteuse en entrainant les modèles avec encore plus de données, et dont les annotations auraient été vérifiées pour corriger d’éventuelles erreurs commises par les regex.

# Regex

En complément de l'approche par fine-tuning de modèles de langage, qui n’a pas donné de résultats satisfaisants, j’ai développé une méthode basée sur des expressions régulières (regex) pour extraire les références juridiques. Cette approche repose sur l'identification de structures récurrentes dans les documents HTML et leur modélisation à l'aide de règles formelles permettant une extraction robuste et explicable.

## Extraction des structures pertinentes

L'analyse des documents a révélé que les références aux textes juridiques se trouvaient majoritairement dans deux types de divisions HTML :

* **La div d’identification** : Contenant la référence du document, elle contient toujours exactement une seule référence.
* **Les div "visa"** : Pouvant contenir une référence à d'autres textes, mais pouvant également être absentes ou vides.

J’ai donc commencé par extraire systématiquement ces div dans chaque document dans les fonctions **extract\_identification** et **extract\_visa**. Cela a permis de restreindre la recherche aux parties les plus pertinentes, et de rendre le système plus robuste.

## Définition des Regex

Une fois ces sections extraites, j’ai observé la liste des div d’identifications et des div visa (fonctions **get\_list\_identification** et **get\_list\_visa**). Cette analyse m’a permis d’identifier des structures récurrentes, implémentées dans les fonctions **add\_tags\_identification** et **add\_tags\_visa**.

À partir de ces observations, j’ai construit une liste exhaustive de regex, puis j’ai vérifié leur efficacité en les appliquant à l’ensemble des div d’identification et des div "visa" (**test\_all\_identification** et **test\_all\_visa**). J’ai progressivement affiné ces expressions pour m’assurer qu’elles capturaient correctement toutes les références dans la base de documents.

## Réécriture des fichiers avec balisage

Une fois les regex validées, la dernière étape a consisté à réécrire chaque fichier HTML en ajoutant les balises <a> autour des références détectées, grâce à la fonction **create\_annoted\_files**. Les fichiers annotés sont créés dans le fichier **data\_4/annoted\_regex**.

## Erreurs identifiées

L’approche par regex a permis d’obtenir une très bonne précision sur tous les documents, avec une identification quasi parfaite de toutes les références. Cependant, j’ai noté quelques erreurs :

* A cause de la normalisation (pour éviter les problèmes d’accents), « N° » devient « Ndeg ».
* On perds aussi les accents de la référence identifiée dans le fichier annoté.
* Une des regex capture « ARRETE PREFECTORAL COMPLEMENTAIRE modifiant l'arrete prefectoral ndeg994156 du 21 decembre 1999 » car elle voit « ARRETE PREFECTORAL … » suivi d’une date. Il aurait fallu ne garder que « ARRETE PREFECTORAL COMPLEMENTAIRE »

Si plus de temps avait été alloué au projet, il aurait été intéressant de regarder chaque fichier annoté, et de repérer d’autres erreurs potentielles pour affiner les regex.