Faculté des **Langues**
Université de Strasbourg

Master en Traduction et interprétation : Technologies des langues
Parcours : Informatique-Traduction

2021-2023

Constitution de corpus : publier au format XML-TEI des dictionnaires et des encyclopédies de la Chine ancienne

Zijun KONG (https://orcid.org/0009-0000-3138-7293)

Mémoire

Sous la direction de
Marie Bizais-Lillig, Maître de conférences
Pablo Ruiz Fabo, Maître de conférences

Remerciements

Dans cette section de remerciements, je tiens tout d'abord à exprimer ma profonde gratitude envers l'ensemble de mes professeurs de Master, qui m'ont accompagné tout au long de mon parcours universitaire et ont contribué à mon épanouissement académique. Je tiens à souligner l'investissement exceptionnel de Monsieur Pablo Ruiz Fabo. Son assistance et son dévouement envers mon développement académique et professionnel ont sans aucun doute eu un impact significatif sur ma réussite.

Par ailleurs, je remercie sincèrement Madame Marie Bizais-Lillig. Ses conseils précieux et son assistance dans la rédaction de mon mémoire ont été des facteurs clés dans l'aboutissement de ce travail. Ses suggestions judicieuses et ses retours constructifs ont été de véritables tremplins pour l'élaboration de mon travail de recherche. Je lui suis reconnaissant pour son accompagnement patient et pour avoir mis en lumière des perspectives inédites qui ont enrichi ma recherche.

De plus, j'adresse mes sincères remerciements à Madame Delphine Bernhard pour son enseignement méticuleux et ses précieux cours en apprentissage automatique qui m'ont été très utiles. Mes remerciements vont également à Madame Amalia Todirascu pour son cours sur le langage XML qui a grandement facilité la rédaction de mon mémoire.

Enfin, mes remerciements vont également à mes camarades de classe. Leur amitié tout au long de ce parcours a créé un environnement stimulant et positif. Ils ont été une source de motivation et d'encouragement dans les moments difficiles, et leurs idées ont souvent éclairé les voies de réflexion les plus complexes.

Je tiens également à exprimer ma profonde gratitude à mes parents et à mon compagnon qui m'ont accompagné tout au long de ce parcours. Leur présence et leur soutien indéfectible m'ont permis de mener à bien ce travail dans les meilleures conditions. C'est grâce à leurs encouragements que j'ai pu surmonter les difficultés et finaliser cette recherche qui me tenait à cœur.

C'est avec une grande humilité que j'exprime ma reconnaissance pour votre dévouement et votre investissement dans ma formation. Vous avez tous joué un rôle majeur dans la réalisation de ce mémoire et dans mon parcours académique.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

La numérisation du chinois classique, une langue ancienne mais essentielle pour comprendre la culture chinoise, pose un ensemble complexe de défis. La reconnaissance optique des caractères chinois est rendue difficile par le fait que le chinois classique n'est plus couramment utilisé. De plus, la codification des nombreux caractères chinois est un défi, avec la norme Unicode ne couvrant pas tous les caractères chinois existants. En outre, les outils existants pour traiter les textes chinois classiques numérisés sont limités, avec peu de modèles pré-entraînés dédiés au chinois classique disponibles dans les outils de TAL tels que spaCy.

Face à ces problèmes, plusieurs questions de recherche se posent : Comment peut-on sélectionner et prétraiter les données textuelles appropriées pour construire un corpus de dictionnaires et d'encyclopédies en chinois classique au format TEI ? Comment développer et optimiser des méthodes automatisées pour traiter les textes en chinois classique, notamment l'annotation de séquences et la reconnaissance d'entités nommées ? Et enfin, comment peut-on concevoir et mettre en œuvre une structure de corpus TEI adaptée aux textes en chinois classique ?

Notre projet vise à proposer une manière de répondre à ces questions en prenant comme cas d'usage la constitution d'un corpus bien annoté de dictionnaires et d'encyclopédies chinois ancien respectant les normes de la XML-TEI. Notre travail comprend l'acquisition et le traitement du texte, la modélisation et le développement du corpus, et le développement de méthodes de traitement du texte. Nous avons également utilisé un code dédié à la relecture du corpus pour assurer la qualité et l'exactitude de notre travail.

Ce mémoire s'inscrit dans le cadre du projet académique CHI-KNOW-PO dont l'objectif est de numériser un corpus d'anthologies poétiques, de commentaires, de dictionnaires et d'encyclopédies de la période médiévale chinoise, de le publier et de développer des outils d'exploration de ce corpus. Le volet de numérisation qui m'a été confié concerne deux œuvres significatives : le *Shuowen Jiezi* et le *Yiwen Leiju*.

Notre objectif est que le corpus développé dans le cadre du mémoire sera une ressource précieuse pour la recherche sur le chinois classique et pourra être utilisé librement pour la recherche et la republication.

2 État de l'art

Dans cette section, nous présentons les technologies sous-jacentes sur lesquelles repose la création de corpus ainsi que la chaîne de traitement associée. La section 2.1. décrit la *Text Encoding Initiative*, la section 2.2 se concentre sur les sources de corpus pertinentes pour le mémoire et la section 2.3 aborde les techniques de balisage automatique de corpus électroniques. Finalement, la section 2.4 décrit rapidement certaines pratiques de projets d'encodage TEI.

2.1 TEI

Progressivement, la TEI est devenue une norme populaire pour l'encodage des textes. En tant que spécification entièrement basée sur XML, la TEI hérite pleinement des avantages de ce langage tels que la grande extensibilité ou le système de balisage basé sur la structure. Parallèlement, la TEI propose également de nombreuses optimisations pour l'encodage des textes avec des systèmes de balises prédéfinies offrant une grande variété de balises pour presque tous les types de texte. Ce système permet notamment aux petits projets de démarrer immédiatement, sans nécessiter de travail supplémentaire. Dans la suite, nous décrirons en détail les caractéristiques de cette norme TEI.

2.1.1 Une grande variété de balises prédéfinies

La Text Encoding Initiative (TEI) a été élaborée en prévision d'un large éventail de scénarios d'application. Elle propose une multitude de balises prédéfinies et offre des recommandations officielles (TEI Guidelines [1]) détaillées pour leur utilisation, incluant des exemples de codage et des explications sur les relations d'imbrication entre différents éléments.

Le jeu de balises de la TEI se divise en deux principales catégories : les balises d'en-tête et les balises de corps. Ces dernières correspondent aux deux parties du corpus TEI, soit l'en-tête pour les métadonnées et le corps pour le contenu principal. L'en-tête peut inclure diverses métadonnées, telles que le titre du document, l'auteur, les informations de publication, la description de l'encodage, la langue, l'historique des révisions, etc.

La TEI offre une grande flexibilité dans l'usage des balises d'en-tête, permettant l'inclusion d'informations correspondant à des métadonnées essentielles et optionnelles. Cela explique pourquoi la plupart des projets basés sur la TEI, même s'ils personnalisent leur jeu de balises, conservent en grande partie la section standardisée de l'en-tête.

Quant aux balises de texte, la TEI propose une vaste gamme subdivisée en deux types : les balises de base (*core*) et les balises étendues. Les balises de base sont les éléments essentiels d'encodage de texte. La TEI met également à disposition un ensemble de balises étendues pour divers genres de texte, par exemple pour les projets de dictionnaires.

1. URL : https ://tei-c.org/guidelines/P5/ (accès le 1er août 2023).

2.1.2 Un système de personnalisation

La TEI est un projet entièrement basé sur la communauté que forment ses utilisateurs, et grâce aux efforts de celle-ci, l'ensemble de ses balises prédéfinies devient de plus en plus riche. Cependant, cela soulève un problème : en réalité, la plupart des projets n'a pas besoin d'utiliser toutes les balises. Si tous les utilisateurs adoptaient l'ensemble complet des balises de la TEI, cela rendrait sans doute les projets trop volumineux. Heureusement, la TEI a pris en compte la caractéristique de personnalisation dès le début de sa conception.

Une innovation proposée par la norme TEI est l'ODD (*One Document Does it all*). Les utilisateurs peuvent définir leurs propres balises et les relations de hiérarchie entre elles grâce à l'ODD. Le fichier ODD généré peut être converti pour différentes utilisations, couvrant tous les besoins en codage de documents. Par exemple, les utilisateurs peuvent utiliser des outils spécifiques (un outil appelé « Roma » est fourni par TEI Consortium) pour convertir le fichier ODD en fichiers de schéma tels que Relax NG ou XSD, qui peuvent être ensuite utilisés pour valider le fichier TEI. De plus, étant donné que le fichier ODD est un document hautement lisible en soi (grâce à son utilisation de la syntaxe XML), il fournit même une description précise du schéma.

La présence de l'ODD, en combinaison avec les balises prédéfinies fournies par la TEI, rend ce dernier adapté à presque tous les projets d'encodage de texte. Une approche viable consisterait à analyser d'abord l'adéquation entre l'ensemble de balises officielles de la TEI et les besoins du projet. Dans la plupart des cas, la version simplifiée de TEI, connue sous le nom de TEI Lite, est suffisante. Ensuite, en fonction de cette adéquation, il peut être déterminé si l'utilisation de l'ODD pour créer un ensemble de balises personnalisées est nécessaire. Même si l'utilisateur estime qu'il n'est pas nécessaire de modifier l'ensemble de balises prédéfinies, la présence de l'ODD reste bénéfique, que ce soit pour l'extension dans la future ou pour simplifier la validation des fichiers.

Nous allons utiliser un exemple pour illustrer la fonctionnalité de l'ODD. Supposons que nous devions numériser une nouvelle d'un journal, qui comprend quatre parties : le titre, l'auteur, la date de publication et le corps du texte. Nous utilisons le fichier ODD suivant, qui peut ensuite être utilisé à des fins diverses :

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>ODD exemple</title>
      </titleStmt>
      <publicationStmt>
        </p>
      </publicationStmt>
    </fileDesc>
  </teiHeader>
  <text>
```

```
    <body>
      <schemaSpec ident="simple_news" prefix="tei_" docLang="en" start="TEI">
        <moduleRef key="core"/>
        <moduleRef key="header"/>
        <moduleRef key="tei"/>
        <classSpec ident="teiHeader" mode="change">
          <attList>
            <attDef ident="type" usage="opt">
              <valList type="closed">
                <valItem ident="simple"/>
              </valList>
            </attDef>
          </attList>
        </classSpec>
        <elementSpec ident="title" mode="change">
          <attList>
            <attDef ident="type" usage="req">
              <valList type="closed">
                <valItem ident="main"/>
              </valList>
            </attDef>
          </attList>
        </elementSpec>
        <elementSpec ident="text" module="tei" mode="change">
          <content>
            <sequence>
              <elementRef key="title"/>
              <elementRef key="author"/>
              <elementRef key="date"/>
              <elementRef key="p"/>
            </sequence>
          </content>
        </elementSpec>
      </schemaSpec>
    </body>
  </text>
</TEI>
```

Dans le bloc de code ci-dessus, nous fournissons un document ODD concis. Il est facile de voir que le fichier ODD lui-même est un document TEI, qui utilise la même structure que les documents TEI ordinaires, c'est-à-dire une structure de `<teiHeader>` + `<text>`. Par conséquent, son en-tête contient des métadonnées (pour les fichiers ODD, cette partie est souvent moins important). Nous mettrons l'accent sur l'élément `<body>`, le sous-élément de `<text>`, où se trouve toutes les modifications.

Dans cet élément, nous utilisons d'abord un élément `<schemaSpec>` pour définir un schéma avec l'identifiant `simple_news`. Dans ce schéma, tous les éléments doivent

avoir le préfixe `tei_`, la langue principale de ce schéma est l'anglais et l'élément de début de ce schéma est TEI.

Ensuite, à l'intérieur de `<schemaSpec>`, nous avons utilisé un total de trois types de sous-éléments pour différentes modifications : `<moduleRef>`, `<classSpec>` et `<elementSpec>`.

L'élément `<moduleRef>` fournit une référence aux modules officiels de TEI. Avec cet élément, nous pouvons utiliser directement les balises prédéfinies de TEI. Dans cet exemple, nous utilisons trois éléments `<moduleRef>`, qui font référence respectivement aux modules `core`, `header` et `tei`. Cela signifie que nous pouvons utiliser des éléments tels que `<ref>` et `<note>` dans notre projet, car ces éléments sont inclus dans les modules que nous avons référencés.

Ensuite, nous avons utilisé l'élément `<classSpec>` pour modifier la classe `teiHeader`. Nous lui avons donné un attribut type, qui est facultative et dont la valeur ne peut être que « simple ».

Après cela, nous avons utilisé l'élément `<elementSpec>` à deux reprises pour modifier certains éléments spécifiques, tels que l'élément `<title>`. Après cette modification, cet élément doit avoir un attribut type et la valeur de cet attribut doit être « main ».

2.1.3 TEI Lex-0

TEI Lex-0 [2] est un projet lexicologique basé sur le projet MONK (Pytlik Zillig, 2009). L'objectif de ce projet communautaire est de créer un ensemble de recommandations pour améliorer la lisibilité des dictionnaires par les machines. Ce projet fournit un bon exemple de corpus de dictionnaire en TEI.

Nous présenterons la personnalisation de TEI Lex-0 en deux parties, correspondant respectivement aux métadonnées (`<teiHeader>`) et à l'élément central `<entry>`.

Entête

La balise `<teiHeader>` contient généralement cinq sous-éléments : `<fileDesc>`, `<encodingDesc>`, `<profileDesc>`, `<xenoData>` et `<revisionDesc>`. Ces cinq parties correspondent respectivement à la description du fichier, à la description de l'encodage, à la description du profil, aux métadonnées externes et à la description de révision. Le projet oblige les éditeurs à utiliser les sections `<fileDesc>` et `<profileDesc>` et recommande aux éditeurs d'utiliser un maximum d'autres éléments pour fournir le plus d'informations possible.

`<fileDesc>`

1. Il est obligatoire d'utiliser la balise `<availability>` et `<licence>` dans l'élément `<publicationStmt>` pour déclarer les conditions dans

2. URL : https ://dariah-eric.github.io/lexicalresources/pages/TEILex0/TEILex0.html (accès le 1er août 2023).

lesquelles des tiers peuvent utiliser ce document. De plus, si l'on a utilisé `<autority>`, il faut indiquer le rôle de l'entité dans la balise en ajoutant dans la balise l'attribut `@role`, les valeurs acceptables sont : *funder*, *sponsor* et *rightsHolder*.
2. La balise `<sourceDesc>` est facultative, mais une fois sélectionnée et la source donnée, `<biblStruct>` (*structured bibliographic citation*) doit être utilisé à l'intérieur de celle-ci pour donner les informations bibliographiques concernant la source de référence pour le travail éditorial.

`<profileDesc>`

1. Cet élément est optionnel dans la norme TEI, mais obligatoire dans TEI Lex-0, car pour les dictionnaires, il est incontournable d'enregistrer les langues utilisées.
2. Pour ces raisons, l'élément `<langUsage>` doit apparaître et avoir au moins un sous-élément.

Entrée

`<entry>` est l'élément de base pour construire un dictionnaire, chaque entrée est stockée dans un élément `<entry>` . Pour `<entry>` , deux attributs sont exigés :

1. `xml :lang`, Cet attribut est utilisé pour indiquer la langue à laquelle appartient l'entrée.
2. `xml :id`, Cet élément est utilisé pour mettre un identifiant unique sur l'entrée. Le premier élément à l'intérieur de `<entry>` est `<form>` dans laquelle on stocke le mot, pas directement, mais dans son sous-élément `<orth>`, soit :

```
<entry xml:lang="fr" xml:id="1">
  <form>
     <orth>lancer</orth>
  </form>
</entry>
```

Le deuxième élément de `<entry>` est `<gramGrp>`, qui est utilisé pour indiquer les propriétés grammaticales d'un mot. TEI Lex-0 a apporté une modification contestée dans cette partie : dans le standard TEI, sept balises sont proposées pour décrire les propriétés grammaticales : `<pos>` (*part of speech*), `<case>`, `<gen>` (*gender*), `<iType>` (*iflection type*), `<mood>` (*mode*), `<number>`, `<per>` (*person*) et `<tns>` (*tense*). En revanche, dans la norme TEI Lex-0, les sept éléments sont rassemblés dans un seul élément `<gram>`, et ces propriétés grammaticales sont distinguées par les attributs. Voici une comparaison entre les deux normes :

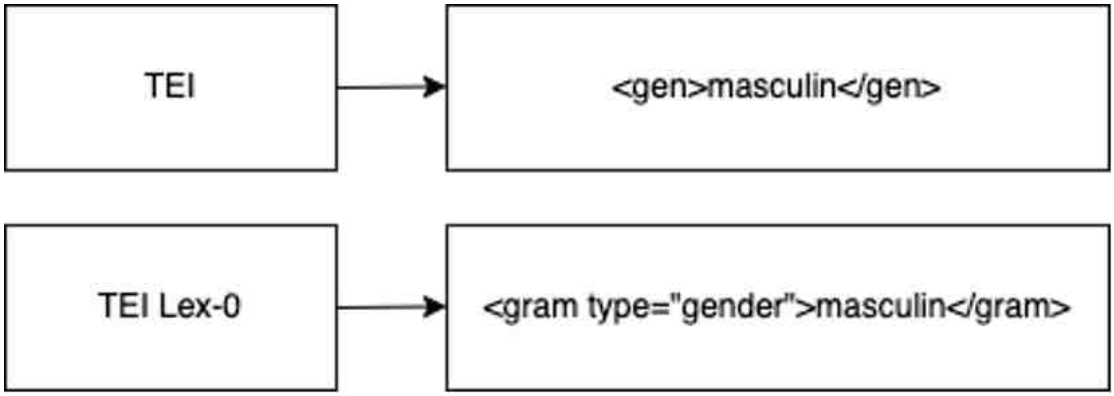

Figure 1 – Différence entre TEI et TEI Lex-0

L'élément obligatoire `<sense>` est dédié à stocker la définition du mot, parmi ses sous-éléments, `<def>` est utilisé comme un conteneur de définition, `<cit>` est pour la citation et il comporte un sous-élément `<quote>` pour stocker les contenus cités. Les deux éléments, `<def>` et `<cit>`, ne peuvent qu'être utilisé comme sous-éléments de `<sense>`.

Dans TEI Lex-0, les références croisées sont effectuées à l'aide des éléments `<xr>` et `<ref>`, ce dernier étant un sous-élément du premier. Quand on utilise `<xr>`, il est obligatoire d'ajouter l'attribut type. L'attribut est aussi exigé dans le sous-élément `<ref>`. Pour `<ref>`, il est recommandé d'ajouter l'adresse de l'objet référencé.

TEI Lex-0 fournit un bon exemple d'encodage numérique pour les projets linguistiques. Il apporte de nombreuses améliorations à la norme TEI afin de faciliter l'encodage numérique des projets comme des dictionnaires, par exemple en ce qui concerne le traitement des références croisées. Cependant, après avoir soigneusement examiné la compatibilité de cette norme avec notre projet, nous avons finalement décidé de ne pas adopter Lex-0. Les raisons sont les suivantes :

1. Notre projet concerne différents types d'œuvres, pas seulement des dictionnaires, et Lex-0 n'est pas adapté aux œuvres telles que les encyclopédies.
2. Même pour la conception d'un dictionnaire, Lex-0 n'est pas compatible avec le dictionnaire utilisé dans notre projet car il contient trop d'éléments qui n'existent tout simplement pas en chinois classique (comme les catégories grammaticales).

Les raisons ci-dessus font qu'il est difficile d'éviter de modifier Lex-0 même si nous l'utilisons, ce qui n'a aucun avantage par rapport à la modification directe du standard TEI.

2.2 Les sources de corpus

La première étape de la création d'un corpus à des fins d'encodage TEI est l'obtention de textes électroniques. Pour cela, nous avons deux options :

1. Transcrire les textes papier existants (à partir d'éditions de ces textes qui appartiennent au domaine public) à l'aide de la reconnaissance optique

de caractères (OCR). Pour notre projet, l'un des grands avantages de cette méthode est d'éviter les problèmes de droits d'auteur, car toutes les œuvres dans ce projet sont déjà dans le domaine public, mais elle présente également deux défis. D'abord, les caractères de l'ancien chinois sont assez complexes et nombreux, rendant l'utilisation de l'OCR très difficile. L'un des principaux défis est la différence de mise en page entre les différentes œuvres et le problème des commentaires. Plus précisément, souvent les commentaires sont présentés sous forme de deux colonnes, ce qui nécessite que le modèle apprenne à traiter deux formes de texte complètement différentes, ce qui augmente la charge de travail.

2. Extraire les textes dont nous avons besoin à partir de bases de données de corpus existantes sur Internet. L'avantage de cette méthode est qu'elle évite le temps de transcription du texte, mais les droits d'auteur du texte appartiennent à la base de données. Par conséquent, notre capacité à utiliser ces textes et à les éditer dépend entièrement de l'attitude du propriétaire de la base de données.

À l'heure actuelle, il existe peu de corpus de textes en chinois classique disponibles gratuitement sur Internet. En outre, très peu de ces corpus ouvrent complètement les droits, y compris la modification et la republication. Nous présenterons deux corpus de textes en chinois classique populaires et comparerons leurs avantages et inconvénients respectifs.

2.2.1 Chinese text project

Le Chinese Text Project [3](abrégé CTP) est un projet lancé en 2006 visant à collecter, rassembler et partager des ouvrages de la littérature en chinois classique, couvrant des domaines tels que la littérature, la philosophie et l'histoire, etc. Ce projet est maintenant un projet communautaire où tout le travail est effectué par des bénévoles.

Le CTP a mis au point un processus complet de numérisation des textes en chinois classique : de la numérisation du texte à la publication pour un usage public, en passant par la construction du corpus.

Tout d'abord, en ce qui concerne l'acquisition du texte, les textes chinois du projet proviennent des universités partenaires. Ces universités fournissent les images des œuvres requises pour le projet. Ensuite, le CTP utilise la reconnaissance optique de caractères pour transcrire le texte.

Après avoir obtenu la transcription du texte, le CTP n'adopte pas le modèle de distribution traditionnel de l'industrie de l'édition. Au lieu de cela, il utilise un modèle de *crowdsourcing* impliquant la participation de toute la communautéSturgeon, 2019. Dans le modèle traditionnel de l'industrie de l'édition, une fois que la transcription du texte est obtenue à l'aide de la technologie OCR, elle est généralement corrigée par des éditeurs, puis annotée avant d'être publiée. Seulement après la publication officielle, le texte devient accessible au public.

Cependant, dans le modèle de crowdsourcing du CTP, une fois que le texte est transcrit

3. URL : https ://ctext.org/ (accès le 1er août 2023).

par OCR, il est immédiatement accessible au grand public. Toutes les étapes suivantes, telles que la correction et l'annotation du texte, sont accomplies en collaboration avec l'ensemble de la communauté (ces étapes peuvent ne pas suivre strictement l'ordre du modèle traditionnel). Cette approche est nécessaire en raison de la taille immense du texte détenu par le site (en 2018, la base de données du site contenait déjà plus de 5 milliards de caractères). Sans faire appel à la puissance de la communauté, il serait impossible de garantir la vitesse de traitement du texte.

L'attitude de CTP en matière de licence est délicate : bien qu'il autorise les utilisateurs à télécharger ou imprimer les textes à des fins personnelles ou de recherche académique non lucratives, la modification du texte et sa republication ne sont pas directement autorisées.

2.2.2 Wikisource

Wikisource est un projet de bibliothèque numérique en ligne géré par la Wikimedia Foundation, qui vise à collecter et à distribuer des œuvres libres de droits. Comme CTP, wikisource est un projet communautaire et utilise un modèle de crowdsourcing, à la différence que Wikisource est ouvert à l'utilisation secondaire de ses textes.

L'avantage de Wikisource sur CTP est sa vision du droit d'auteur libre, et toutes les œuvres collectées sur le site suivent une licence émise par l'organisation Creative Commons. Plus précisément, Wikisource utilise la licence CC BY-SA 3.0. La licence est divisée en deux parties : la première partie BY, qui exige que toute personne apportant des modifications à une œuvre y appose le nom de l'auteur original, et la seconde partie SA (*share alike*), qui stipule que toute version modifiée doit respecter la même licence que l'œuvre originale (c'est-à-dire CC BY-SA 3.0).

Le site offre un large éventail de formats d'exportation, ce qui permet de télécharger rapidement des textes entiers. En outre, Wikisource propose également un certain nombre d'images des glyphes et comme nous le savons, Unicode ne prend pas en charge complètement les caractères du chinois classique. Donc ces images nous fournissent un bon complément.

Nous avons finalement choisi Wikisource comme principale source de corpus, principalement pour deux raisons. Tout d'abord, il n'y a pas de problème de licence avec Wikisource, nous pouvons modifier librement le texte et le republier. Deuxièmement, les textes sur Wikisource ont déjà été modifiés plusieurs fois par la communauté, ce qui est beaucoup plus pratique que de commencer à partir de zéro en utilisant l'OCR pour reconnaître optiquement les livres imprimés. Après tout, notre projet contient certains caractères du chinois ancien qui ne sont pas pris en charge par Unicode. Nous avons exporté tous les textes au format TXT.

2.3 Balisage automatique de corpus électroniques

Après avoir reçu le texte, il est souvent nécessaire de le traiter afin de le préparer. Cette étape implique généralement le nettoyage du texte, la segmentation des mots, l'annotation des parties du discours et d'autres procédures. L'objectif de cette étape est de

démêler le texte et de le décomposer en différentes unités (la division des unités dépend du type de corpus, par exemple, pour un corpus dramatique, les personnages constituent une unité importante). Les méthodes de traitement du texte ont évolué avec le développement de la technologie informatique.

Pour le traitement de texte, nous avons envisagé plusieurs solutions, d'abord une méthode basée sur des règles. Ces méthodes fonctionnent en appliquant manuellement des règles prédéfinies au texte. Les règles peuvent prendre différentes formes, telles que des expressions régulières ou des dictionnaires. Par exemple, pendant la constitution du corpus Sinica, les auteurs ont utilisé un dictionnaire de mots courants pour la segmentation et une règle basée sur la morphologie pour détecter les dérivés et les mots composés (Chen et al., 1996).

Par la suite, avec le développement de l'apprentissage automatique, les chercheurs ont de plus en plus recours aux modèles d'apprentissage automatique pour le traitement des textes. Cette méthode consiste à ajuster une fonction ou un modèle pour essayer de traiter correctement les données inconnues.

Il n'est pas difficile de constater qu'en ce qui concerne les méthodes basées sur des règles, elles présentent un défaut majeur : ces méthodes deviennent plus complexes quand la complexité du texte augmente. Ainsi, le traitement de textes complexes nécessite souvent beaucoup de temps pour écrire les règles et celles-ci peuvent même entrer en conflit entre elles. L'apprentissage automatique permet d'éviter ce problème : Dans les méthodes d'apprentissage automatique traditionnelles utilisées pour le traitement de données textuelles, l'utilisateur annote d'abord le texte, puis l'algorithme apprendra automatiquement la relation de correspondance entre les deux types de données (texte et étiquettes). Dans les tâches ultérieures, le modèle traitera en fonction des connaissances précédemment acquises.

En raison de la diversité des méthodes d'apprentissage automatique, un grand nombre de ces méthodes sont appliquées au traitement du texte. Nous pouvons les classer en trois catégories : les méthodes traditionnelles d'apprentissage automatique, les méthodes d'apprentissage profond et l'apprentissage par transfert à partir de modèles pré-entraînés.

Dans notre projet, nous essayons d'utiliser différentes méthodes d'apprentissage automatique. La raison en est que dans le projet CHI-KNOW-PO, je suis responsable du traitement de la numérisation de deux œuvres (le *Shuowen Jiezi* et le *Yiwen Leiju*) qui sont totalement différentes en termes de genre. Nous pensons qu'il est difficile d'atteindre une performance parfaite dans toutes les œuvres avec une seule méthode, il est donc très nécessaire d'essayer différentes méthodes.

2.3.1 Apprentissage automatique traditionnel

Dans le cadre de leur projet consacré à la création de corpus de théâtre alsacien encodé en TEI, Ruiz Fabo et al. (2023) ont mis à profit la méthode du Champ aléatoire conditionnel (CRF) comme un élément fondamental de leur chaîne de traitement. Cette approche consistait à annoter manuellement une petite portion du texte qui servait ensuite de base d'entraînement pour le modèle CRF. Le modèle ainsi entraîné était par la suite utilisé

pour annoter automatiquement le reste du texte. De plus, des techniques basées sur des dictionnaires et des règles ont été employées pour compléter les annotations du modèle CRF.

Le champ aléatoire conditionnel (Lafferty et al., 2001) est une méthode traditionnelle d'apprentissage automatique, souvent utilisée pour l'annotation de séquences. Le CRF peut apprendre efficacement les caractéristiques d'un texte pour le classifier.

Lors de l'utilisation du CRF pour des tâches d'annotation de séquences, nous devons construire d'abord manuellement les caractéristiques, par exemple les mots précédents et suivants de chaque mot dans une séquences de texte, la position absolue des mots (s'il est au début ou à la fin de la phrase), etc.

Ensuite, l'algorithme crée des fonctions de caractéristiques basées sur les caractéristiques construites. Le modèle comprend généralement plusieurs fonctions de caractéristiques, qui sont généralement binaires (0 ou 1), et l'algorithme détermine si un élément remplit les conditions pour être classé sous une certaine classe à partir de la valeur de ces fonctions.

Ensuite, l'algorithme apprend le poids correspondant à chaque fonction de caractéristique. Lors de l'utilisation du modèle pour faire des prédictions, le modèle donne le résultat final en fonction de la valeur de l'élément dans plusieurs fonctions de caractéristiques et du poids de la fonction.

Le modèle CRF est souvent utilisé pour les tâches d'annotation de séquences, il peut bien gérer les relations de dépendance entre différentes séquences. Bien que les réseaux neuronaux soient de plus en plus utilisés pour la même tâche, le CRF reste encore très important dans les petits projets car ce type de modèle exige moins de données d'entraînement par rapport aux réseaux neuronaux.

Dans notre projet, le modèle CRF est le premier modèle d'apprentissage automatique que nous avons envisagé, car sa légèreté nous permet de vérifier rapidement la faisabilité des méthodes d'apprentissage automatique dans la partie la plus difficile de ce projet, soit l'annotation automatique des séquences.

2.3.2 Apprentissage profond

Les méthodes traditionnelles d'apprentissage automatique sont extrêmement utiles pour traiter des textes bien structurés. Cependant, lorsque nous sommes confrontés à des textes complexes, ces méthodes ont souvent du mal à apprendre les caractéristiques. C'est pourquoi nous privilégions généralement l'utilisation de réseaux neuronaux pour le traitement de tels textes.

Les algorithmes d'apprentissage profond utilisent des structures de réseaux neuronaux qui comportent un nombre élevé de paramètres (allant de centaines de milliers à des milliards). Cette caractéristique les rend bien plus adaptés à certaines tâches que les algorithmes traditionnels d'apprentissage automatique. Cependant, il est important de noter que ces modèles de réseaux neuronaux nécessitent davantage de données et de

temps d'entraînement pour obtenir de bonnes performances.

Un autre avantage des réseaux neuronaux réside dans leur structure hiérarchique, qui nous permet d'ajouter des couches personnalisées afin d'améliorer les performances du modèle. Dans le cadre de ce projet, nous examinons deux structures de modèles : le BiLSTM (Bidirectional Long-Short Term Memory) et le Transformer. Ces deux architectures sont particulièrement efficaces pour capturer les dépendances dans les phrases longues. Le BiLSTM s'appuie sur son mécanisme de « mémoire », tandis que le Transformer utilise le mécanisme de *self-attention*.

Avant de plonger dans les modèles spécifiques d'apprentissage profond, il est nécessaire de comprendre comment cela fonctionne. La classification des textes par la plupart des algorithmes d'apprentissage automatique passe par trois étapes :

1. Prétraitement du texte : cette étape est principalement utilisée pour gérer le bruit dans le texte et le normaliser. Elle consiste à supprimer les éléments inutiles, à convertir le texte en minuscules, à segmenter les mots, etc. Les étapes spécifiques varient en fonction de la langue traitée.

2. Extraction de caractéristiques : cette étape diffère entre les algorithmes traditionnels d'apprentissage automatique et les algorithmes d'apprentissage profond. Dans les premiers, nous devons créer manuellement de nouvelles caractéristiques en fusionnant des caractéristiques existantes. Les modèles de sacs de mots et l'encodage one hot sont des méthodes courantes d'extraction de caractéristiques. En revanche, dans l'apprentissage profond, l'algorithme analyse automatiquement les caractéristiques du texte sans intervention humaine.

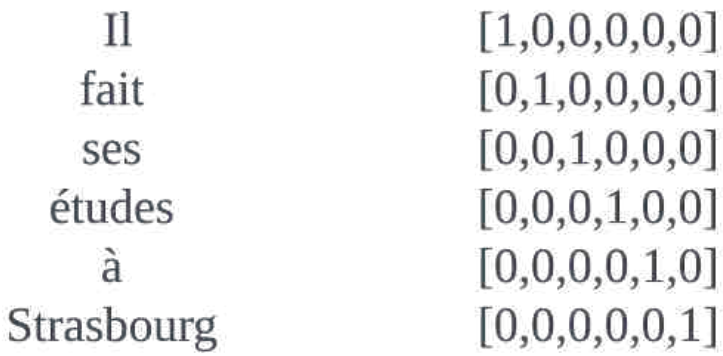

Figure 2 – Encodage *one hot* d'une phrase en français

Dans la figure ci-dessus, nous avons effectué un encodage one hot pour une phrase simple. Cette technique d'encodage consiste tout d'abord à construire un « vocabulaire » qui inclut tous les mots uniques du texte à traiter, puis à créer un vecteur de n dimensions (n étant la longueur du vocabulaire) pour chaque mot. Dans ce vecteur, seul un chiffre est égal à 1 et tous les

autres sont égaux à 0. C'est une technique très facile à mettre en œuvre et à comprendre, mais elle présente certaines limitations importantes. Fondamentalement, l'encodage one hot ne peut pas encoder d'informations liées aux mots, il ne contient pas de sémantique, de grammaire ni d'informations contextuelles, voire la position absolue des mots car la position du 1 dans l'encodage correspond en réalité à la position du mot dans le vocabulaire.

3. Représentation du texte : les algorithmes d'apprentissage automatique ne peuvent pas traiter les caractères directement, nous devons donc les convertir en un format compréhensible pour l'algorithme, c'est-à-dire en vecteurs. Les algorithmes traditionnels réalisent cette étape en parallèle de l'extraction de caractéristiques, en utilisant l'encodage one hot ou les modèles de sacs de mots mentionnés précédemment. En revanche, l'apprentissage profond utilise une méthode de représentation du texte plus complexe et avancée : *word embeddings*.

Les *word embeddings* désignent une classe de techniques. Ces techniques consistent essentiellement à projeter les mots dans un espace multidimensionnel. Tout comme avec l'encodage *one hot*, les mots sont convertis en vecteurs de dimension *n*. Cependant, contrairement au *one hot*, les *word embeddings* sont capables d'encoder des informations sémantiques, grammaticales, etc. Il convient de noter qu'il s'agit d'un encodage relatif. Cela signifie que l'*embedding* d'un seul mot n'a pas de sens, nous ne pouvons pas déterminer ce que ce mot signifie en utilisant son *embedding* correspondant. Ce que font les *word embeddings*, c'est rapprocher autant que possible les mots ayant des significations similaires dans un espace multidimensionnel et éloigner autant que possible les mots ayant des significations opposées les uns des autres.

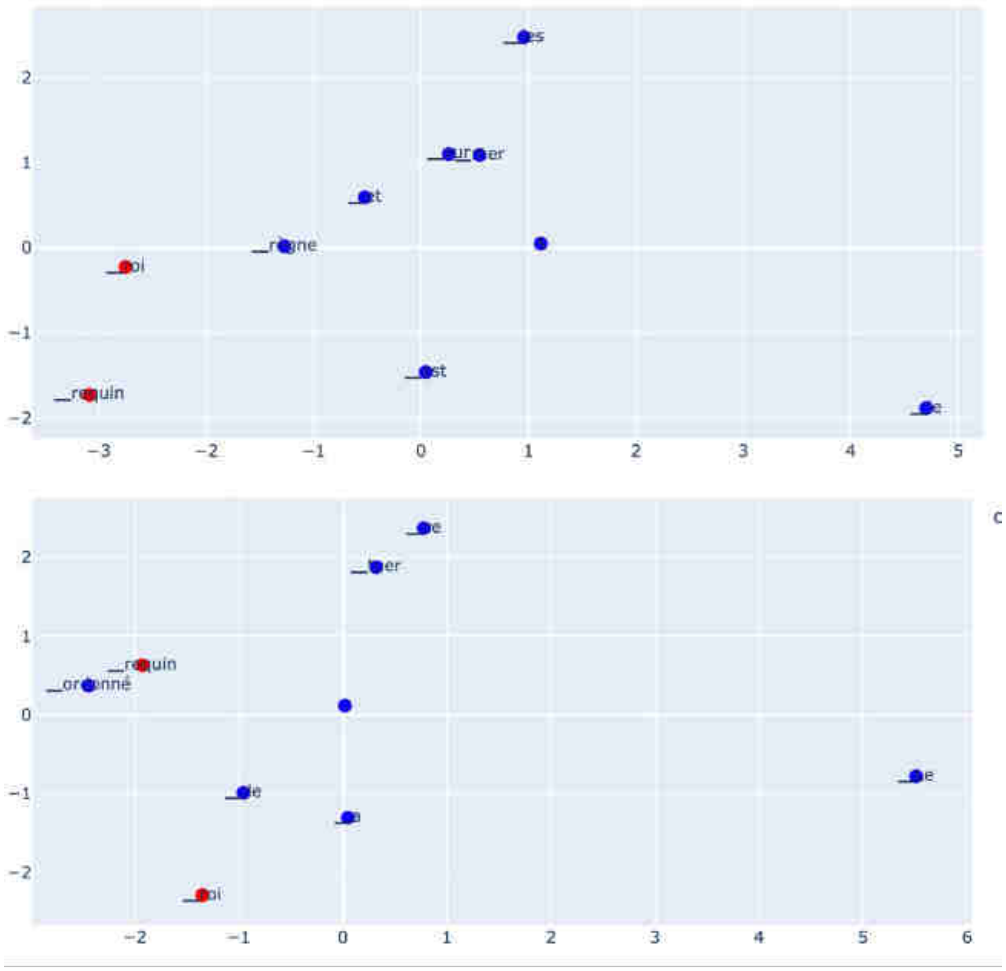

Figure 3 – *Word embeddings* créés par le modèle CamemBERT

Pour montrer la capacité des *Word embeddings* à « comprendre » le sens du mot, nous avons utilisé un modèle pré-entraîné basé sur le français : CamemBERT (Martin et al., 2020), pour créer des *embeddings* des deux phrases en français : « Le requin est roi et règne sur les mers. » et « Le roi a ordonné de tuer ce requin. ». Il est évident que dans ces deux phrases, la relation entre « requin » et « roi » est complètement différente. Dans la première phrase, il s'agit d'une relation de métaphore où les deux font référence à une même entité ; tandis que dans la deuxième phrase, il s'agit d'une relation antagoniste où les deux font référence à deux entités différentes. Après avoir généré les *word embeddings* avec CamemBERT, nous avons d'abord effectué une réduction de dimension à l'aide de la méthode PCA (sinon, ces données multidimensionnelles ne pourraient pas être traitées par les outils de visualisation). Ensuite, nous les avons visualisés avec Plotly [4]. Selon la figure ci-dessus, nous pouvons constater que dans la première phrase, la distance entre « requin » et « roi » est en effet plus proche, cela signifie que la technologie des *word embeddings* est capable de bien « comprendre » le sens du mot.

Grâce au développement des modèles pré-entraînés ces dernières années, des modèles

4. URL : https ://plotly.com/python/(accès le 1er août 2023).

pré-entraînés spécifiques au chinois classique ont été créés, ce qui signifie que nous disposons maintenant des *word embeddings* adaptées à l'ancien chinois. Dans notre projet, nous utiliserons les *word embeddings* du modèle SikuBERT (Wang et al., 2022). Le dernier est un modèle pré-entraîné en chinois classique, crée sur la base du *Siku Quanshu* (un énorme projet d'édition de la dynastie de Qing en Chine).

2.3.2.1 LSTM et ses variantes

Le modèle LSTM (Long-Short Term Memory) est une structure RNN (Recurrent Neural Network) spéciale (Hochreiter & Schmidhuber, 1997). Ce dernier est une structure de réseau neuronal conçue pour traiter des données séquentielles telles que des séquences temporelles (vidéos) et textuelles. Le RNN utilise une structure correspondant à la séquence pour traiter les données, c'est-à-dire des étapes temporelles pour chaque élément de la séquence. Pour chaque séquence, le RNN la décompose en plusieurs étapes temporelles, puis traite ces étapes dans l'ordre. Afin de capturer les dépendances entre chaque étape temporelle, le RNN utilise un état caché (*hidden state*) pour transmettre des informations entre les différentes étapes, c'est-à-dire que la sortie de chaque étape actuelle est utilisée comme entrée pour l'étape suivante. Grâce à cette structure, le RNN a une capacité de modélisation de séquence plus forte par rapport aux modèles traditionnels comme CRF.

Cependant, le modèle RNN a un inconvénient, à savoir qu'il est difficile de traiter de longs textes avec lui. Plus précisément, lorsque deux composants interdépendants dans une longue section de texte sont trop éloignés l'un de l'autre, il est probable que le RNN « oublie » leur dépendance. Supposons que nous ayons cette phrase : « J'aime voyager car cela me permet d'aller dans des endroits différents, de découvrir différentes cultures et de rencontrer des gens différents. Récemment, je suis allé en Égypte. » Lorsque le modèle RNN traite le mot « Égypte », il est possible qu'il ait déjà oublié le mot « voyage » car ils sont assez éloignés l'un de l'autre. Cela finira par empêcher le modèle de traiter correctement les relations de dépendance dans la phrase.

Le LSTM a été conçu pour répondre aux déficiences de la mémoire du RNN lors du traitement de longues séquences. Les LSTM ont introduit un état de transmission supplémentaire et un mécanisme de porte (*gate*) dans les réseaux RNN traditionnels. Dans le LSTM, deux états sont transmis entre chaque étape temporelle, correspondant à la mémoire à long terme et à la mémoire à court terme. Ensuite, le mécanisme de porte décide quelles informations doivent être oubliées et quelles informations doivent être transmises. Cette caractéristique améliore les performances du modèle LSTM lors du traitement de textes de grande longueur.

Le BiLSTM, pour sa part, se compose de deux réseaux LSTM qui traitent la même séquence dans des directions opposées. Cette configuration permet au modèle d'intégrer à la fois les informations précédant et suivant la séquence en cours. Ainsi, pour le traitement de textes, le modèle LSTM est en mesure de prendre en compte l'intégralité du contexte.

De plus, les chercheurs ont toujours essayé de combiner le modèle LSTM avec d'autres modèles pour maximiser ses performances. Ma et Hovy (2016) ont proposé une structure

de modèle qui combine BiLSTM, CRF et CNN. Dans la structure du modèle proposé, les auteurs utilisent d'abord un CNN (Convolutional Neural Network) comme extracteur de caractéristiques (*token embedding*), puis ajoute une couche CRF à l'intérieur du modèle BiLSTM pour optimiser les résultats de prédiction du BiLSTM.

La même année, d'autres chercheurs ont également proposé des modèles similaires pour la reconnaissance des entités nommées en utilisant un modèle combinant LSTM et CRF (Lample et al., 2016).

Lorsque nous commençons à envisager des méthodes d'apprentissage profond, le modèle LSTM est le premier à être pris en compte, car tout d'abord, il y a beaucoup moins de longs textes par rapport aux courts textes dans ce projet. Dans cette situation, le LSTM peut efficacement traiter les dépendances et deuxièmement, la mise en œuvre du modèle est relativement simple. En outre, Ezen-Can (2020) ont également montré que la performance du modèle LSTM est mieux que certains grands modèles dans des situations où il y a peu de données.

2.3.2.2 Transformer

Bien que le modèle LSTM ait amélioré dans une certaine mesure la capacité du modèle RNN traditionnel à traiter les dépendances à longue distance, il reste encore relativement faible lorsqu'il s'agit de traiter des séquences très longues.

Le Transformer est un nouveau modèle d'apprentissage profond proposé en 2017 qui a déjà battu les modèles RNN dans certains tests de performance dès sa proposition (Vaswani et al., 2017). Ce type de modèle utilise un mécanisme appelé *self-attention* : pour chaque mot dans une séquence de texte, le modèle calcule la pertinence de ce mot par rapport à chaque mot dans la séquence ; le calcul est basé sur les *word embeddings* construits. Cette pertinence signifie le niveau d'attention que le modèle doit accorder à ce mot dans une certaine tâche.

Nous allons expliquer brièvement le fonctionnement du modèle Transformer avec un exemple concret. Supposons que nous devions utiliser le Transformer pour construire un modèle de traduction du français vers l'anglais.

Tout d'abord, lors de la phase de vectorisation des mots, contrairement à d'autres réseaux neuronaux tels que les RNN, où nous pouvons simplement effectuer les *word embeddings*, dans le cas du Transformer, nous devons non seulement effectuer les *word embeddings*, mais aussi encoder la position de chaque mot dans une séquence. La raison en est que le mécanisme de calcul parallèle utilisé par le Transformer ne lui permet pas d'obtenir les informations sur la position des mots dans un texte. Par exemple, pour les séquences « un, deux, trois » et « trois, deux, un », le Transformer ne sera pas capable de distinguer les deux sans l'encodage de position.

Supposons que le premier texte de la phase d'entraînement soit : « Je suis heureux » et « I am happy », Après avoir terminé l'étape précédente, le texte en français sera entré dans l'encodeur. L'encodeur utilisera un mécanisme de *self-attention* pour calculer la corrélation entre chaque mot de la séquence, le but est de générer une représentation

vectorielle plus précise qui comporte les informations contextuelles (supposons que cela soit *v*) que les *word embeddings* initiaux. Ensuite, *v* sera entré dans le décodeur. Le décodeur combinera alors *v* avec le texte en anglais pour apprendre de la manière suivante : tout d'abord, un marqueur spécial comme <START> sera ajouté au début du texte en anglais : « <START> I am happy ». Le décodeur tentera d'abord de générer le premier mot de la langue cible en combinant <START> avec *v*, puis il combinera « <START> I » avec v pour générer le deuxième mot de la langue cible, et ainsi de suite. Le décodeur prend toujours le texte brut (plutôt que du texte généré par lui-même) pour essayer de deviner le mot suivant. Dans chaque processus de génération de nouveaux mots, le décodeur calcule d'abord le *self-attention* du texte en anglais ainsi que l'attention croisée (*cross attention*) entre ce texte et le texte en français (*v*). De cette manière, il apprendra progressivement comment générer le mot suivant en fonction du self-attention de tout le texte déjà généré et l'attention croisée entre le texte généré et le texte en langue source.

Le modèle Transformer est notre deuxième tentative de réseau neuronal. En raison de sa puissante capacité de modélisation, nous avons décidé d'utiliser ce modèle.

2.3.3 Apprentissage par transfert

L'émergence du Transformer a considérablement stimulé le développement du domaine du traitement automatique des langues (TAL). Sa structure de calcul parallèle a grandement amélioré l'efficacité de l'entraînement des modèles, tandis que son mécanisme de *self-attention* a renforcé leur capacité à traiter des textes longs. Sous l'effet de ces facteurs, les modèles pré-entraînés ont connu une rapide progression.

Depuis la naissance du Transformer, les modèles pré-entraînés basés sur celui-ci sont devenus très populaires. Selon leur lignée technologique respective, ces modèles peuvent être divisés en trois catégories :

1. Les modèles de type BERT (Devlin et al., 2019) dont l'objectif principal est l'extraction de caractéristiques, le mécanisme bidirectionnel utilisé par BERT lui permet de générer des représentations vectorielles précises pour chaque mot en combinant le contexte à gauche et à droit. Ces représentations contiennent des informations sémantiques, grammaticales et contextuelles pour chaque mot dans le texte. Pour le modèle BERT-base, chaque mot correspond à un vecteur de 768 dimensions. Comme ce type de modèle ne concerne pas la génération de texte, ils utilisent tous uniquement l'encodeur du Transformer.
2. Les modèles de type GPT (Radford et al., 2018). Ce type de modèle est conçu dans le but de générer du texte, sa caractéristique principale étant son aspect auto-régressif. Cela signifie que le modèle prend en compte les textes précédemment générés lorsqu'il génère le texte actuel. Cette caractéristique permet au modèle de produire un texte très fluide. Ce type de modèle utilise uniquement la partie décodeur du Transformer.
3. Les modèles T5 (Raffel et al., 2020), qui utilisent une structure complète encodeur-décodeur similaire au Transformer original.

Ces modèles reposent entièrement sur l'apprentissage non supervisé et l'apprentissage par transfert. Leur approche fondamentale, pour la partie non-supervisée, consiste à collecter d'abord de grandes quantités de textes non étiquetés, puis à utiliser des tâches

spécifiques (par exemple, pour le célèbre modèle BERT, les chercheurs ont adapté une méthode qui s'appelle « Masked Language Modeling », où certains mots dans les données d'entraînement sont masqués et le modèle tente de deviner ces mots) pour inciter le modèle à apprendre les propriétés d'une langue, telles que la structure grammaticale, les collocations de mots et de phrases, etc.

Ainsi, nous obtenons un modèle pré-entraîné, que nous pouvons ensuite affiner (*fine-tune*) spécifiquement pour une tâche donnée, en utilisant de l'apprentissage supervisé sur un corpus annoté, afin d'obtenir un modèle performant. En raison de la taille massive des données d'entraînement utilisées pour créer le modèle pré-entraîné, les modèles finaux obtenus par affinage des modèles pré-entraînés sont généralement supérieurs aux modèles d'apprentissage supervisé traditionnels.

Ces dernières années, un certain nombre de modèles pré-entraînés basés sur le chinois classique ont vu le jour, ces modèles utilisent généralement l'architecture BERT ou ses dérivés. Par exemple, le projet SikuBERT est un modèle pré-entraîné basé sur RoBERTa (une dérivée de l'architecture BERT), pour lequel les auteurs ont utilisé presque 1 milliard de textes pour son pré-entraînement.

Grâce à la communauté ouverte de Hugging Face [5] , nous pouvons facilement appeler ces modèles via une API et les affiner pour les appliquer directement aux tâches en aval.

L'apprentissage par transfert a déjà été utilisé pour l'encodage automatique de corpus en TEI. Pagel et al. (2022) ont utilisé BERT pour encoder des pièces de théâtre en allemand à partir de versions de ces pièces en texte brute, avec des bons résultats. Dans notre travail, nous utiliserons également BERT, seul ou combiné avec d'autres modèles.

2.4 Conception du corpus

Dans la chaîne de traitement que je projette de mettre en place, le balisage automatique est suivi par la définition du conteneur pour contenir les balises. Il existe aujourd'hui plusieurs projets de dictionnaire qui sont basés sur TEI pouvant nous servir de référence. C'est le cas de e-SND.

Le projet e-SND vise à numériser le dictionnaire national écossais (*Scottish National Dictionary*), qui est un projet lancé en 1929 dans le but de construire un dictionnaire de la langue écossaise depuis le XVIIIe siècle. Le processus de travail du projet e-SND consiste tout d'abord à transcrire les dictionnaires existants grâce à la technologie OCR, le texte transcrit sera ensuite disponible sous forme de document Word et sera corrigé manuellement ; ensuite, ces documents seront convertis en format XML par des programmes informatiques, nécessitant toujours un contrôle éditorial pendant ce processus (Rennie, 2000).

e-SND a analysé la compatibilité entre les Guidelines de TEI et son propre projet, dont trois points sont inspirants.

Tout d'abord, au niveau de la prononciation, l'auteur mentionne une présomption dans

5. URL : https ://huggingface.co/ (accès le 1er août 2023).

les Guidelines TEI concernant les normes d'encodage des dictionnaires : la partie prononciation doit toujours suivre immédiatement la partie orthographique. Cela explique pourquoi dans les Guidelines TEI, <orth> et <pron> sont tous deux des sous-éléments de <form>. Cependant, cette présomption n'existe pas réellement dans le projet e-SND. Par conséquent, l'auteur a modifié cette restriction pour rendre la TEI plus adaptée au projet. Notre projet est également confronté à ce problème. Dans le dictionnaire que nous encodons, la partie de prononciation est séparée de celle de l'orthographe. La méthode adoptée par l'auteur nous offre donc une piste pour résoudre ce problème.

En ce qui concerne les citations, étant donné que ce dictionnaire contient des références à différents dialectes écossais de différentes régions, l'auteur a ajouté un nouvel élément <geo> pour indiquer la provenance d'une citation spécifique. Cela nous inspire car, pour la partie des citations, étant donné que l'élément <cit> ne peut par défaut contenir que des informations liées au livre lui-même, si la partie de citation contient du contenu similaire à des commentaires ou à des métadonnées, nous pouvons le résoudre en utilisant un nouvel élément. Pour le contenu cité, nous avons également rencontré un problème similaire : c'est-à-dire que le contenu cité, en plus du nom de l'auteur, du titre de l'œuvre et de la citation elle-même, contient souvent des informations redondantes telles que :《易》曰：“禔既平。” Dans cet exemple, il y a beaucoup de signes de ponctuation inclus, alors qu'en chinois classique on n'utilise pas de signes de ponctuation.

Dans la section étymologie, l'auteur estime que seul l'élément <mentioned> pourrait ne pas suffire à répondre aux besoins, c'est pourquoi elle introduit deux autres éléments : <etymon> et <cognate>, qui correspondent respectivement à l'étymologie et aux mots apparentés.

Ce projet, en tant que projet de dictionnaire, présente de nombreuses similitudes avec certaines œuvres de notre propre projet. Par exemple, les deux projets sont confrontés à des problèmes concernant la distance entre la partie prononciation et la partie orthographe, ainsi que des problèmes liés aux informations supplémentaires dans les citations. e-SND nous offre une référence très utile pour la conception de balisage.

3 Méthodologie : Sources choisies et chaîne de traitement

Nous avons construit une chaîne de traitement complète dans ce projet, comprenant l'acquisition, le traitement, la modélisation et la relecture du texte, ainsi que la constitution de corpus. Nous décrirons chaque étape en détail.

Dans cette section, nous commençons par présenter notre sélection de source de texte (3.1), suivi de la proposition de modélisation en TEI pour le contenu (3.2). Par la suite, nous détaillons la chaîne de traitement intégral pour l'étiquetage TEI du corpus, impliquant l'utilisation de règles, d'apprentissage automatique et d'apprentissage profond pour la prédiction automatique des balises TEI (3.3). Nous expliquons également la procédure permettant de formaliser le résultat de la prédiction en XML-TEI (3.4). Pour conclure, nous décrivons le processus de relecture nécessaire pour la validation manuelle du résultat (3.5).

3.1 Acquisition de texte

Lorsque l'on compare les deux sources de corpus, à savoir Wikisource et Chinese Text Project, il est évident qu'elles partagent de nombreux points communs. Ce sont toutes deux des initiatives communautaires qui mettent gratuitement à disposition leurs ressources pour les utilisateurs. Cependant, la différence majeure réside dans leur approche : Wikisource adopte une vision d'ouverture totale qui permet la modification et la republication des textes sans demander d'autorisation écrite ou verbale, ce qui n'est pas le cas pour le CTP. Si nous devions attribuer un adjectif à chacun de ces projets, "ouvert" et "gratuit" serait probablement le plus approprié pour Wikisource, tandis que "gratuit" pourrait mieux décrire CTP.

Il est indéniable que Wikisource est la source de texte la plus adaptée à ce projet. Par conséquent, notre projet utilise ce site web comme source de données principale. Cependant, cela ne signifie pas que CTP soit dénué de pertinence. Par exemple, CTP a construit une base de données pour un grand nombre d'entités nommées. L'avantage est que lorsque nous voyons une référence à l'œuvre b dans l'œuvre a, les utilisateurs peuvent être redirigés vers toutes les instances qui font référence à l'œuvre b en cliquant directement sur celle-ci, ainsi que la présentation soignée des pages de lecture du texte, constituent également des références importantes pour ce projet. Dans la partie suivante, nous utiliserons principalement le *Shuowen Jiezi* comme exemple pour présenter en détail notre chaîne de traitement.

Pour obtenir le texte, nous avons choisi Wikisource comme source car ses droits d'auteur sont entièrement ouverts. Wikisource nous permet de sauvegarder son contenu dans plusieurs formats et nous avons choisi le format txt car il ne contient que du texte qui représente la meilleure édibilité. Nous avons directement téléchargé le texte principal de l'œuvre, ce qui nous a donné un seul fichier txt contenant toutes les sections. La division était très simple car chaque section commençait explicitement par une indication, par exemple "卷一 (*juan* 1)" pour la première ligne du premier volume. Nous avons donc pu effectuer cette division facilement à l'aide d'expressions régulières. De

plus, nous avons remarqué que Wikisource semble fournir directement aux utilisateurs du texte en capturant des pages web similaires à celles présentes sur leur site, parce que notre texte obtenu contenait des traces de pages web telles que des boutons de navigation présents sur les pages Wikisource. Lors du nettoyage du texte, nous utilisons aussi des expressions régulières pour supprimer tous ces éléments.

3.2 Structure de la source et conception du schéma TEI

Dans cette section, nous allons présenter en détail les deux ouvrages que nous traitons, y compris leur analyse de structure et le schéma TEI que nous avons conçu en fonction de la structure du texte original.

3.2.1 Shuowen Jiezi

Le *Shuowen Jiezi* est considéré comme le premier dictionnaire de caractères chinois, il propose une analyse complète de plus de 9000 caractères au point de vue lexicologique. Xu Shen (58-147), son auteur, a inventé un système de classement des caractères basé sur les clés (Bottéro, 1996), c'est-à-dire qu'il a identifié dans chaque caractère un élément graphique qui sert à son classement. Le *Shuowen Jiezi* compte 540 clefs. Cette méthode de classement a exercé une grande influence et a été reprises maintes fois par des dictionnaires ultérieurs. C'est une méthode de classement dérivée de celle-ci qui est la plus répandue dans les dictionnaires de la langue chinoise contemporains en version papier.

3.2.1.1 Analyse de la structure

Notre étude de la structure de l'œuvre se déroule en deux étapes, en examinant d'abord la macrostructure, puis la microstructure. La macrostructure, unique et invariable, détermine l'organisation générale de l'œuvre, englobant à la fois la méthode de classification et la structure hiérarchique.

Cependant, la microstructure, qui diffère de la macrostructure, n'est pas unique. Bien qu'une œuvre ne puisse posséder qu'une seule macrostructure, il n'y a aucune contrainte sur le nombre de microstructures qu'elle peut contenir. Ce caractère illimité et variable des microstructures présente un défi majeur lorsqu'il s'agit de leur traitement automatisé dans les corpus de textes.

Le nombre fluctuant de microstructures, variant d'une œuvre à l'autre, complique souvent l'application des méthodes basées sur des règles, rendant leurs résultats incertains. En outre, la difficulté est accrue lorsqu'il s'agit de résumer toutes les microstructures d'une œuvre riche en mots avec un ensemble fixe de règles.

Cette œuvre présente une macrostructure relativement épurée, articulée autour de trois niveaux distincts. Elle se compose de quinze volumes (*juan* 卷), chacun d'eux subdivisé en plusieurs sections (*bu* 部), qui elles-mêmes abritent une ou plusieurs entrées. Cette structure standardisée, dénuée de complexité superflue, est en grande partie due au fait que l'œuvre est un dictionnaire. De fait, les différentes parties ne sont pas interconnectées, ce qui simplifie grandement la structure globale.

Ceci illustre également pourquoi la macrostructure du *Shuowen Jiezi* est presque entièrement fonctionnelle. En effet, elle ne véhicule pas d'information directe. Au contraire, les informations sont contenues au sein des entrées individuelles qui constituent l'œuvre.

La microstructure fait référence à la composition spécifique de l'entrée, qui peut inclure les éléments suivants : définition, étymologie, prononciation, titre et citation, chaque entrée peut contenir un ou plusieurs de ces éléments. Étant donné que le texte original contient de nombreuses entrées complexes, l'analyse manuelle de la structure est très chronophage. Pour cela, nous avons introduit le concept de « mot-outil » pour aider à analyser la microstructure des différentes entrées en utilisant un script Python. Les mots-outils n'ont pas eux-mêmes une signification lexicale spécifique mais sont utilisés pour aider à former une certaine structure. Dans le *Shuowen Jiezi*, presque chaque partie différente contient un ou plusieurs mots-outils spécifiques ; par exemple, la partie définition d'une entrée se termine souvent par le caractère chinois *ye* 也 . Nous utilisons donc ces mots-outils pour déterminer la composition des entrées.

Nous devons d'abord trouver les mots-outils les plus utiles, c'est-à-dire ceux qui apparaissent le plus souvent. Pour ce faire, nous avons calculé à l'aide d'un script l'occurrence de chaque caractère. Avant cela, nous avons créé une liste de mots vides (*stoplist*) contenant certains signes de ponctuation pour réduire les résultats inutiles.

```
from collections import Counter
import re
#ignore punctuation
list_stopword = ['。','：',', ',',"',' “',' 《',' 》']
list_entries = []
liste_token = []
liste_occurence = []
count = 0
with open('shuowen.txt','r',encoding='utf-8') as f:
    for i in f.readlines():
        if i != '\n'  and not re.match( r'. 部', i) :
            count += 1
            list_entries.append(i)
        for i2 in i :
            if i2 != '\n' and i2 != '' and i2 not in list_stopword :
                liste_token.append(i2)
for i in Counter(liste_token).most_common(10) :
    liste_occurence.append(i)
for i in liste_occurence :
    print(i)
```

Le code nous donnera le résultat suivant :

1. (从 , 11046)
2. (也 , 9075)
3. (聲 , 8555)
4. (曰 , 2159)
5. (之 , 1750)
6. (一 , 994)
7. (水 , 931)
8. (若 , 803)
9. (讀 , 790)
10. (木 , 747)

Les résultats ci-dessus nous ont permis d'identifier les 5 mots-outils les plus fréquemment utilisés dans le texte original : *cong* 从 , *ye* 也 , *sheng* 聲 , *yue* 曰 et *zhi* 之 .

Le mot *cong* 从 , qui est souvent utilisé pour indiquer l'étymologie, est le plus fréquent parmi ces mots, et nous pouvons constater que ses occurrences dépassent le nombre total d'entrées du texte original. Ceci est principalement dû au fait que certaines entrées contiennent plusieurs sources étymologiques. Le mot se trouve souvent au début de la partie étymologique et est souvent combiné avec la partie sur la prononciation. Cette combinaison est parfois directe, c'est-à-dire qu'il n'y a pas d'autre élément entre les deux, et parfois elle est indirecte : il existe une virgule ou un point-virgule qui sépare les deux (Il convient de noter que cela ne provient pas de l'auteur lui-même, mais a été ajouté récemment, car dans l'ancien chinois, nous n'utilisons aucun signe de ponctuation) et cette dernière situation se produit souvent lorsqu'il y a plus d'un mot-outil *cong* 从 .

Le mot *ye* 也 , quant à lui, indique la fin d'une définition. Il existe toutefois des définitions qui n'utilisent aucun mot-outil. Et ce phénomène est presque inexistant dans d'autres composantes, par exemple on ne peut imaginer une partie étymologique sans le mot *cong* 从 (qui signifie suivre, venir de). C'est également en raison de ce phénomène que notre analyse de la microstructure est compliquée.

Le troisième mot, *sheng* 聲 , existe pour donner la prononciation d'une entrée. Il est placé après le caractère homophone de celui présenté en entrée. Outre ce mot, il existe un autre mot, *du ruo* 讀若 , qui peut également être utilisé à cette fin, mais qui dans ce cas est placé avant le caractère homophone..

Le dernier mot, *yue* 曰 , est un cas particulier : il apparaît dans diverses parties. Il sert généralement à introduire une citation en lien avec la référence bibliographique qui précède le mot. Mais il est aussi utilisé dans l'expression fixe *yi yue* 一曰 , qui sert à introduire une autre définition (polysémie). Il est important de noter que tant ce mot que le mot *zhi* 之 apparaissent beaucoup moins fréquemment que les trois autres mots (从 , 也 et 聲), et que la partie qu'ils constituent au sein d'une entrée se présente comme une structure accessoire plutôt qu'une structure primaire.

Figure 4 – Composition d'une entrée typique dans le *Shuowen Jiezi*

En résumé, nous avons examiné ces mots-outils pour déterminer la composition de chaque entrée, puis nous avons calculé la proportion des entrées de chaque type. Ces résultats ont guidé notre modélisation du corpus.

D'après nos observations, nous pouvons diviser ces cinq mots-outils ou les composants qu'ils représentent en deux catégories : les parties fondamentales et les parties supplémentaires. Les premières sont des éléments que la plupart des entrées possèdent, tandis que la fréquence d'apparition des secondes est inférieure à celle des premières. Les mots-outils correspondant aux parties fondamentales sont *cong* 从 , *ye* 也 et *sheng* 聲 , tandis que ceux correspondant aux parties supplémentaires sont *yue* 曰 et *zhi* 之 . Notre analyse structurelle sera basée sur cette classification.

Nous mesurons d'abord le nombre d'entrées qui contiennent simultanément les trois mots-outils fondamentaux :

```
liste_ 也从聲 = []
compteur = 0
for i in liste_entree:
     if '也' in i and '从' in i and '聲' in i :
          compteur += 1
          liste_ 也从聲.append(i)
print(compteur2)
```

Cela nous donne le nombre d'entrées qui contiennent simultanément les trois principaux mots-outils : 6887.

Pour faciliter le traitement ultérieur, il est important de subdiviser les différents types dès que possible. En lien avec l'analyse microstructurelle précédemment effectuée, chaque type peut contenir des références à d'autres travaux ainsi que d'autres définitions, c'est pourquoi nous continuons la subdivision sur la base de la classification précédente :

```
compteur3 = 0
liste_avec_reference = []
for i in liste_ 也从聲 :
    if "《" in i and "》" in i and " 一曰 " not in i :
         compteur3 += 1
         liste_avec_reference.append(i)
print(f"Nombre de ce type : {compteur3}")
```

```
print(f"Pourcentage de ce type : {round(compteur3/compteur*100,2)} %")
```

Ce code permet d'obtenir le nombre d'entrées du type 也从聲 qui contiennent uniquement des références à d'autres ouvrages. Nous poursuivons ensuite cette opération en comptant le nombre des trois autres sous-classes du type, à savoir celles qui ne contiennent aucun élément supplémentaire, celles qui contiennent à la fois des références et des autres définitions, ainsi que celles qui contiennent uniquement des autres définitions. Nous obtenons finalement des informations suivantes (Tous les calculs de pourcentage sont basés sur la proportion de cet élément par rapport à toutes les 9831 entrées de l'œuvre) :

```
Nombre des entrées qui comportent 也从聲 : 6887 / 69.88%, dont :
687 / soit 6.97% du total avec référence
518 avec plusieurs définitions / 5.26%
64 avec les deux en même temps / 0.65%
5618 sans aucun élément supplémentaire / 57.0%.
```

Ensuite, nous répétons ce processus et obtenons le nombre d'entrées pour chaque type comme indiqué dans la figure 5. Dans cette figure, l'étiquette : « autre » fait référence aux entrées qui n'utilisent pas de mots-outils.

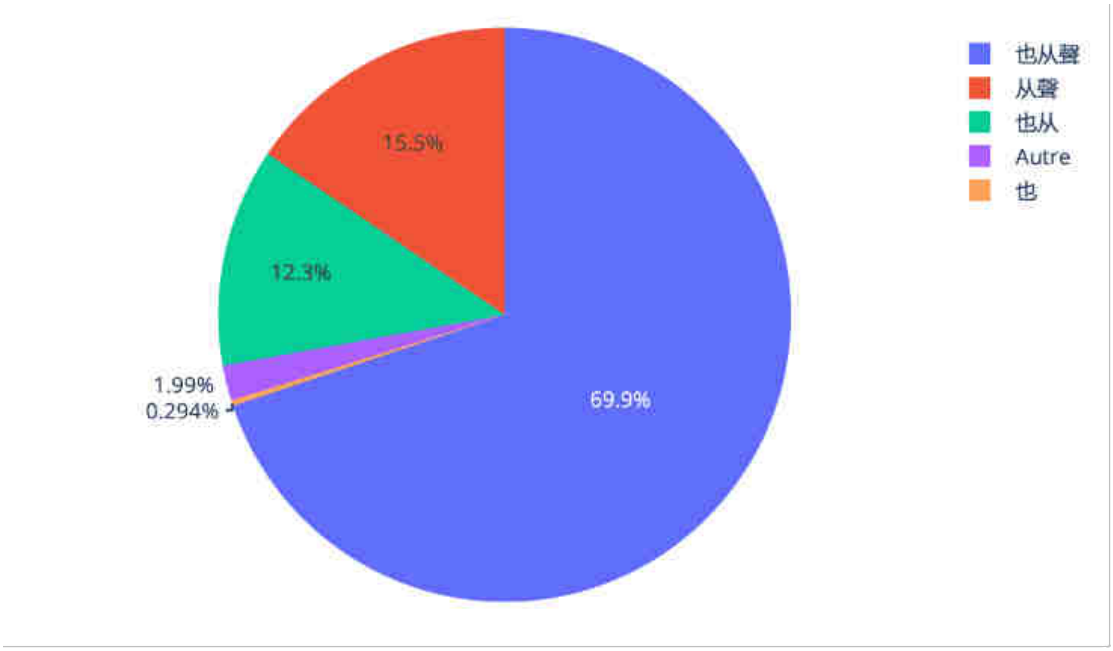

Figure 5 – Répartition des différents types d'entrée dans le *Shuowen Jiezi*

Grâce à l'analyse des mots-outils, nous pouvons conclure que près de 70% des entrées dans le *Shuowen Jiezi* contiennent au moins une définition, une prononciation et une étymologie. Environ 57% des entrées ne contiennent que ces trois éléments, tandis que près de 13% incluent également d'autres définitions ou références à d'autres œuvres. En outre, 12,3% des entrées ne contiennent pas de partie prononciation, tandis que 15,5% des entrées n'incluent pas de définition (comme chaque entrée a obligatoirement une

définition, cela signifie en réalité qu'il y a 15,5% des entrées qui n'utilisent pas de mot-outils pour marquer la définition). Sur la base des analyses décrites dans cette section, nous avons conçu la modélisation TEI décrite dans la section suivante.

3.2.1.2 Conception du schéma TEI

Nous avons d'abord conçu la macrostructure du corpus TEI. Selon notre analyse précédente, étant donné que le *Shuowen Jiezi* utilise une structure à trois niveaux, nous avons donc conçu une structure TEI correspondante avec des niveaux hiérarchiques complets.

En raison du fait que la structure de deuxième niveau « ouvrage » et « volume » ne contient pas directement d'entrées, nous les avons conçus comme des conteneurs de métadonnées. Autrement dit, nous avons créé un fichier TEI pour l'ouvrage afin de stocker les métadonnées relatives aux volumes (y compris les chemins d'accès et les identifiants pour chaque volume). De même, pour chaque volume, nous avons créé un fichier qui contient les métadonnées des sections qu'il contient (y compris le chemin d'accès et l'identifiant pour chaque section). En ce qui concerne la structure des sections, étant donné qu'elle inclut directement des entrées, nous l'avons considérée comme l'unité de base du corpus.

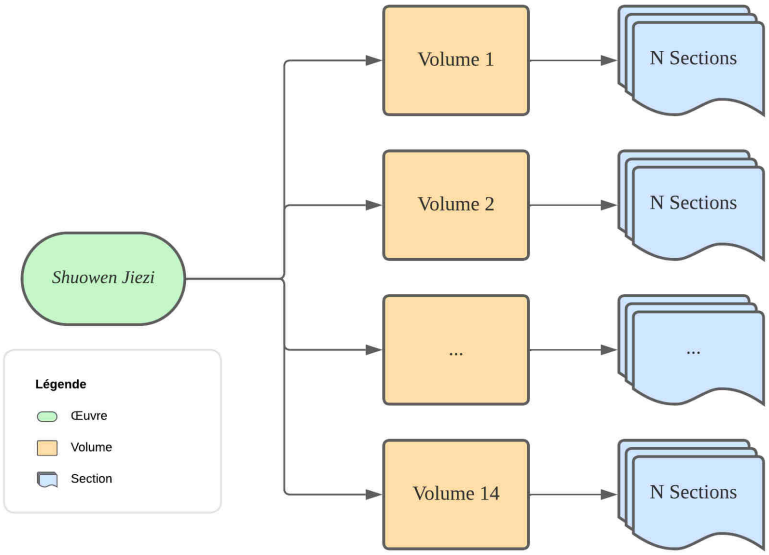

Figure 6 – Macrostructure du *Shuowen Jiezi*

Pour tous les fichiers de corpus, nous utilisons l'éléments `<TEI>` comme élément racine, chaque élément racine contenant deux sous-éléments : `<teiHeader>` et `<text>`. Le premier stocke les métadonnées de l'œuvre, cette partie étant identique pour tous les fichiers. Quant à l'élément `<text>`, il contient le corpus lui-même qui est stocké dans le sous-élément `<body>`.

Dans l'élément `<text>`, il n'y a qu'un seul sous-élément `<body>` qui contient toutes les entrées d'une section. Pour chaque entrée, nous utilisons un élément `<entry>`. Selon notre analyse du texte, les parties possibles d'une entrée sont : définition, étymologie, prononciation, titre et citation. Nous allons à présenter discuter de l'utilisation des

balises correspondant à chacune de ces parties.

<entry>

Tout d'abord, notre premier élément, sera l'élément <entry>, qui est l'unité constitutive de base du corpus d'un point de vue de macrostructure, et la racine de tous les composants à l'intérieur d'une entrée d'un point de vue microstructurel. C'est l'élément le plus important, car il est la clé qui relie la macrostructure à la microstructure.

Du fait que l'entrée de dictionnaire est l'élément le plus important, presque tout le contenu du dictionnaire est directement stocké dans l'entrée. Par conséquent, nous devons attribuer un identifiant unique à chaque entrée pour une meilleure utilisation par les systèmes informatiques futurs. Les avantages sont les suivants :

1. Éviter les erreurs d'indexation : bien que chaque titre d'entrée soit un caractère chinois unique, en raison du support Unicode incomplet des caractères CJK, il n'est pas possible de garantir la perfection si on utilise directement ces titres comme index.
2. L'utilisation de l'identifiant permet une meilleure indexation du dictionnaire et nous pouvons inclure plus d'informations telles que la section et le thème auxquels appartient cette entrée dans l'identifiant afin d'éviter une indexation basée sur les caractères chinois et fournir ainsi une indexation parfaite.

Une balise d'entrée pourrait ressembler à celle de l'exemple ci-dessus :

```
1 <entry xml:id="">
```

Pour le format de l'id, il existe deux types : la position absolue et la position relative. Ces deux concepts sont empruntés à l'informatique pour les chemins absolus comme pour les chemins relatifs. La position absolue est la position de l'entrée dans l'ensemble de l'ouvrage, par exemple, `xml :id = '0001'` indique la première entrée dans l'ensemble du dictionnaire. La position relative, en revanche, tient compte de la position d'une structure à trois niveaux, par exemple, `xml :id = 'v1_s1_001'` indique également la position dans l'ensemble du dictionnaire, mais elle tient compte de la structure à trois niveaux mentionnée précédemment : la première entrée dans le première section du premier volume. En raison de la quantité d'informations supplémentaires et de la lisibilité qu'elle offre, nous avons finalement choisi d'utiliser la position relative comme méthode de codage pour les `xml :id`.

<form>

Il s'agit du premier sous-élément dont le rôle est de stocker le mot vedette d'une entrée. Selon la norme TEI, cet élément peut contenir plusieurs sous-éléments, par exemple : l'élément <orth>, qui peut être utilisé pour stocker le mot lui-même, <abbr> pour stocker les abréviations, <pron> pour la prononciation, etc. Pour ce projet, nous ne disposons que d'un seul type d'informations sur la tête de l'entrée : le mot lui-même, donc seulement un sous-élément est requis : <orth>.

Il convient de noter que dans ce projet, nous avons décidé de ne pas utiliser l'élément

<pron>, sous-élément de <form> pour stocker la partie prononciation. La raison en est que dans cette œuvre, la prononciation des mots est souvent étroitement liée à la partie étymologique, qui se trouve généralement dans l'élément <etym>, qui peut à son tour être directement un sous-élément de <entry> ou un sous-élément de <sense>. Or, <pron> ne peut pas être contenu dans <etym>. En outre, cette partie prononciation fournit une référence et n'est pas nécessairement la véritable prononciation du mot. Par conséquent, nous avons décidé d'utiliser l'élément <pRef> pour stocker la partie prononciation et d'utiliser l'élément <oRef>, parallèle à celui-ci, pour la partie étymologique, comme indiqué ci-dessous :

```
<entry xml:id="swjz_j02_b05_e07">
<form><orth type="standard"></orth>: </form>
<sense>
  <def>二歲牛</def>。
</sense>
<etym>
  <oRef>从牛</oRef>
  <pRef>聲</pRef>。
</etym>
</entry>
```

<sense>

Cet élément sert principalement à contenir des définitions et des références à d'autres œuvres. Cet élément contient deux sous-éléments : <def> et <cit>. En raison de la complexité de la structure des entrées, cet élément peut apparaître plusieurs fois dans une seule entrée.

Parmi les sous-éléments de <sense>, <def> apparaît le plus fréquemment comme balise de stockage des définitions. Et à cause de la prévalence des polysémies, <def> a tendance à apparaître plusieurs fois.

<cit>

Nous utilisons l'élément <cit> pour stocker des informations sur la citation, qui est souvent la partie la plus complexe d'une entrée de dictionnaire car elle comprend trois sous-parties : le titre, le texte cité et autres. Pour cela, nous avons conçu une structure composite pour stocker les différentes sous-parties. Comme indiqué ci-dessous :

```
<cit type="classic" corresp="#ckpw-TBD">
<bibl>
<title>《周禮》</title>
</bibl>有“
<quote>莶菹</quote>
”。
</cit>
```

Dans l'exemples ci-dessus, nous avons donné deux attributs à l'élément <cit> : type et

corresp. En fait, il s'agit de la connexion entre ce corpus et le projet CHI-KNOW-PO. Les valeurs de ces deux attributs sont entièrement fournies par la base de données du projet CHI-KNOW-PO, qui définit un type et un identifiant pour chaque citation dans la base de données.

<etym>

Cet élément est utilisé non seulement pour stocker des informations étymologiques, mais aussi pour stocker des informations de prononciation. Les deux parties sont ensemble sous-éléments de l'élément <etym>. Par exemple :

```
<etym>
  <oRef>从牛</oRef>
  <pRef>參聲</pRef>。
</etym>
```

En résumé, pour la structure des entrées, nous avons adopté une conception en 3 couches. La première couche est l'élément <entry>, la deuxième couche est composée des sous-éléments <form>, <sense>, <cit> et <etym>, et la troisième couche est constituée des sous-éléments de la deuxième couche : <orth>, <def>, <cit>, <oRef> et <pRef>. Voici un exemple :

```
<entry xml:id="swjz_j02_b05_e19">
<form><orth type="standard">犉</orth>：</form>
<sense>
  <def>黃牛黑脣也</def>。
</sense>
<etym>
  <oRef>从牛</oRef>
  <pRef>□□ 聲/pRef>。
</etym>
<sense>
  <cit type="classic" corresp="#ckpw-TBD">
    <bibl>
      <title>詩</title>
    </bibl>曰：“
    <quote>九十其犉</quote>。”
  </cit>
</sense>
</entry>
```

Le figure ci-dessous montre de manière plus intuitive la relation hiérarchique entre toutes ces balises.

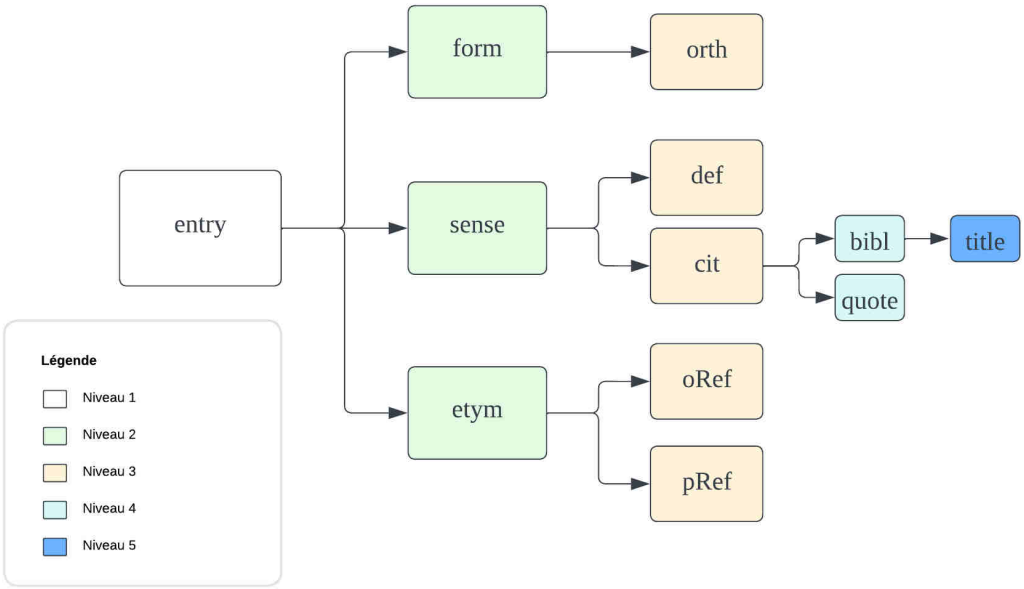

Figure 7 – Schéma TEI pour le *Shuowen Jiezi*

3.2.2 Yiwen Leiju

Le *Yiwen Leiju* est une encyclopédie compilée pendant la dynastie Tang au VIIe siècle. Le projet a commencé en l'an 588 de notre ère, dirigé par le célèbre érudit Ouyang Xun 歐陽詢 (557-641), avec la participation de savants à travers le pays. Achevé en 624, ce livre définissait 46 domaines tels que la nature, la politique et la religion, etc. L'objectif d'une encyclopédie à l'époque est de référencer des passages célèbres dans lesquels le mot titre de l'entrée figure. Aussi, l'une des grandes richesses de cet ouvrage réside-t-elle dans son inclusion d'un grand nombre de textes antérieurs au VIIe siècle. Bien qu'il s'agisse seulement d'extraits, ils ont fourni aux chercheurs ultérieurs des références sur les variantes textuelles (Denecke et al., 2020). Par ailleurs, l'encyclopédie cite les extraits en les classant systématiquement par genre littéraire.

3.2.2.1 Analyse de la structure

La première chose à noter est que les auteurs ont utilisé une méthode spéciale pour diviser les différentes parties dans cet ouvrage : bien que l'œuvre comporte 46 domaines, elle est divisée en 100 volumes (*juan* 卷). Parfois, un domaine est divisé en plusieurs volumes, tandis que plusieurs domaines peuvent être regroupés dans un même volume. Nous avons ensuite comparé les deux méthodes de partitionnement par domaine d'une part et par volume d'autre part. Nous avons calculé le nombre d'entrées pour chaque partie (domaine ou volume). Comme l'illustre la figure ci-dessous, si nous divisons directement en fonction du domaine, il y a une grande fluctuation du nombre d'entrées pour chaque partie. En revanche, par volume, la fluctuation est très faible. Par conséquent, nous pensons que les auteurs ont divisé le texte original en 100 parties afin d'équilibrer le texte à travers les parties. Cependant, dans un corpus électronique, nous n'avons pas besoin de prendre cela en compte, donc nous divisons directement selon le domaine

dans notre projet, tout en conservant la marque des subdivisions par volume.

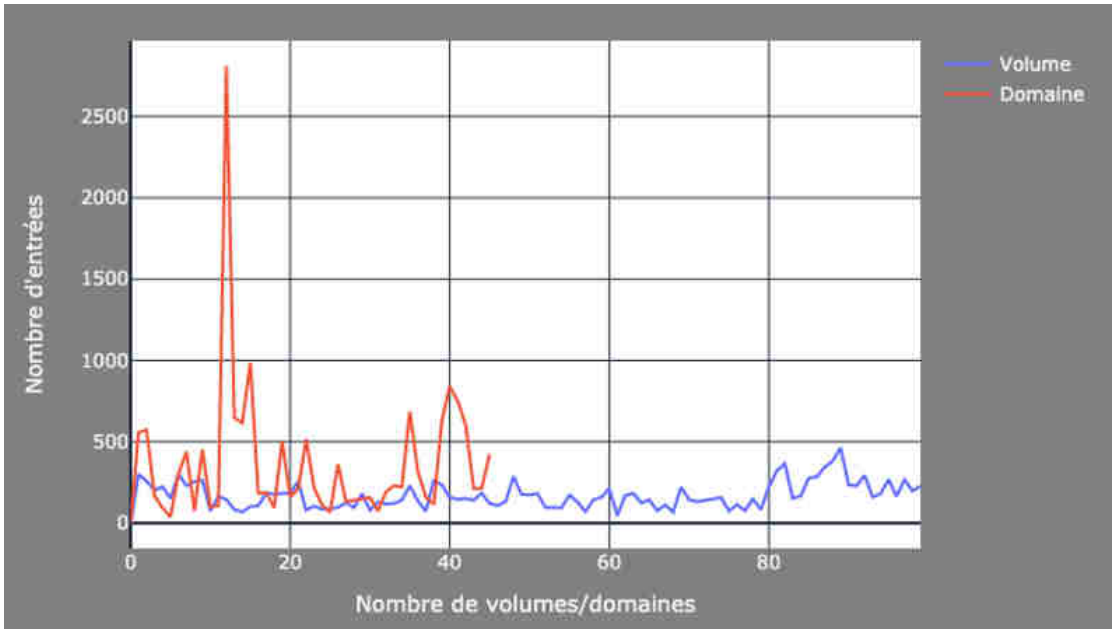

Figure 8 – Différence entre les deux méthodes de division

Du point de vue de la macrostructure, le *Yiwen Leiju* utilise une structure similaire que celle du Shuowen Jiezi : l'œuvre comprend 46 domaines (*juan*), chaque domaine étant composé de plusieurs sections (*bu*). La figure ci-dessous décrit précisément cette structure.

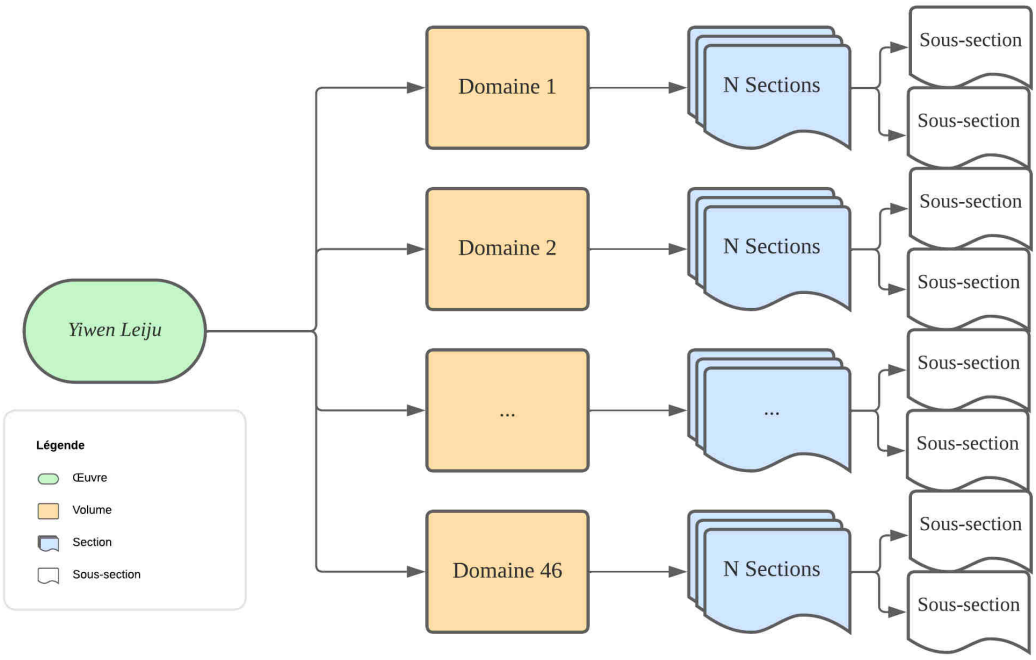

Figure 9 – Macrostructure du *Yiwen Leiju*

Cependant, il existe encore une petite différence entre les deux. Cette différence réside principalement à l'intérieur du niveau de section. Dans le *Shuowen Jiezi*, l'unité de base à l'intérieur de la section est une entrée (correspondant à l'élément TEI `<entry>`), ces éléments sont tous structurées en parallèle ; tandis que dans le *Yiwen Leiju*, l'unité de base à l'intérieur de la section est une citation (correspondant à l'élément TEI `<cit>`), ces citations ne sont pas structurées en parallèle.
Plus précisément, dans le *Yiwen Leiju*, la première partie est la définition de cette section. Cette partie n'est pas une définition créée par les auteurs eux-mêmes, mais plutôt une liste contenant plusieurs citations à d'autres œuvres, c'est-à-dire que les auteurs utilisent les citations comme définitions, toutes les citations dans cette partie appartiennent directement au niveau de section. La deuxième partie consiste en d'autres citations qui sont classées en sous-sections selon différents genres littéraires. Donc, les citations de la deuxième partie appartiennent directement au niveau de sous-section, qui est ensuite directement appartient au niveau de section.
La figure ci-dessous nous montre les différentes structures des deux au niveau de la section. Ce que nous pouvons facilement remarquer, c'est que pour le *Shuowen Jiezi*, les éléments de base sont complètement parallèles entre eux, tandis que pour le *Yiwen Leiju*, ce n'est pas le cas entre les éléments de base.

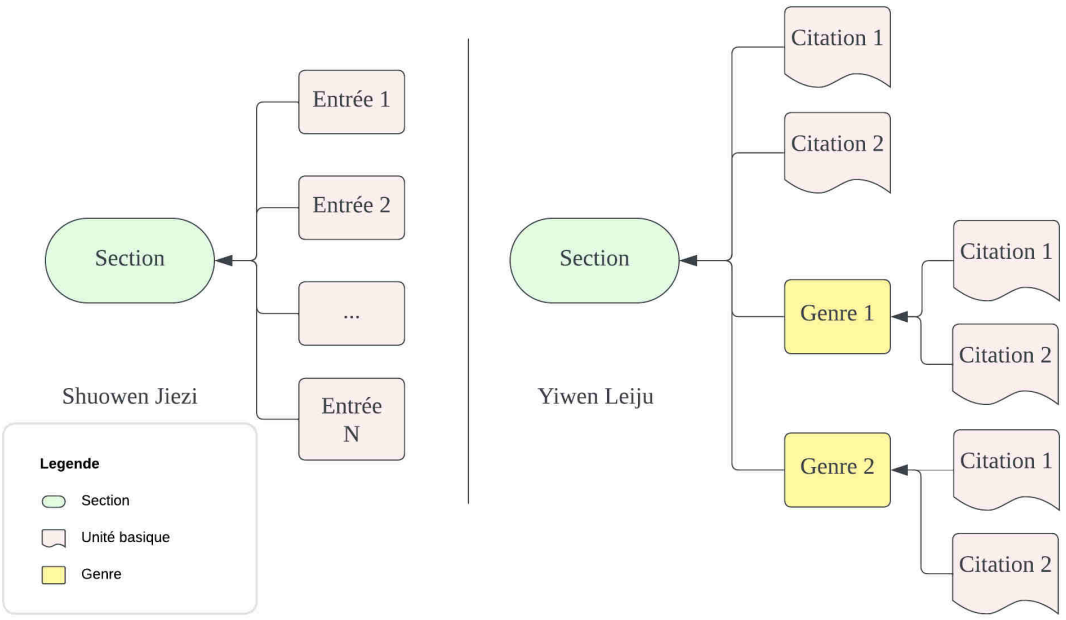

Figure 10 – Différence à l'intérieur de section

Si nous nous concentrons sur la microstructure du texte à l'échelle de la citation, nous constatons que nous avons affaire à des listes dont chaque élément est construit de manière uniforme. En effet, cette collecte de citations énonce, à chaque fois, dans la mesure du possible, les éléments bibliographiques, à savoir : la dynastie à laquelle l'auteur est associé, le nom de l'auteur et le titre de l'ouvrage cité. Cette première partie est reliée à la suivante, à savoir la citation proprement dite, à l'aide de mots outils dans un nombre limité. Autrement dit, on retrouve l'ensemble des métadonnées pertinentes pour retrouver la citation dans la première partie, puis un mot-outil, et enfin, le contenu de la citation.

Un autre point est à noter : quand une œuvre ou un auteur est cité de manière répétée, à partir du deuxième texte, le nom de l'œuvre ou de l'auteur ne sera pas mentionné et sera remplacé par un autre mot-outil, *you* 又 (à nouveau).

Pour toutes les structures mentionnées ci-dessus, nous pouvons les schématiser en utilisant les expressions régulières suivantes :

```
1 ^(.*?曰. )(.*)
```

3.2.2.2 Conception du schéma TEI

Lors de la conception de la structure du corpus, nous avons adopté une approche totalement similaire à celle du *Shuowen Jiezi*, c'est-à-dire que pour les niveaux supérieurs à la section (œuvre et volume), ils ne servent que de conteneurs de métadonnées fournissant des informations pertinentes sur le contenu du niveau inférieur qu'ils contiennent. La section (*bu* 部) est donc l'élément constitutif fondamental au niveau de macrostructure.

Et à l'intérieur de la section, le texte est directement inclus dans l'élément `<body>`, sous-élément de `<text>`. Nous utilisons les éléments suivants pour le texte :

`<entry>`
Ceci est le premier élément dans `<body>`, dont la fonction est de stocker le sujet traité, qui est représenté par un mot que l'on retrouve dans toutes les citations de l'entrée.

`<sense>`
Cet élément est utilisé pour stocker les premières citations de chaque section. Dans cette œuvre, les premières citations de chaque section sont directement liées à la section elle-même et ne sont associées à aucun genre littéraire particulier. C'est pourquoi nous utilisons cet élément pour sauvegarder ce contenu.

`<div>`
Une fois que les premières citations de la section est terminée, les citations restantes sont toutes incluses dans des sous-sections classées par genre. Nous utilisons donc la méthode d'encodage en utilisant l'élément div avec l'attribut type pour ces sous-sections. Par exemple, `<div type="shi">` est utilisé pour stocker les citations de genre poétique.

`<cit>`
Cet élément est l'unité de base du corpus, il sert de conteneur direct pour le texte. Cet élément comprend deux sous-éléments :

1. `<bibl>` : Cet élément est utilisé pour stocker l'auteur et le titre, le premier étant inclus dans le sous-élément `<author>`, tandis que le second est inclus dans `<title>`.
2. `<quote>` : Cet élément est utilisé pour stocker la citation.

La structure finale du corpus est la suivante :

```
<text>
<body>
  <entry>白鷺</entry>
  <sense>
    <cit type="collection" corresp="#TBD">
        <bibl><title>爾雅</title>曰</bibl>
        <quote>鷺舂鋤. </quote>
    </cit>
    <cit type="collection" corresp="#TBD">
        <bibl><title>毛詩</title></bibl>
        <quote>曰. </quote>
    </cit>
    <cit type="collection" corresp="#TBD">
        <bibl><title>詩義疏</title>曰</bibl>
        <quote>鷺. 水鳥也. 好而絜白. </quote>
    </cit>
  </sense>
  <div type=" 賦 ">
    <cit type="collection" corresp="#TBD">
        <bibl>宋<author>謝惠連</author><title>白鷺賦</title>曰</bibl>
        <quote>有提樊而見獻. </quote>
    </cit>
    </div>
  <div type=" 詩 ">
    <cit type="collection" corresp="#TBD">
        <bibl>陳<author>蘇子卿</author><title>鼓吹曲朱鷺
        ↪  詩</title>曰</bibl>
        <quote>玉山一朱鷺. </quote>
    </cit>
    </div>
</body>
</text>
```

3.3 Chaîne d'encodage automatique

L'ensemble de la chaîne de traitement du texte se divise en trois parties : la segmentation du texte, la génération de modèles et le remplissage des modèles avec du texte. Pour les deux dernières parties, nous utiliserons les fonctions de recherche dans des chaînes de caractères (telles que les expressions régulières). Pour la segmentation du texte, cependant, nous avons plusieurs options et il n'est pas possible de déterminer a priori laquelle permet d'obtenir les meilleurs résultats. Nous avons considéré plusieurs méthodes, que nous présentons dans cette section : la méthode basée sur des règles et les méthodes d'apprentissage automatique. Pour ces dernières, nous pouvons les subdiviser en deux catégories : les méthodes d'apprentissage supervisé et les méthodes d'apprentissage par transfert à partie des modèles pré-entraînés. Dans cette section, nous comparerons les avantages et les inconvénients des différentes méthodes. En effet, dans le cadre du pro-

jet CHI-KNOW-PO, j'ai été responsable de l'encodage du *Shuowen Jiezi* et du *Yiwen Leiju*. En raison de la possibilité d'utiliser presque le même code pour entraîner les modèles d'apprentissage automatique correspondant aux deux œuvres, nous n'avons utilisé que le code du *Shuowen Jiezi* pour présenter le modèle. Cependant, étant donné qu'il existe encore quelques différences entre les deux, dans cette section nous commencerons par présenter les choix différents que nous avons faits lors du prétraitement pour deux œuvres distinctes, puis nous illustrerons tous les traitements (symboliques, statistiques, neuronaux et leurs combinaisons).

3.3.1 Différences pendant le pré-traitement

L'analyse de la structure des deux ouvrages présentés précédemment permet de constater qu'ils diffèrent tant au niveau de la macrostructure que de la microstructure. Cependant, étant donné que notre modèle d'apprentissage automatique n'opère qu'au niveau de la microstructure, nous pouvons totalement ignorer la macrostructure à cette étape.

En ce qui concerne le *Shuowen Jiezi*, comme la répartition entre ses différentes parties n'est pas entièrement régulière, nous ne les traitons pas séparément. Cependant, pour le *Yiwen Leiju*, comme nous l'avons montré précédemment dans notre analyse, chaque citation peut être divisée en deux parties : les informations sur l'auteur ou l'œuvre et la citation elle-même. Notre principal traitement pour cet ouvrage consiste à séparer ces deux parties et à annoter les entités nommées, qui se trouvent exclusivement dans la première partie. Nous avons donc effectué les opérations suivantes : pour chaque citation, nous utilisons une expression régulière pour cibler sa première moitié (c'est-à-dire la partie contenant les informations sur l'auteur et l'œuvre), puis nous entraînons un modèle qui s'applique uniquement à cette partie. La raison de le faire est que seule la première partie contient les informations à traiter, la deuxième partie (le texte cité) n'a pas besoin d'être annotée.

3.3.2 Traitement basé sur des règles

Nous envisageons tout d'abord une approche automatisée basée sur des règles qui segmente le texte à l'aide d'expressions régulières, en classe les segments, crée un squelette de corpus vide à partir des résultats de la classification et, enfin, remplit le cadre avec le texte segmenté.

Comme les textes de ce projet sont des dictionnaires ou des encyclopédies, les textes à traiter sont bien structurés. Dans le contexte du dictionnaire, les mots-outils sont largement utilisés, ils peuvent être trouvés dans presque chaque partie de la plupart des entrées.

Grâce à ces mots-outils, il est plus facile de découper les entrées en plusieurs parties. Cependant, il est important de noter que certaines définitions n'incluent pas les mots-outils, il est donc impossible de segmenter le texte en utilisant uniquement des mots-outils. Lors de la conception de la méthode de segmentation du texte, deux options ont été envisagées : une méthode basée uniquement sur les mots-outils et une méthode de segmentation combinant la ponctuation et le mot-outil.

Tout d'abord, quelle que soit la manière dont une méthode de mise en correspondance

basée sur des règles est utilisée, il est impossible d'éviter l'étape de collecte des mots-outils. Pour cette étape, nous l'avons en réalité déjà accomplie dans l'analyse de l'œuvre.

Nous avons d'abord conçu une méthode de traitement entièrement basée sur les mots-outils : une liste de mots-outils est d'abord créée, puis chaque mot de l'entrée est parcouru pour extraire tous les mots-outils de l'entrée (en supposant un nombre de n). Ensuite, l'entrée est divisée en n parties. Enfin, nous déterminons le type (définition, étymologie, etc.) de chacune de ces n parties. Nous allons démontrer cette approche en détail en utilisant l'exemple d'entrée précédemment utilisé : 禧：禮吉也。从示喜聲。

Une fois que nous avons la liste des mots-outils, nous allons appliquer la première méthode. Nous allons d'abord utiliser Python pour itérer à travers les entrées, puis extraire les mots-outils dans les entrées. Pour l'entrée de l'exemple, nous obtenons la liste suivante : 也从聲 . Nous pouvons ensuite construire une expression régulière pour diviser le texte sur la base de cette liste : (.：)(.* 也。)(从 .)(. 聲。). À l'aide de cette expression régulière, nous pouvons diviser l'entrée en quatre parties et déterminer le type de chaque partie en fonction du mot-outil.

Cependant, cette approche présente de sérieux inconvénients : elle a du mal à traiter les phrases longues et complexes et ne permet pas de traiter les segments qui ne contiennent pas de mot-outil. Pour remédier à cette lacune, nous avons développé une autre méthode qui s'appuie sur la ponctuation et qui est utilisée en combinaison avec les mots-outils pour améliorer la qualité de la segmentation du texte.

Cette méthode améliorée effectue la segmentation du texte en deux étapes : lors de la première segmentation, nous divisons une entrée en plusieurs parties à l'aide des ponctuations. Nous définissons tout d'abord une fonction permettant de détecter et d'extraire les ponctuations des entrées :

```
import re
list_punctuation = ['。',', ']
def extract_punctuation(entry):
    regex = '|'.join(list_punctuation)
    pattern = re.compile(regex)
    punctuations = pattern.findall(entry)
    return punctuations
```

Nous parcourons ensuite les entrées, en extrayant chaque virgule et chaque point pour chaque entrée, en la faisant précéder d'une expression régulière pour un nombre quelconque de caractères arbitraires, et en mettant l'expression régulière entre parenthèses lorsqu'un point est présent (dans les expressions régulières, cela représente un groupe, et c'est au moyen de groupes que nous segmentons le texte). Enfin, nous obtenons l'expression régulière pour chaque entrée.

```
import re
list_punctuation = ['。',', ']
import pandas as pd
with open('text.txt','r',encoding='utf-8') as f :
```

```
    for line in f.readlines() :
        toolwords = extract_toolwords(line)
        regex_final = ''
        list_text.append(line)
        if toolwords:
            regex_part = ''
            for i in toolwords :
                if i != 'o ' :
                    regex_part += '.*'+i
                else:
                    regex_part += '.*'+i
                    regex_part = '(' + regex_part + ')'
                    regex_final += regex_part
                    regex_part = ''
            regex_final = '(.: )' + regex_final
            list_regex.append(regex_final)
        else:
            regex_final = '(' + line[:-1] + ')'
            list_regex.append(regex_final)


    dict_text_regex.update({'text':list_text,'regex_split':list_regex})
    df = pd.DataFrame(dict_text_regex)
```

À la fin du code ci-dessus, nous avons converti les données collectées dans un format de *DataFrame* pour la visualisation.

text	regex
卷一 一部 一：惟初太始，道立於一，造分天地，化成萬物。凡一之屬皆从一。 元：始也。从一从兀。 天：顛也。至高無上，从一、大。 ... □□：酒器也。从酋，廾以奉之。《周禮》六尊：犧尊、象尊、著尊、壺尊、太尊、山尊，以待祭祀賓客之 ...	(卷一) (一部) (.：)(.*，.*，.*，.*。)(.*。) (.：)(.*。)(.*。) (.：)(.*。)(.*，.*。) ... (.：)(.*。)(.*，.*。)(.*，.*。)
戊部 戊：滅也。九月，陽气微，萬物畢成，陽下入地也。五行，土生於戊，盛於戊。从戊含一。凡戊之屬皆从 ...	(戊部) (.：)(.*。)(.*，.*，.*，.*。)(.*，.*，.*。)(.*。)(.*。)
亥部 亥：荄也。十月，微陽起，接盛陰。从二，二，古文上字。一人男，一人女也。从乙，象褢子咳咳之形。...	(亥部) (.：)(.*。)(.*，.*，.*。)(.*，.*，.*。)(.*，.*。)(.*，.*。...

À ce stade, nous avons effectué notre première segmentation, basée sur le point final. Nous poursuivons ensuite avec une segmentation plus détaillée du texte, cette fois à l'aide de mots-outils, et nous nous concentrons sur les deux parties du texte qui sont souvent collées l'une à l'autre : l'étymologie et la prononciation.

Le code commence par parcourir toutes les entrées, en créant une liste temporaire à chaque itération. Ensuite, si la partie ne contient pas de mots-outils liés à l'étymologie, à la prononciation ou à la référence, le code stockera la partie directement dans la liste temporaire, tandis que pour les parties précédemment mentionnées, le code les traitera plus en détail et les stockera ensuite dans la liste temporaire. Une fois que chaque partie de l'entrée a été vérifiée ou traitée par le code, celui-ci stocke le contenu de la liste temporaire dans son ensemble dans la liste finale et répète le processus pour chaque entrée suivante.

```
import re
list_split_text = []
for a,b in zip(list_text,list_regex):
    list_singal_text = []
    regex = re.compile(b)
    split = re.search(regex,a)
```

```
    if split :
        for i in split.groups():
            if '从' in i and '聲' in i :
                if ',' not in i and ';' not in i and '、' not in i :
                    regex = re.compile(r'(从 .)(. 聲。)')
                    i_split = re.search(regex,i)
                    if i_split:
                        for a in i_split.groups() :
                            list_singal_text.append(a)
                else :
                    regex = re.compile(r'(从 .*(, |; ))(. 聲。)')
                    i_split = re.search(regex,i)
                    if i_split:
                        for a in i_split.groups() :
                            if a != ', ' and a != '; ' :
                                list_singal_text.append(a)
            elif "《" in i and "》" in i and ": " in i :
                i+='” '
                i_split = re.search(r'(.*》)(.*)',i)
                for a in i_split.groups() :
                    list_singal_text.append(a)
            else :
                list_singal_text.append(i)
        list_split_text.append(list_singal_text)
for i in list_split_text :
    print(i)
```

Nous obtiendrons un meilleur résultat de segmentation :

```
['卷一 ']
['一部']
['一：', '惟初太始，道立於一，造分天地，化成萬物。', '凡一之屬皆从一。']
['元：', '始也。', '从一从兀。']
['天：', '顛也。', '至高無上，从一、大。']
...
['亥部']
['亥：', '荄也。', '十月，微陽起，接盛陰。', ...'” 凡亥之屬皆从亥。']
```

Après avoir segmenté le texte avec succès, nous avons effectivement rempli la moitié des conditions de production du corpus, l'autre moitié étant les informations de composition de chaque entrée (c'est-à-dire le type d'éléments constitutifs de chaque entrée), et ce n'est qu'avec les informations de composition que nous pouvons automatiser le processus de l'annotation de séquences. Pour cette étape, nous utilisons également les mots-outils : nous vérifions les mots-outils contenus dans les parties segmentées et déterminons leur type en fonction des mots qu'elles contiennent.

```
#get the type of every part
import re
list_type = []
for list in list_split_text :
    list_component = []
    for component in list :
        if re.match(r'.：$', component) :
            list_component.append('head')
        elif re.match(r'. 部', component) :
            list_component.append('section')
        elif re.match(r'卷 .{1,2}', component) :
            list_component.append('volume')
        elif re.match(r'.* 也。', component) :
            list_component.append('definition')
        elif re.match(r'从 .*', component) :
            list_component.append('etymology')
        elif re.match(r'凡 . 之屬皆从 .*', component) :
            list_component.append('etymology')
        elif re.match(r'.* 聲', component) :
            list_component.append('pronunciation')
        elif '讀若' in component :
            list_component.append('pronunciation')
        elif '一曰' in component :
            list_component.append('definition')
        elif '《' in component and '》' in component :
            list_component.append('reference')
        elif '：' in component :
            list_component.append('citation')
        else :
            list_component.append('definition')
    list_type.append(list_component)
dict_final = {}
dict_final.update({'text split' : list_split_text,
                   'type':list_type})
df2 = pd.DataFrame(dict_final)
display(df2)
```

Finalement nous aurons un jeu de données contenant les textes segmentés et leurs types :

text	type
卷一	volume
一部	section
一：, 惟初太始，道立於一，造分天地，化成萬物。, 凡一之屬皆从一。	head, definition, etymology
元：, 始也。, 从一从兀。	head, definition, etymology
天：, 顛也。, 至高無上，从一、大。	[head, definition, definition
...	...
□□：, 酒器也。, 从酋，廾以奉之。,《周禮》, 六尊：犧尊、象尊、著尊、壺尊、太尊、山 ...	[head, definition, etymology, reference, citation...
戊部	section
戊：, 滅也。, 九月，陽气微，萬物畢成，陽下入地也。, 五行，土生於戊，盛於戌。, 从戊 ...	head, definition, definition, definition, etymology...
亥部	section
亥：, 荄也。, 十月，微陽起，接盛陰。, 从二，二，古文上字。, 一人男，一人女也。, ...	head, definition, definition, etymology, definition...

En résumé, l'utilisation d'une approche basée sur des règles permet de segmenter un texte très rapidement sans étapes supplémentaires, mais l'inconvénient de cette approche est que la complexité des règles varie complètement en fonction du texte. Pour les textes volumineux, il se peut que nous ne soyons pas en mesure de créer une règle pour chaque cas. En fait, même le texte de ce projet présente une structure difficile à exprimer à l'aide de règles basées sur des expressions régulières et des mots-outils, c'est pourquoi nous avons essayé une approche plus flexible en plus de celle-ci, basée sur l'apprentissage machine.

3.3.3 Traitement avec apprentissage automatique

Outre l'approche fondée sur des règles, nous avons également envisagé l'utilisation de l'apprentissage automatique pour la segmentation des entrées. L'avantage de cette approche est que les réseaux neuronaux voire les algorithmes classiques ont tendance à être plus performants lorsqu'ils traitent des structures complexes. Nous avons conçu une variété de méthodes de traitement basées sur l'apprentissage automatique, que nous avons subdivisées en deux grandes catégories : les méthodes basées sur l'apprentissage automatique et celles basées sur l'apprentissage profond.

3.3.3.1 Création du corpus d'entraînement

Avant d'introduire les méthodes d'apprentissage automatique, nous devons d'abord présenter la plateforme d'annotation de données que nous utilisons ainsi que les critères d'annotation.

Dans ce projet, le processus d'annotation consiste en fait à diviser manuellement les différentes parties du texte et à étiqueter chaque partie, afin que l'algorithme puisse apprendre à classer les différentes parties lors de l'entraînement du modèle sur la base des données annotées manuellement. Nous avons d'abord supprimé du texte original tout

texte qui ne devait pas être classé, comme le numéro dans chaque volume.

Lors de la sélection des données d'entraînement, nous avons d'abord effectué une analyse basée sur les types des entrées dans le texte pour éviter un déséquilibre dans le nombre de types d'entrées dans le corpus d'entraînement. Ceci est important avec un texte comme le *Shuowen Jiezi*, où plus de la moitié des entrées correspondent à l'expression régulière suivante :

```
.:.*也。从..聲。( 讀若.*){0,1}
```

Si nous choisissons des entrées de manière aléatoire, alors le même type d'entrée inondera massivement les données d'entraînement, ce qui pourrait empêcher l'algorithme d'apprendre suffisamment de données d'autres types. Pour éviter ce problème, nous avons défini un seuil pour limiter la proportion des types d'entrées qui sont largement représentés dans le texte original. Pour les données restantes, nous les avons sélectionnées au hasard.

L'outil que nous avons utilisé pour étiqueter les données est Label Studio [6] (version 1.8.2), qui présente l'avantage d'offrir une interface visuelle et une variété de formats d'exportation, ce qui permet de l'utiliser en conjonction avec une variété d'autres outils. Comme notre tâche est l'annotation de séquences, mais qu'il n'y a pas de modèles disponibles dans Label Studio pour ce genre de tâche, nous avons choisi la reconnaissance d'entités nommées comme modèle le plus proche de la tâche d'annotation de séquences.

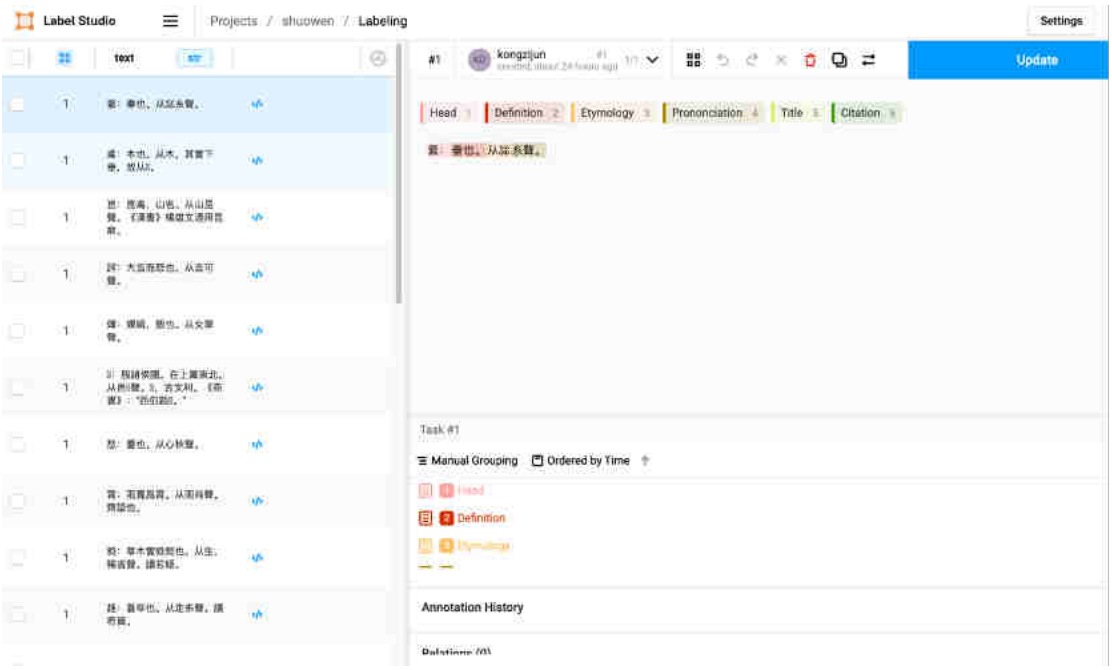

Figure 11 – Interface utilisateur de Label Studio

Nous avons ajouté manuellement six étiquettes dans Label Studio : *head*, *definition*,

6. URL : https ://labelstud.io/ (accès le 1er août 2023).

etymology, *pronunciation*, *title*, *citation*, qui correspondent à chacune des six classes. Le processus d'étiquetage suit les règles suivantes :

1. Tout caractère (que nous assimilons à un mot) doit appartenir à une catégorie. Pour la ponctuation, lors de l'annotation, nous étendons toutes les balises vers la droite pour inclure la ponctuation qui suit.
2. Essayer d'éviter les annotations transversales, par exemple, pour cette entrée : 鶿 : 小羊也。从羊大聲 . 讀若達。Cette entrée est très représentative : elle contient trois propositions : « 小羊也 », « 从羊大聲 » et « 讀若達 », dont la deuxième et la troisième incluent toutes deux une prononciation. Autrement dit, dans cette entrée, les informations de prononciation traversent en réalité deux phrases. Dans ce cas, nous devons les annoter séparément car pour la deuxième phrase, le mot-outil se trouve à la fin du segment, tandis que le mot-outil pour la troisième proposition se trouve au début du segment. Si nous considérons ces deux parties comme un ensemble lors de l'annotation, nous perdrons finalement des informations sur les mots-outils. Il y a plusieurs exemples similaires dans le texte d'origine, c'est pourquoi nous avons décidé d'utiliser les propositions comme unité de base lors de l'annotation et d'interdire toute annotation à cheval sur plusieurs propositions.
3. Pour l'annotation des données, nous utilisons la méthode BIO, une méthode fréquemment utilisée pour les tâches de reconnaissance d'entités nommées. Dans la méthode BIO, nous attribuons deux sous-étiquettes à chaque classe : B (*Begin*) et I (*Inside*), et pour le texte qui n'appartient à aucune classe, nous le marquons avec l'étiquette O (Outside). L'utilisation de cette méthode permet aux algorithmes de mieux comprendre la structure des entités nommées. Par exemple, pour cette phrase : « Lucas habite actuellement à Villefranche sur Saône », selon la méthode BIO, elle sera annotée comme le suivant :

Texte	Étiquette
Lucas	B-PER
habite	O
actuellement	O
à	O
Villefranche	B-LOC
sur	I-LOC
Saône	I-LOC

Nous avons suivi les règles ci-dessus pour effectuer le travail d'annotation. Pour le *Shuowen Jiezi*, nous avons annoté 800 entrées et pour le *Yiwen Leiju*, nous avons annoté 500 citations.

Une fois l'annotation terminée, nous pouvons exporter les données d'entraînement, ce que nous avons choisi de faire au format JSON. Les données à ce format ne peuvent pas être utilisées directement pour l'entraînement du modèle, mais la structure de données JSON nous permet d'extraire et de transformer les données avec très peu de code.

De plus, nous avons utilisé la méthode de validation croisée pour pallier au manque de données d'entraînement. La validation croisée est une technique qui permet d'évaluer les performances des modèles d'apprentissage automatique et qui peut aider à évaluer la capacité de généralisation du modèle lorsque la quantité de données est limitée. Dans ce projet, nous avons opté pour une validation croisée à k plis, c'est-à-dire que les données étiquetées ont été divisées en k sous-ensembles distincts ; chaque sous-ensemble a été utilisé tour à tour comme ensemble d'évaluation tandis que les autres ont servi à entraîner le modèle.

3.3.3.2 Apprentissage automatique

CRF

Lorsque nous utilisons le CRF pour l'annotation de séquences, le modèle CRF doit apprendre les caractéristiques des éléments (dans notre projet, il s'agit de caractères) plutôt que de l'élément lui-même. Par conséquent, nous devons construire manuellement les caractéristiques pour chaque caractère. Nous avons conçu le dictionnaire de caractéristiques suivant : le caractère lui-même, s'il est le premier ou le dernier dans une séquence (c'est une valeur booléenne), le caractère précédent et le caractère suivant. Ces cinq caractéristiques prennent en compte le contexte et la position relative de chaque caractère.

Pour améliorer la capacité de CRF à apprendre les modèles à partir des données, nous devons apporter certaines modifications aux données d'entraînement. Comme mentionné précédemment, Label Studio ne fournit pas de modèle pour les tâches d'annotation de séquences. Par conséquent, les résultats d'annotation que nous avons obtenus dans les données d'entraînement finales sont les suivants :

```
"label": [
        {
          "start": 0,
          "end": 2,
          "text": " 縈: ",
          "labels": [
            "Head"
          ]
        },
        {
          "start": 2,
          "end": 5,
          "text": " 垂也。",
          "labels": [
            "Definition"
          ]
       }
```

Les données dans ce format ne peuvent pas être utilisées directement pour l'apprentissage automatique. Nous avons donc extrait le texte et les étiquettes du fichier JSON à

l'aide de quelques lignes de code simples et réétiqueté le texte à l'aide de la méthode BIO.

L'algorithme CRF exige que nous fournissions deux types de données : les caractéristiques et les étiquettes. Nous avons déjà obtenu les étiquettes, tandis que les caractéristiques peuvent inclure plusieurs types d'information. En général, plus de caractéristiques sont bénéfiques pour l'apprentissage du modèle. En plus du texte lui-même, nous avons également ajouté les informations suivantes dans les caractéristiques : « is_first » (si le caractère est de la sous-étiquette « B »), « is_last » (si le caractère est le dernier de la sous-étiquette « I »), « prev_char » (le caractère précédent) et « next_char » (le caractère suivant). En combinant ces deux types de données, nous obtenons notre ensemble de données.

```
#read json
with open('PATH TO JSON', 'r', encoding='utf-8') as f:
    data_list = json.load(f)

# extract features and labels
all_features = []
all_labels = []

for data in data_list:
    text = data['text']
    labels = data['label']
    tag_sequence = ['O'] * len(text)

    for label in labels:
        start, end, type = label['start'], label['end'],
        ↪  label['labels'][0]
        tag_sequence[start] = f'B-{type}'
        for i in range(start + 1, end):
            tag_sequence[i] = f'I-{type}'

    features = []
    for i, char in enumerate(text):
        feature = {
            'char': char,
            'is_first': i == 0,
            'is_last': i == len(text) - 1,
            'prev_char': text[i - 1] if i > 0 else '',
           'next_char': text[i + 1] if i < len(text) - 1 else '',
        }
        features.append(feature)

    all_features.append(features)
    all_labels.append(tag_sequence)
```

Les données que nous avons présentées précédemment (p. 48) seront transformées comme

suit après l'ajout de la méthode BIO. Ce type de données est par ailleurs la forme standard que nous utilisons ultérieurement pour entraîner les réseaux neuronaux) :

```
繁: 垂也。
['B-Head', 'I-Head', 'B-Definition', 'I-Definition', 'I-Definition']
```

Après cela, le code continuera à traiter la partie texte en construisant une caractéristique pour chaque caractère. Prenons l'exemple du premier caractère « 繁 » dans le texte, il deviendra finalement :

```
{'char': '繁',
'is_first': True,
'is_last': False,
'prev_char': '',
'next_char': ': '}
```

Une fois que nous avons obtenu notre ensemble de données, nous avons ensuite entraîné le modèle avec 800 entrées des 9831 entrées au total, l'objectif étant de vérifier que la structure CRF fonctionnait bien avec une très petite quantité de données d'entraînement :

```
# Split
 X_train, X_test, y_train, y_test = train_test_split(all_features,
  ↪  all_labels, test_size=0.2)

# Fit the model
crf = CRF(
    algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=True
 )
crf.fit(X_train, y_train)
```

Nous avons utilisé la bibliothèque sklearn-crfsuite [7] (version 0.3.6) pour entraîner le modèle. Au début du code ci-dessus, nous utilisons une méthode de division aléatoire pour diviser les données collectées en deux parties : les données d'entraînement et les données de validation. Après l'entraînement, nous avons évalué les performances du modèle en utilisant les données de validation. Les résultats, encourageants, seront présentés dans la section « Résultats et discussion » (p. 70 et suivantes).

Par la suite, pour vérifier si les bonnes performances étaient limitées à certaines données spécifiques, les performances du modèle CRF ont été validées à l'aide d'une méthode de validation croisée. Dans la deuxième phase, nous avons fixé la valeur de k à 5, c'est-à-dire que nous avons divisé les données en 5 plis, en utilisant 4 d'entre eux pour

7. URL : https ://github.com/TeamHG-Memex/sklearn-crfsuite (accès le 1er août 2023).

l'entraînement du modèle à chaque fois et le dernier pli restant pour la validation. Le processus est donc répété 5 fois.

3.3.3.3 Apprentissage profond

La performance globale de CRF est bonne, mais sa performance dans des textes relativement complexes a légèrement diminué. Nous souhaitons donc essayer d'utiliser un réseau neuronal pour améliorer davantage les performances de l'annotation automatique.

Pour les réseaux neuronaux, le processus est similaire à celui du CRF. Tout d'abord, nous construisons des caractéristiques pour chaque caractère. Dans le CRF, il s'agit d'un dictionnaire de caractéristiques, tandis que dans les réseaux neuronaux, il s'agit des *word embeddings*. Ensuite, nous construisons un réseau neuronal. Le processus de construction du réseau neuronal est similaire : tout d'abord, nous définissons une classe Python qui comprend au moins deux méthodes : l'une est la méthode d'initialisation où nous définissons les variables nécessaires telles que la taille de l'état caché dans LSTM ; l'autre est la méthode de propagation avant (*forward propagation*) qui définit comment les données circulent entre différentes couches du réseau neuronal. Pour construire des réseaux neuronaux, nous utilisons PyTorch [8] (version 2.0.1), un *framework* d'apprentissage profond très populaire. Il offre une prise en charge intégrée pour tous les principaux types de réseaux neuronaux. Grâce à PyTorch, nous pouvons construire un réseau neuronal avec peu de code.

Dans cette section, nous allons présenter le processus de construction des word embeddings, parce que cette étape est commune à tous les réseaux neuronaux dans notre projet.

Tout d'abord, nous chargeons le modèle de Hugging Face ainsi que son *tokenizer* correspondant.

```
from transformers import AutoTokenizer, AutoModel

tokenizer = AutoTokenizer.from_pretrained("SIKU-BERT/sikuroberta")
model = AutoModel.from_pretrained("SIKU-BERT/sikuroberta")
```

Ensuite, nous avons construit une fonction appelée `roberta_embeddings` qui lit le texte et ses étiquettes correspondantes. Ensuite, elle utilise un modèle pré-entraîné pour convertir le texte en vecteurs de haute dimension, c'est-à-dire des *word embeddings*. Il est important de noter que dans la définition de la variable inputs, nous utilisons deux paramètres : `trunction` et `padding`. Cela signifie que si le texte dépasse la longueur maximale définie, il sera tronqué directement. Si sa longueur est inférieure à la longueur maximale, elle sera remplie avec une marque spéciale `[PAD]`. Nous effectuons également cette opération sur les étiquettes à la fin, mais la valeur de remplissage pour les étiquettes est -100. La raison de le faire est que lorsque nous voulons utiliser un réseau neuronal pour apprendre simultanément plusieurs séquences, nous devons nous assurer que ces séquences ont la même longueur.

8. URL : https ://pytorch.org/ (accès le 1er août 2023).

```
def Roberta_embeddings(text, tag, max_length=20):
    inputs = tokenizer(text, max_length=max_length, truncation=True,
    ↪  padding="max_length", return_tensors="pt",
    ↪  is_split_into_words=False)

    outputs = model(**inputs)
    embeddings = outputs[0].squeeze().detach()

    if len(tag) > max_length:
        tag_padded = tag[:max_length]
    else:
        tag_padded = tag + [-100] * (max_length - len(tag))

    return embeddings, torch.tensor(tag_padded)
```

BiLSTM

Comme nous ne disposons pas de beaucoup de données d'entraînement, nous utiliserons encore un mécanisme de validation croisée. Nous devons tout d'abord traiter les données exportées depuis Label Studio comme nous l'avons fait pour l'entraînement du modèle CRF, la différence est que nous n'avons plus besoin de construire manuellement des caractéristiques. Pour la structure BiLSTM, il suffit d'extraire le texte et les étiquettes correspondantes (la méthode BIO doit être appliquée comme précédemment). Jusqu'à présent, notre combinaison de texte-étiquette est entièrement en texte brut. Notre texte est composé de caractères chinois et de ponctuation, et nos étiquettes sont des mots anglais utilisant la méthode BIO, comme indiqué ci-dessous :

```
縈: 垂也。
['B-Head', 'I-Head', 'B-Definition', 'I-Definition', 'I-Definition']
```

Ensuite, nous devons encoder les étiquettes. Nous avons créé un dictionnaire tag2vec où chaque étiquette est attribuée à un code numérique unique :

```
tag2vec = {"B-Head" : 0,
           "I-Head":1,
           "B-Definition":2,
           "I-Definition":3,
           "B-Etymology":4,
           "I-Etymology":5,
           "B-Pronunciation":6,
           "I-Pronunciation":7,
           "B-Title":8,
           "I-Title":9,
           "B-Citation":10,
           "I-Citation":11}
```

Après cela, nous remplacerons les étiquettes dans l'exemple ci-dessus par des codes numériques. Cette conversion est pour faciliter notre traitement ultérieur.

```
繁: 垂也。
[0, 1, 2, 3, 3]
```

Ensuite, nous introduisons les données de ce format dans la fonction`roberta_embeddings` définie avant. Cette fonction crée des *word embeddings* pour chaque caractère du texte et convertit le codage numérique des étiquettes en tenseur. La fonction finira par renvoyer une liste, dont le premier élément est les *embeddings* de chaque mot. Dans notre exemple, si nous essayons d'imprimer sa forme, nous constaterons que la valeur renvoyée est : `torch.Size([5, 768])`, ce qui signifie que notre fonction a réussi à créer un vecteur à 768 dimensions pour chaque caractère du texte.

L'*embedding* de l'entrée « 繁 » qui nous sert d'exemple correspondent aux valeurs suivantes. Les données brutes contiennent 768 valeurs, ici nous avons omis la plupart d'entre elles et n'avons conservé que le début et la fin.

```
tensor([-3.8772e-01,  5.8578e-01, -5.9371e-02,  3.8765e-01,
 ↪  -2.4388e-01,
         2.0084e-01, -4.3315e-01,  2.7905e-01, -8.4516e-01,
          ↪  1.8266e-01,
         1.7410e-01,  1.7683e-01,  8.3419e-01, -5.4319e-01,
          ↪  -1.3706e+00,
         ... ... ... ... ...
         4.0668e-02,  1.4225e-01,  4.0605e-01,  1.5356e-01,
          ↪  9.2958e-01,
         5.0643e-01, -2.0257e-01, -4.2690e-01,  5.8263e-01,
          ↪  -1.7068e-01,
        -6.2314e-01, -7.3090e-01,  1.0349e+00, -2.5221e-01,
         ↪  -8.4324e-01,
        -2.2518e-01, -4.0629e-01, -1.7643e-01])
```

À ce stade, nous avons terminé toutes les préparations de données, nous pouvons commencer à définir le modèle BiLSTM.

```
class BiLSTM(nn.Module):
    def __init__(self, size_in, size_hidden, size_out):
        super().__init__()
        self.lstm = nn.LSTM(size_in, size_hidden,
         ↪  bidirectional=True)
        self.linear = nn.Linear(size_hidden * 2, size_out)

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        linear_out = self.linear(lstm_out)
        return linear_out
```

```
11
12 model = BiLSTM(768, 128, 12)
```

Dans le code ci-dessus, nous avons défini une classe python appelée BiLSTM. Cette classe contient deux méthodes. Dans sa méthode d'initialisation, nous passons trois variables : `size_in`, `size_hidden` et `size_out`. Ces trois variables correspondent respectivement aux dimensions des données d'entrée du modèle BiLSTM, à la taille de l'état caché du modèle BiLSTM et aux dimensions des données de sortie du modèle. Nous définissons également deux composants de modèle dans cette méthode : le modèle LSTM lui-même, qui accepte les variables `size_in` et `size_hidden` que nous avons définies, et nous activons le paramètre `bidirectional` pour convertir le modèle en un modèle BiLSTM. En plus du modèle BiLSTM, nous définissons également une couche linéaire dont le rôle est de projeter la sortie du modèle vers 12 étiquettes différentes. Ainsi, pour chaque caractère, il y aura un score de probabilité pour chaque catégorie possible afin que nous puissions déterminer à quelle catégorie précise ce caractère appartient en fonction de ce score. Ensuite, dans la deuxième méthode de cette classe, nous définissons la propagation avant du modèle. Cette méthode décide comment le modèle est construit et comment les données circulent à travers celui-ci. Dans notre modèle, les données passeront d'abord par le calcul du modèle BiLSTM puis seront ensuite traitées par la couche linéaire afin d'obtenir les probabilités correspondant à chaque catégorie pour les données fournies.

Dans la dernière ligne de code, nous avons instancié le modèle. Ici, les valeurs de nos trois variables sont respectivement 768, 128 et 12. En réalité, parmi ces trois variables, `size_in` et `size_out` sont fixes et ne peuvent pas être modifiées. En effet, `size_in` correspond à la dimension des données d'entrée du modèle. Comme nous utilisons des *word embeddings* comme entrée du modèle, et RoBERTa qui ont une dimension fixe de 768, la valeur de `size_in` doit donc être égale à 768. De même, `size_out` correspond à la dimension de sortie de la couche linéaire dont le but est de projeter les sorties du modèle vers chaque catégorie possible. Étant donné que nous n'avons que 12 catégories, `size_out` doit être égal à 12. Le seul paramètre modifiable est donc `size_hidden`.

Ensuite, nous procédons à la dernière étape avant d'entraîner le modèle : créer le jeu de données et instancier le modèle. Nous utilisons également 80% des données pour l'en-traînement, les 20% restants sont utilisés pour la validation. À ce stade, toutes les préparations pour le modèle sont terminées. Il ne reste plus qu'à combiner le code de validation croisée avec le code d'entraînement pour entraîner le modèle.

BiLSTM-CRF

Dans la pratique du traitement automatique des langues, il est courant de combiner un modèle BiLSTM avec un modèle CRF pour les tâches d'annotation de séquences, une structure qui combine la mémoire contextuelle du modèle BiLSTM avec la capacité du modèle CRF à traiter des séquences complexes. Et comme nous avons déjà testé des modèles CRF simples et que même les modèles CRF simples ont bien fonctionné dans la tâche d'annotation de séquences de ce projet, nous avons décidé d'essayer d'ajouter

une couche CRF au modèle BiLSTM afin d'améliorer ses performances du modèle. En raison de l'absence d'une implémentation du CRF dans PyTorch, nous utilisons une bibliothèque tierce appelée TorchCRF [9] (version 1.1.0) pour intégrer la couche CRF dans le réseau neuronal.

Nous avons défini une nouvelle classe `BiLSTM_CRF`, qui ajoute une couche CRF à la classe BiLSTM originale. Le fragment du code suivant montre comment nous avons procédé :

```
class BiLSTM_CRF(nn.Module) :
    def __init__(self,size_in,size_hidden,size_out):
        super.__init__()
        self.lstm = nn.LSTM(size_in, size_hidden,
        ↪  bidirectional=True)
        self.linear = nn.Linear(size_hidden * 2, size_out)
        self.crf = CRF(size_out, batch_first=True)

    def forward(self,x,tags=None):
        lstm_out, _ = self.lstm(x)
        linear_out = self.linear(lstm_out)
        if tags is not None:
            mask = (tags != -100)
            tags = torch.where(tags == -100, torch.tensor(0,
            ↪  device=tags.device), tags)
            loss = -self.crf(linear_out, tags, mask=mask)
            return loss
        else:
            prediction = self.crf.decode(linear_out)
            return prediction

model = BiLSTM_CRF(768, 128, 12)
```

Lors de l'ajout de la couche CRF, nous avons constaté que cette couche ne pouvait pas identifier automatiquement la valeur -100 utilisée pour le *padding* des étiquettes. Donc, nous avons créé un tenseur ayant la même forme que tags. Dans ce tenseur, nous indiquons par des valeurs booléennes si une étiquette donnée est une valeur de *padding*. La couche CRF prendra en compte ce tenseur et, lors du calcul de la fonction de coût (*loss function*), elle ignorera les parties.

Transformer

Lors de l'entraînement du modèle de Transformer, nous pouvons réutiliser une partie du code du modèle BiLSTM précédent, soit : le code pour lire le fichier JSON et obtenir les données, l'encodage des caractères et l'encodage des balises.

Nous avons ensuite défini une classe pour l'encodage de position. Comme le Trans-

9. URL : https ://github.com/s14t284/TorchCRF (accès le 1er août 2023).

former traite toutes les données en parallèle, il n'a pas la capacité de se souvenir de l'ordre des données et nous devons donc encoder la position de chaque élément.

```
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=30):
        super().__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len,
        ↪  dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() *
        ↪  (-math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0), :]
        return x
```

Une fois l'encodage de position terminé, nous pouvons définir la dernière classe : le Transformer lui-même. À ce stade, nous avons achevé la construction du modèle.

```
class Transformer(nn.Module):
    def __init__(self, size_in, size_out, num_heads,
    ↪  num_encoder_layers):
        super().__init__()
        self.positional_encoding = PositionalEncoding(size_in)
        self.transformer_encoder =
        ↪  nn.TransformerEncoder(nn.TransformerEncoderLayer(size_in,
        ↪  num_heads), num_encoder_layers)
        self.linear = nn.Linear(size_in, size_out)

    def forward(self, x):
        x = self.positional_encoding(x)
        x = self.transformer_encoder(x)
        x = self.linear(x)
        return x
```

Cette classe Python est similaire à la classe BiLSTM précédente. Nous avons également défini deux méthodes dedans : une méthode d'initialisation pour recevoir les paramètres que nous définissons et construire les couches du modèle. Dans cette méthode, nous avons défini trois couches : une couche d'encodage de position, un encodeur de Transformer et une couche linéaire. Nous n'avons pas défini de décodeur car notre tâche ne concerne pas la génération de texte.

Enfin, comme nous l'avons fait précédemment, nous utilisons la validation croisée pour entraîner les modèles et évaluer leurs performances.

Transformer-CRF

Sur la base de l'expérience précédente, nous avons à nouveau essayé d'ajouter une couche CRF au modèle Transformer pour tenter d'améliorer ses performances.

Pour ce faire, il suffit d'apporter une petite modification au code du Transformer d'origine. Nous avons juste besoin d'ajouter une couche CRF supplémentaire à l'intérieur de Transformer.

```
class Transformer_CRF(nn.Module):
    def __init__(self, size_in, num_heads, num_encoder_layers,
    ↪  size_out):
        super().__init__()
        self.positional_encoding = PositionalEncoding(size_in)
        self.transformer_encoder =
        ↪  nn.TransformerEncoder(nn.TransformerEncoderLayer(size_in,
        ↪  num_heads), num_encoder_layers)
        self.linear = nn.Linear(size_in, size_out)
        self.crf = CRF(size_out, batch_first=True)

    def forward(self, x, tags=None):
        x = self.positional_encoding(x)
        x = self.transformer_encoder(x)
        x = self.linear(x)

        if tags is not None:
            mask = (tags != -100)
            tags = torch.where(tags == -100, torch.tensor(0,
            ↪  device=tags.device), tags)
            loss = -self.crf(x, tags, mask=mask)
            return loss
        else:
            return self.crf.decode(x)
```

Dans cette conversion, nous avons encore rencontré le même problème que dans le modèle BiLSTM_CRF, à savoir que la couche CRF ne reconnaît pas automatiquement -100 comme valeur de *padding*. Par conséquent, nous avons adopté la même approche pour résoudre le problème.

3.3.3.4 Apprentissage par transfert

Nous avons également essayé d'utiliser des modèles pré-entraînés de la communauté open source pour le traitement de texte, en utilisant une petite quantité de données annotées pour affiner les grands modèles et les adapter à cette tâche.

Nous avons choisi SikuBERT comme modèle à utiliser, et avons mis en œuvre le *fine-tuning* en utilisant la bibliothèque `transformers` de Hugging Face.

Tout d'abord, nous avons utilisé le même code d'extraction de données d'entraînement que précédemment. Grâce à ce code, nous avons obtenu deux listes : `texts` et `tags`. Le premier contient 800 textes, le second contient 800 listes avec des étiquettes correspondant à chaque texte.

Ensuite, nous chargeons le modèle pré-entraîné et créons un encodage numérique pour chaque étiquette dans les données d'entraînement.

```
tokenizer = AutoTokenizer.from_pretrained("SIKU-BERT/sikuroberta")
```

Ensuite, nous définissons une fonction pour remplir le texte et les étiquettes, comme nous l'avons fait précédemment dans le modèle Transformer. Le modèle pré-entraîné utilise également la structure du Transformer, donc nous devons toujours normaliser la longueur des textes.

Ce qu'il faut particulièrement noter, c'est que le *tokenizer* de la bibliothèque `transformers` dispose d'un paramètre intégré appelé `is_split_into_words`. Étant donné que notre segmentation de données est effectuée au niveau des caractères, ce qui signifie que chaque caractère est un *token*, dans ce cas nous devons définir `is_split_into_words` sur `False`.

Ensuite, nous définirons le modèle et les paramètres d'entraînement. Nous expliquerons deux des paramètres importants, le premier étant `num_train_epochs`, qui correspond au nombre de cycles d'entraînement du modèle. Comme le modèle pré-entraîné a déjà été suffisamment entraîné, nous n'avons pas besoin de fixer un nombre trop élevé de cycles pour la phase de *fine-tuning*, sinon il y aura un risque de surapprentissage. Le paramètre `per_device_train_batch_size` détermine la taille des lots de données d'entraînement. Ici, nous adoptons la même stratégie que celle utilisée lors de l'entraînement précédent du réseau neuronal en fixant la taille initiale à 1.

```
model =
↪  AutoModelForTokenClassification.from_pretrained("SIKU-BERT/sikuroberta
↪  ", num_labels=len(unique_tags))

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=2,
    per_device_train_batch_size=1,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    evaluation_strategy="steps",
    eval_steps=200,
)
```

Enfin, nous pouvons commencer à affiner le modèle.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
trainer.train()
```

Nous pouvons constater que la bibliothèque `transformers` fournie par Hugging Face nous offre une manière d'entraînement plus simple. Avec cette bibliothèque, nous n'avons même pas besoin de définir manuellement le modèle.

3.4 Création de corpus TEI

Une fois le texte traité, il ne reste plus qu'à générer un *template* de corpus et à le remplir avec le texte pertinent. Comme nous l'avons déjà fait pour la description de la chaîne d'encodage, nous commencerons par le *Shuowen Jiezi* puis nous présenterons le traitement du *Yiwen Leiju* afin d'expliquer le processus de traitement.

3.4.1 Shuowen Jiezi

Pour commencer, nous devons traiter les résultats de prédiction du modèle. Lors du traitement de cette œuvre, nous avons choisi le modèle CRF. Le modèle CRF prend les dictionnaires de caractéristiques en entrée et produit un résultat de même longueur que le nombre des dictionnaires. pour cette entrée « 祦：衬、祦，祖也。从示危聲。», le modèle retournera le résultat suivant :

```
[['B-Head', 'I-Head', 'B-Definition', 'I-Definition',
↪  'I-Definition', 'I-Definition', 'I-Definition', 'I-Definition',
↪  'I-Definition', 'B-Etymology', 'I-Etymology', 'B-Pronunciation',
↪  'I-Pronunciation', 'I-Pronunciation']]
```

Ces résultats ne peuvent pas être directement utilisés pour la création de corpus, donc nous avons construit la fonction suivante pour le post-traitement des résultats de prédiction :

```
def crf_convert_prediction(text) :
    dict_final = {}
    count = -1
    text_reuse = ""
    prev_tag = ''
    for i in crf_prediction(text)[0] :
        count += 1
        current_tag = i[2:]
        if current_tag == prev_tag :
            text_reuse += text[count]
        else :
```

```
            if len(text_reuse) != 0 :
                dict_final.update({text_reuse : prev_tag})
            text_reuse = text[count]
        prev_tag = current_tag

    dict_final.update({text_reuse : prev_tag})

    return dict_final
```

Cette fonction segmente le texte d'origine en fonction de la distribution des étiquettes dans les résultats de prédiction, et renvoie finalement un dictionnaire dont les clés sont les textes segmentés et les valeurs sont leurs types. Pour l'exemple précédent, la valeur de retour de cette fonction serait :

```
{'祧: ': 'Head', '祧、祧，祖也。': 'Definition', '从示': 'Etymology', '危
↪  聲。': 'Pronunciation'}
```

Ensuite, nous avons commencé à construire l'ensemble du *pipeline*. L'idée de ce *pipeline* est de lire des fichiers texte bruts, puis à utiliser des expressions régulières pour déterminer le type de texte. Pour les entrées, nous faisons appel au modèle pour effectuer une classification et générer des données TEI en fonction des résultats fournis par le modèle. Nous avons créé deux expressions régulières, une pour faire correspondre les numéros de volume et une autre pour faire correspondre les sections. À chaque itération d'une entrée par l'itérateur, le code utilise ces deux expressions régulières pour vérifier si elles correspondent ou non. En cas de correspondance, cela signifie que nous traitons un nouveau numéro de volume ou une nouvelle section et le code enregistre alors le fichier actuel du numéro de volume ou de la section tout en réinitialisant certaines variables. Nous utilisons la bibliothèque lxml [10] (version 4.9.2) pour toutes les opérations TEI. Cette bibliothèque offre de nombreuses méthodes qui nous permettent de générer facilement des structures TEI complexes.

```
current_volume = 0
re_volume = re.compile(r'卷[一二三四五六七八九十 ]')
re_section = re.compile(r'\b. 部\b')
clef = ""

with open('shuowen.txt','r',encoding='utf-8') as text:
    for line in text.readlines():
        predictions = crf_prediction(line.strip())

        if re.match(re_volume,line):
            current_volume += 1
            current_section = 0
            continue

```

10. URL : https ://lxml.de/ (accès le 1er août 2023).

```
        if re.match(re_section,line):
            current_section += 1
            current_entry = 0
            clef = line.rstrip()
            NSMAP = {None: 'http://www.tei-c.org/ns/1.0'}
            tei = etree.Element("TEI", nsmap=NSMAP)
            create_tei_header(tei)  # create teiHeader
            text = etree.SubElement(tei, "text")
            body = etree.SubElement(text, "body")

            continue
```

Dans le code ci-dessus, chaque fois que nous détectons un nouveau volume, la variable `current_volume` est incrémentée de 1. Cela signifie que l'itérateur entre dans un nouveau volume et en même temps, la variable `current_section` est réinitialisée à zéro car les numéros de section de chaque volume sont indépendants. Lorsque nous détectons une nouvelle section, en plus d'incrémenter la variable `current_section`, nous appelons également la fonction `create_tei_header`. Cette fonction génère les métadonnées (`teiHeader`) du nouveau fichier TEI ainsi que l'élément `<text>` pour contenir les entrées.

```
def create_tei_header(tei):
    teiHeader = etree.SubElement(tei, "teiHeader")
    fileDesc = etree.SubElement(teiHeader, "fileDesc")
    titleStmt = etree.SubElement(fileDesc, "titleStmt")

    title1 = etree.SubElement(titleStmt, "title")
    title1.set("type", "main")
    title1.set("{http://www.w3.org/XML/1998/namespace}lang", "lzh")
    title1.text = convert_to_chinese(current_volume,
    ↪  current_section,clef)
    ...
    title2 = etree.SubElement(titleStmt, "title")
    title2.set("type", "alt")
    title2.set("{http://www.w3.org/XML/1998/namespace}lang", "en")
    title2.text = f"TOn glyphs and characters: Scroll
    ↪  {current_volume}, Radical {current_section} (An electronic
    ↪  version)"
    ...
    respStmt1 = etree.SubElement(titleStmt, "respStmt")
    resp1 = etree.SubElement(respStmt1, "resp")
    resp1.text = "Author"
    persName1 = etree.SubElement(respStmt1, "persName")
    ...
    monogr = etree.SubElement(biblStruct, "monogr")
    title_monogr = etree.SubElement(monogr, "title")
    title_monogr.set("level", "s")
```

```
    title_monogr.set("{http://www.w3.org/XML/1998/namespace}lang",
    ↪  "lzh")
    title_monogr.text = " 說文解字 "
    ...
```

Lors de la création des métadonnées, la plupart des informations (telles que la description du projet, le responsable, etc.) n'ont pas besoin d'être modifiées. Ce que nous devons modifier, ce sont les titres de chaque section. Cela est plus complexe en chinois car nous devons conserver à la fois le titre en chinois et le titre en anglais, et cela implique également une conversion entre les chiffres arabes et les chiffres chinois. C'est pourquoi nous avons intégré une autre fonction, `convert_to_chinese`, dans les fonctions ci-dessus pour convertir les chiffres arabes en chiffres chinois. Au lieu de créer un dictionnaire des chiffres arabes et chinois, nous avons utilisé une méthode de conversion qui combine les caractéristiques morphologiques du chinois, ce qui la rend plus concise. Voici comment cela se présente :

```
def convert_to_chinese(current_volume, current_section, clef):
    chinese_dict = {
        0: " 零 ", 1: " 一 ", 2: " 二 ", 3: " 三 ", 4: " 四 ",
        5: " 五 ", 6: " 六 ", 7: " 七 ", 8: " 八 ", 9: " 九 "
    }

    if current_section < 10:
        chinese_number = chinese_dict[current_section]
    elif current_section == 10:
        chinese_number = '十'
    elif current_section < 20:
        chinese_number = '十' + chinese_dict[current_section % 10]
    elif current_section < 100:
        if current_section % 10 == 0:
            chinese_number = chinese_dict[current_section // 10] +
            ↪  '十'
        else:
            chinese_number = chinese_dict[current_section // 10] +
            ↪  '十' + chinese_dict[current_section % 10]

    title = f" 說文解字卷{current_volume}之第{chinese_number}部首:
    ↪  {clef}（電子版）"
    return title
```

En combinant tous les codes ci-dessus de cette section, nous pouvons finalement générer un en-tête comme le suivant :

```
<teiHeader>
    <fileDesc>
      <titleStmt>
        <title type="main" xml:lang="lzh">說文解字卷三之第二十三部首：晨
        ↪  部（電子版）</title>
```

```
        <title type="alt" xml:lang="en">TOn glyphs and characters: Scroll 3, Radical 23 (An electronic version)</title>
        <title type="short">TBD</title>
        <respStmt>
          <resp>Author</resp>
          <persName/>
        </respStmt>
        <respStmt>
          <resp>Original Compiler and Editor</resp>
          <persName corresp="#XS"/>
        </respStmt>
        <respStmt>
          <resp>Project Leader and Main Researcher</resp>
          <persName corresp="#MBL"/>
        </respStmt>
      </titleStmt>
      <publicationStmt>
        <publisher>Wikisource</publisher>
        <availability>
          <licence target="https://creativecommons.org/licenses/by-sa/3.0/">
            <p xml:lang="en">The original text is from Wikisource.</p>
          </licence>
        </availability>
      </publicationStmt>
      <sourceDesc>
        <listBibl>
          <head>Reference Work</head>
          <biblStruct>
            <analytic>
              <title level="a" xml:lang="lzh">說文解字</title>
              <respStmt>
                <resp>Main Editor</resp>
                <persName corresp="#TBD"/>
              </respStmt>
            </analytic>
            <monogr>
              <title level="s" xml:lang="lzh">說文解字</title>
              <edition>XXX</edition>
              <imprint>
                <publisher>XXX</publisher>
                <pubPlace>XXX</pubPlace>
                <date when="XXX"/>
                <biblScope unit="page" from="X" to="X"/>
              </imprint>
            </monogr>
          </biblStruct>
```

```
        </listBibl>
      </sourceDesc>
    </fileDesc>
    <revisionDesc>
      <change when="2023" who="#KZ">
        <p>First version. </p>
      </change>
    </revisionDesc>
  </teiHeader>
```

Ensuite, nous construisons des entrées. Pour chaque entrée, le code crée un élément <entry> et y ajoute un identifiant qui contient le numéro du volume ainsi que les informations sur la section choisie. Le code parcourt ensuite les résultats de classification donnés par les modèles d'apprentissage automatique. Le texte dans ces résultats conserve l'ordre des entrées de mots d'origine, donc nous n'avons pas besoin de traiter d'informations supplémentaires sur l'ordre. Pour chaque segment de texte, le code génère différents sous-éléments <entry> en fonction de son type.

```
current_entry += 1
    entry = etree.SubElement(body, "entry")
    entry.set('{http://www.w3.org/XML/1998/namespace}id',
    ↪  f'swjz_j{current_volume:02}_b{current_section:02}_e{current_entry:02}')

    # Processing predictions
    form = etree.SubElement(entry, "form")
    last_element = None

    for key, value in predictions.items():
        if value == 'Head':
            if key[-1] == ': ': # If last character is a colon
                orth = etree.SubElement(form, "orth",
                ↪  type="standard")
                orth.text = key[:-1] # Exclude last character
                orth.tail = key[-1] # Add the colon as a tail to the
                ↪  orth element
            else:
                orth = etree.SubElement(form, "orth",
                ↪  type="standard")
                orth.text = key
        elif value in ['Definition', 'Title', 'Citation']:
            if last_element is not None and last_element.tag ==
            ↪  'sense':
                sense = last_element
            else:
                sense = etree.SubElement(entry, "sense")
            if value == 'Definition':
                def_ = etree.SubElement(sense, "def")
```

```
                def_.text = key
            elif value == 'Title':
                cit = etree.SubElement(sense, "cit", type="classic",
                ↪  corresp="#ckpw-TBD")
                bibl = etree.SubElement(cit, "bibl")
                title = etree.SubElement(bibl, "title")
                title.text = key
            elif value == 'Citation':
                quote = etree.SubElement(cit, "quote")
                quote.text = key
            last_element = sense
        elif value in ['Etymology', 'Prononciation']:
            if last_element is not None and last_element.tag ==
            ↪  'etym':
                etym = last_element
            else:
                etym = etree.SubElement(entry, "etym")
            if value == 'Etymology':
                oRef = etree.SubElement(etym, "oRef")
                oRef.text = key
            elif value == 'Prononciation':
                pRef = etree.SubElement(etym, "pRef")
                pRef.text = key
            last_element = etym
```

Dans le code ci-dessus, nous utilisons la variable `current_entry` qui est réinitialisée chaque fois qu'une nouvelle section est détectée. Sa logique est similaire à celle de la variable `current_section` et sert à générer un id. Le code parcourt les résultats de prédiction du modèle d'apprentissage automatique que nous avons converti (un dictionnaire avec des clés correspondant au texte original et des valeurs correspondant aux classes prédites par le modèle). Ensuite, en fonction de la classe du texte, différentes balises TEI sont générées. Pour cela, nous avons créé une branche if pour chaque classe. De plus, lors de la génération des entrées, l'un des principaux défis consiste à prendre en compte l'agrégation des balises TEI, telles que les éléments `<oRef>` et `<pRef>`, qui sont tous deux des sous-éléments de `<etym>`. Ainsi, lorsque ces deux éléments apparaissent consécutivement, le code doit les placer dans le même élément `<etym>`. Pour cela, nous utilisons un pointeur `last_element` dans notre code. Ce pointeur fait référence à la dernière balise générée et nous permet de vérifier si la balise actuelle correspond à celle du pointeur afin de décider s'il faut les agréger, comme indiqué ci-dessous :

```
<text>
    <body>
      <entry xml:id="swjz_j03_b23_e02">
        <form><orth type="standard">麎</orth>: </form>
        <sense>
          <def>耕也</def>。
```

```
        </sense>
        <etym>
          <oRef>从晨</oRef>
          <pRef>囪聲</pRef>。
        </etym>
      </entry>
    </body>
  </text>
```

3.4.2 Yiwen Leiju

Dans cette œuvre, en raison de la présence d'un grand nombre d'entités nommées, nous avons choisi d'utiliser des modèles pré-entraînés pour les prédictions, car ce type de modèle a souvent un avantage dans la reconnaissance des entités nommées, surtout lorsque les données d'entraînement sont limitées.

Pour le *Yiwen Leiju*, nous utilisons une méthode différente à celle du *Shuowen Jiezi*. La raison de le faire est que lorsqu'il s'agit de traiter le *Yiwen Leiju*, un autre défi majeur réside dans le fait que cette œuvre contient de nombreux commentaires qui peuvent apparaître n'importe où dans le texte source. Lorsqu'un commentaire se trouve au milieu d'un paragraphe, cela signifie que nous devons créer une balise TEI à l'intérieur d'une autre balise TEI et y insérer le texte. Cette opération ne peut pas être réalisée avec du code car elle est considérée comme non sécurisée par les outils XML, donc aucune bibliothèque XML ne propose cette fonctionnalité. Par conséquent, dans cette œuvre, nous avons abandonné l'approche de réalisation en une seule fois et nous sommes passés à une approche en plusieurs étapes. Le processus de travail dans cette œuvre est le suivant :
Le code lit le texte ligne par ligne et pour chaque ligne de texte, nous utilisons d'abord le code pour extraire la partie commentaire et enregistrer sa position de début. L'objectif de cette étape est de séparer les commentaires des autres contenus. Les commentaires et leurs positions seront ensuite stockés dans un tuple.

```
Commentary = namedtuple('Commentary', ['start', 'text'])

def extract_and_remove_commentaries(line):
    re_matches = list(re.finditer(' 〈.*?〉 ', line))
    start_position = []
    # Adjust start positions
    dynamic_reduce = 0
    for match in re_matches:
        start_position.append(match.start() - dynamic_reduce)
        dynamic_reduce += len(match.group(0))

    commentaries = [Commentary(start, match.group(0)[1:-1]) for
    ↪  start, match in zip(start_position, re_matches)]
    text_no_commentaries = re.sub(' 〈.*?〉 ', '', line)
```

```
14
15      return commentaries, text_no_commentaries
```

Ensuite, nous utilisons des expressions régulières pour séparer la partie métadonnées (la partie que le modèle doit traiter) et la partie citation (qui n'a pas besoin d'être traitée par le modèle) du texte.

Ensuite, nous utilisons le modèle RoBERTa pour traiter la partie métadonnées ; les données traitées seront un dictionnaire Python avec les clés correspondant au texte original et les valeurs correspondant à leurs types.

Nous fusionnons à nouveau la partie métadonnées traitée avec la partie citation ; comme la partie métadonnées est déjà un dictionnaire, nous effectuons également la même opération sur la partie citation : chaque section de citation sera transformé dans une paire clé-valeur où la clé est le texte et la valeur est l'étiquette « QUOTE ».

Ensuite, nous insérons les commentaires que nous avions précédemment sauvegardés dans ce dictionnaire ; comme nous avons sauvegardé sa position de début lors de sa séparation initiale, cette opération n'est pas difficile à réaliser ; il faut noter que c'est aussi pendant cette opération où l'on ajouter des balises TEI pour les commentaires, autrement dit, les commentaires seront insérées dans le dictionnaire avec les balises TEI, Cela signifie que pour la partie des commentaires, ses balises sont directement attachées sous forme de texte brut. L'avantage de le faire est d'éviter les limitations de la bibliothèque XML.

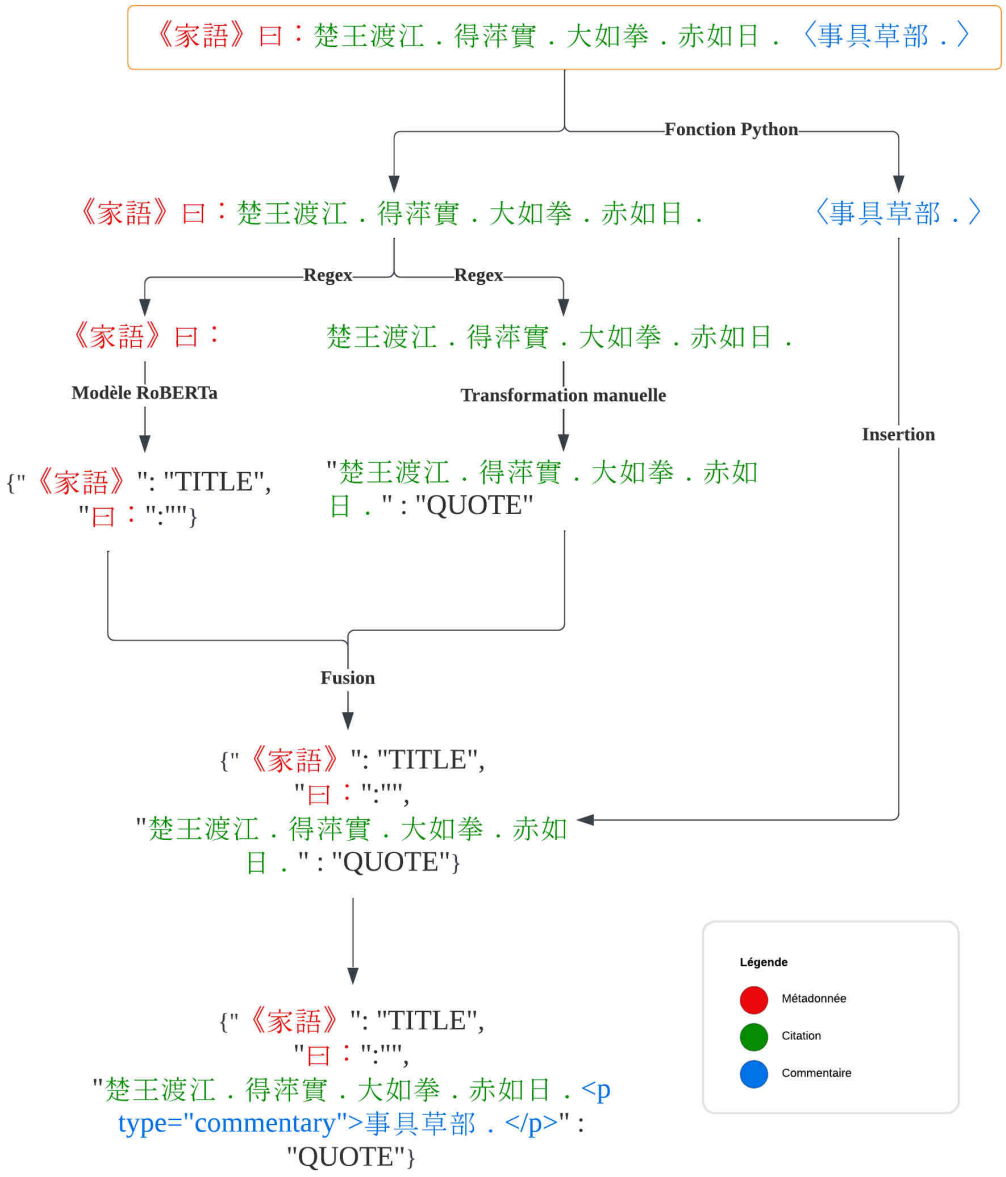

Figure 12 – Chaîne de traîtement pour le *Yiwen Leiju*

La figure ci-dessus montre notre processus de traitement, qui peut être résumé comme suit : diviser les différentes parties, puis les faire passer par différents traitements avant de les recombiner. Nous sauvegardons ensuite le dictionnaire Python final dans un fichier JSON, car ce type de fichier convient très bien à cette structure de données complexe.

Une fois que toutes les données ont été converties et sauvegardées au format JSON, nous lisons à nouveau le fichier JSON pour générer le corpus TEI ; cette étape est similaire à celle du *Shuowen Jiezi*, où nous générons des balises TEI différentes en fonction des balises rencontrées.

3.5 Relecture

La relecture est la dernière étape de la chaîne de traitement. À cette étape, nous vérifions manuellement si le corpus généré contient des erreurs. Étant donné que le corpus contient de nombreuse balises TEI, sa lisibilité n'est pas bonne. Nous extrayons d'abord le texte à l'aide du code, puis utilisons displaCy [11] pour le visualiser.

Figure 13 – Résultat rendu par displaCy

Lorsque nous extrayons du texte à l'aide de code, nous enregistrons également la position absolue de chaque caractère dans le texte d'origine ainsi que la balise TEI associée. Ensuite, displaCy peut créer une visualisation hautement lisible en fonction de ces informations.

En même temps, nous extrayons également le `xml:id` correspondant à chaque élément. Dans le cas où une erreur est détectée dans un élément, cet identifiant permet une localisation rapide pour une modification manuelle.

11. URL : https ://spacy.io/usage/visualizers (accès le 1er août 2023).

4 Résultats et discussion

La démarche entreprise dans ce mémoire pour l'encodage TEI comporte les six étapes suivantes : acquisition et nettoyage du texte, analyse de la structure du texte, modélisation TEI des corpus basée sur l'analyse de la structure, annotation automatique de séquences pour obtenir les étiquettes TEI, création des objets XML-TEI et relecture. Ici nous présentons les résultats de l'annotation automatique car c'est celle qui présente le plus de défis et pour laquelle on peut envisage différentes méthodes. Une analyse des résultats est également fournie. Nous commencerons par discuter de la performance des différents types de modèles dans le *Shuowen Jiezi*, puis nous aborderons leur performance dans le *Yiwen Leiju*.

4.1 Shuowen Jiezi

Lors de l'annotation de séquences, nous avons d'abord essayé une méthode basée sur des règles. Étant donné que près de 70% des entrées dans le *Shuowen Jiezi* ont la même structure, cette méthode basée sur des règles a produit des résultats acceptables. Pour améliorer davantage la qualité du traitement automatisé, nous avons essayé trois méthodes d'apprentissage automatique différentes : les méthodes traditionnelles (CRF), les réseaux neuronaux (BiLSTM, Transformer et leurs versions optimisées) ainsi que les modèles pré-entraînés.

Pour comparer les performances entre ces différentes approches, nous avons effectué une évaluation des performances. Au cours de l'évaluation, nous avons utilisé le score F1 pour évaluer les performances. Le score F1 est une mesure d'évaluation très courante en apprentissage automatique, sa définition est :

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

La *precision* calcule la proportion des instances réellement appartenant à la classe *c* parmi toutes les instances jugées appartenir à la classe *c* par le modèle. Le *recall* calcule la proportion des instances réellement appartenant à la classe *c* qui ont été correctement identifiées par le modèle, le score F1 combine les résultats des deux mesures. Étant donné que certaines données sont en quantité insuffisante dans nos données annotées (comme la partie de citation), le score F1 peut bien équilibrer la situation où la distribution des données est inégale. En outre, pour rendre les résultats de l'évaluation plus fiables, nous avons utilisé une validation croisée lors de l'évaluation. Pour chaque modèle, nous avons utilisé une validation croisée à 5 plis et obtenu finalement 5 scores F1.

	1	2	3	4	5	Average
Rule Based	0.9198	0.9357	0.9151	0.9211	0.9234	0.9231
CRF	0.9702	0.9762	0.9756	0.9783	0.9712	0.9743
BiLSTM	0.9037	0.9161	0.9118	0.8955	0.9202	0.9095
BiLSTM + CRF	0.9327	0.9482	0.9242	0.9187	0.9419	0.9331
Transformer	0.8452	0.8203	0.8059	0.9244	0.9061	0.8682
Transformer + CRF	0.9343	0.9121	0.9145	0.8941	0.9265	0.9163
RoBERTa	0.9509	0.9508	0.9392	0.9070	0.9562	0.9408

Table 1 – Les scores F1 de toutes les méthodes (le *Shuowen Jiezi*)

Tout d'abord, ce qui est surprenant, c'est que la méthode basée sur des règles a une performance très bonne dans les données de validation, voire comparables à la plupart des réseaux neuronaux. La raison en est probablement que la plupart des entrées dans le texte ont de structures simples, donc cette méthode est particulièrement adaptée.

Ensuite, en ce qui concerne le modèle CRF, après avoir ajouté la validation croisée, nous avons constaté que sa performance globale était stable. Étant donné que le processus de sélection de nos données d'entraînement et de validation a limité le nombre de structures principales (très régulières), cette performance peut être considérée comme représentative. La raison de sa bonne performance pourrait être que les textes dans ce projet sont relativement courts et que la structure de la plupart des textes est assez simple, le modèle CRF étant capable d'apprendre efficacement les relations de dépendance entre les différentes parties du texte. Nous pouvons en conclure que le modèle CRF est extrêmement performant dans la tâche d'annotation de séquences de textes hautement structurés de ce projet, et qu'il ne nécessite qu'une petite quantité de données (environ 9 % des entrées) pour atteindre des performances qui peuvent être appliquées à des tâches réelles.

Dans la méthode du réseau neuronal, nous avons constaté que l'ajout d'une couche CRF sur le réseau neuronal de base peut effectivement améliorer ses performances. Cependant, ce qui nous a surpris, c'est que le modèle BiLSTM semble être supérieur au modèle Transformer. Nous supposons que cela est dû à l'absence de dépendances à longue distance dans le texte de notre projet, et donc les avantages du Transformer n'ont pas été exploités. De plus, selon nos analyses précédentes, il apparaît que dans cette œuvre, la sémantique n'est pas importante ; ce qui compte vraiment, ce sont les distances relatives entre les mots-outils, la ponctuation et le texte. Par conséquent, il est possible que les réseaux neuronaux aient plus de difficultés à remarquer cette relation implicite lorsqu'ils utilisent des *word embeddings*. Dans le cas du CRF, nous informons directement l'algorithme de cette relation en tant que caractéristique explicite, ce qui permet au CRF d'avoir une meilleure performance.

Pour le modèles pré-entraînés, leur précision en matière de reconnaissance d'entités nommées (titres de livres et noms de personnes) est très élevée. Cela peut être dû à l'apprentissage d'un grand nombre d'expressions d'entités nommées grâce à ses données d'entraînement massives.

La performance excellente de la méthode basée sur des règles nous a rendus curieux quant à sa performance dans les textes complexes. Pour cela, nous avons choisi au ha-

sard 50 entrées du *Shuowen Jiezi*. Dans notre processus de sélection, nous avons utilisé une expression régulière pour exclure le type d'entrée le plus courant et avons défini un seuil de longueur nettement supérieur à la longueur moyenne des entrées. Seules les entrées dont la longueur dépasse cette valeur sont considérées comme des candidats. Ensuite nous les avons annotées avec Label Studio, puis nous avons évalué ces données avec la méthode basée sur des règles. Les résultats de cette méthode sont les suivants :

	precision	recall	f1	support
Citation	0.86	0.60	0.71	30
Definition	0.69	0.78	0.73	124
Etymology	0.83	0.58	0.69	77
Head	1.00	1.00	1.00	50
Pronunciation	0.75	0.65	0.70	23
Title	0.91	0.71	0.80	28
micro avg	0.80	0.74	0.77	332
macro avg	0.84	0.72	0.77	332
weighted avg	0.81	0.74	0.76	332

Table 2 – La performance de la méthode basée sur les règles sur les entrées complexes

Il n'est pas difficile de constater que lorsque la complexité du texte augmente, les performances des méthodes basées sur les règles diminuent considérablement. Ensuite, nous avons utilisé les mêmes données pour tester le modèle CRF.

	precision	recall	f1-score	support
Citation	0.92	0.80	0.86	30
Definition	0.83	0.93	0.88	124
Etymology	0.92	0.77	0.84	77
Head	1.00	1.00	1.00	50
Pronunciation	0.86	0.83	0.84	24
Title	1.00	0.93	0.96	28
micro avg	0.90	0.88	0.89	332
macro avg	0.92	0.87	0.90	332
weighted avg	0.90	0.88	0.89	332

Table 3 – La performance de CRF sur les entrées complexes

Les résultats ont montré que bien que les performances du modèle CRF aient également diminué avec l'augmentation de la complexité du texte, sa baisse était beaucoup moins importante que celle des méthodes basées sur des règles.

4.2 Yiwen Leiju

	1	2	3	4	5	Average
CRF	0.7061	0.7576	0.8342	0.7866	0.6914	0.7552
BiLSTM	0.9274	0.8952	0.9203	0.8974	0.8938	0.9068
BiLSTM + CRF	0.9251	0.9120	0.9392	0.8966	0.9223	0,9190
Transformer	0.8967	0.9202	0.9187	0.9195	0.9188	0.9147
Transformer + CRF	0.9178	0.8878	0.9265	0.9010	0.9140	0.9094
RoBERTa	0.9110	0.8865	0.9215	0.9215	0.9167	0.9114

Table 4 – Les scores F1 de toutes les méthodes (le *Yiwen Leiju*)

Dans le *Yiwen Leiju*, nous n'avons pas utilisé de méthode basée sur des règles car, bien que les mots-outils soient présents dans les textes que nous traitons, ils ne sont pas utiles pour établir des règles d'annotation. Nous avons donc uniquement utilisé des méthodes d'apprentissage automatique. En examinant le tableau ci-dessus, il est facile de constater que pour le *Yiwen Leiju*, les performances du modèle CRF a diminué par rapport au *Shuowen Jiezi*. Cela peut être dû à une réduction de l'influence des mots-outils. En plus de cela, les performances des réseaux neuronaux sont assez stables. Nous pensons que cela peut être dû à l'utilisation des *word embeddings*, ce qui fait qu'ils ne dépendent pas trop des mots-outils pour l'apprentissage.

Après avoir mené des essais sur ces algorithmes différents, nous pouvons maintenant conclure que le modèle CRF a obtenu les meilleures performances lorsqu'il est confronté à des scénarios où les mots-outils jouent un rôle important. Nous encodons explicitement la dépendance entre les mots-outils et le texte sous forme de caractéristiques, ce qui permet au modèle d'apprendre efficacement cette relation. Nous avons constaté que très peu de données d'entraînement (moins de 9% du texte original) étaient nécessaires pour entraîner un modèle capable d'être appliqué dans des scénarios du monde réel.

En plus de cela, les modèles de réseaux neuronaux ont également obtenu d'excellentes performances. Cependant, étant donné que nous ne pouvons pas contrôler quelles caractéristiques le réseau neuronal doit se concentrer, les performances du réseau neuronal sont relativement inférieures à celles du CRF dans le cas où les mots-outils sont largement utilisés.

5 Synthèse, limites et perspectives

Dans cette étude, l'acquisition des données a été réalisée à partir de sources en ligne. Nous avons collecté un vaste ensemble de textes en chinois classique. Bien que cette étape ait été relativement simple, il est important de souligner que la qualité des données et les défis liés à la numérisation ont été pris en compte lors de la sélection de la source. Le plus grand problème que nous avons rencontré à cette étape est en fait le droit d'utilisation du texte. Certains sites web ne fournissent pas gratuitement leur corpus (Sinica), et bien que certains permettent aux utilisateurs d'utiliser gratuitement leur texte, ils limitent la modification et la republication du texte (Chinese Text Project), ce qui nous oblige à nous tourner vers Wikisource qui utilise entièrement une licence de partage open source. Suivant la licence de Wikisource, notre corpus dans le projet sera également entièrement ouvert au public (sur Nakala), ce qui signifie qu'il peut être modifié et republié. De plus, tout le code sera également rendu public sur GitLab.

Une attention particulière a été accordée à la conception de la structure du corpus. Pendant cette étape, nous pensons que la structure du corpus doit être aussi proche que possible de celle du texte original. Par exemple, dans le *Shuowen Jiezi*, la structure originale est une structure en trois niveaux : volume - section - entrée. Par conséquent, lors de la conception du corpus, nous avons choisi d'utiliser les sections comme unité de base pour construire une telle hiérarchie : œuvre - volume - section. Chaque niveau correspond à un conteneur dans la structure originale du texte, c'est-à-dire que le niveau des œuvres sert de conteneur pour le niveau des volumes dans le texte original ; Le niveau des volumes sert de conteneur pour le niveau des sections dans l'original et celui-ci sert lui-même comme conteneur pour toutes les entrées lexicales présentes dans l'original. En outre, en raison de la présence de plusieurs corpus dans notre projet, nous avons conçu plusieurs normes pour faciliter leur intégration uniforme dans le système informatique. Tout d'abord, nous avons établi que chaque unité fondamentale de chaque corpus (dans le *Shuowen Jiezi* c'est l'entrée) doit comporter un identifiant unique. Cela vise non seulement à garantir la sécurité des données, mais aussi à permettre aux chercheurs de localiser rapidement les corpus lorsqu'ils les consultent. Deuxièmement, en raison de la présence de nombreuses citations dans ce projet, nous souhaitons pouvoir facilement rechercher chaque œuvre ou personnage cité dans l'ensemble des corpus, ce qui facilite les études sur le chinois classique basées sur les œuvres ou les personnages (qu'il s'agisse d'une perspective synchronique ou diachronique). Pour chaque personne ou œuvre citée, nous utilisons un identifiant unique et nous constituons une base de connaissances en documentant ces personnes et œuvres dans une base ouverte.

La tâche de TAL pertinente pour l'encodage TEI automatique tel que conçu dans ce projet est l'annotation de séquences, c'est-à-dire le balisage des différentes parties dans le même texte. Pour cela, nous avons conçu plusieurs solutions, y compris des méthodes basées sur des règles et des méthodes d'apprentissage automatique, parmi lesquelles nous avons comparé plusieurs types différents de modèles. En termes de résultats, le modèle CRF a donné la meilleure performance lorsqu'on traite le *Shuowen Jiezi*. Cela peut être dû au fait que nous traitons les entrées comme unités de base, que la longueur des entrées est courte et qu'il existe une distance courte entre les parties dépendantes. Ainsi, le CRF peut modéliser la relation de dépendance avec peu de données. En dehors de cela, les performances des réseaux neuronaux sont très similaires. Bien que les per-

formances dans le *Shuowen Jiezi* ne soient pas aussi remarquables que celles du modèle CRF, elles restent stables entre différentes œuvres et ne présentent pas de fluctuations importantes de performance comme le modèle CRF.

Enfin, nous avons utilisé un code pour générer automatiquement le corpus au format XML-TEI à partir des textes traités et des étiquettes prédites pour chaque séquence de caractères dans le texte. Ce code a été développé en prenant en compte les spécificités de notre structure de corpus. Pour le *Shuowen Jiezi*, nous utilisons un processus dynamique et à usage unique, en utilisant des itérateurs pour parcourir les entrées de texte d'origine. Pour chaque entrée, le code appelle un modèle d'apprentissage automatique pour la traiter, puis le résultat renvoyé par le modèle est converti en un dictionnaire Python par une fonction de post-traitement. Dans ce dictionnaire, nous stockons le texte segmenté et les étiquettes correspondantes sous forme de paires clé-valeur. Ensuite, le code lit le dictionnaire dans l'ordre et génère des balises TEI correspondantes en fonction des étiquettes du texte avant de remplir le contenu textuel. Pour le *Yiwen Leiju*, étant donné qu'il contient de nombreux commentaires qui peuvent apparaître n'importe où, il est difficile pour nous d'utiliser la même méthode que celle utilisée dans le *Shuowen Jiezi* pour générer le corpus TEI, car nous ne pouvons pas insérer de nouveaux éléments à n'importe quel endroit dans les éléments déjà générés en utilisant la bibliothèque XML. Par conséquent, nous utilisons un fichier JSON comme intermédiaire : premièrement, nous traitons le texte à l'aide du modèle combiné avec les fonctions Python et l'enregistrons dans un fichier JSON ; ensuite, nous lisons ce fichier JSON pour générer les corpus TEI.

Pour conclure, au cours de notre recherche, nous avons réussi à construire un corpus en chinois classique, en surmontant de nombreux défis. Ce corpus, structuré de manière à respecter au mieux l'organisation originale des textes, représente une contribution significative à l'étude du chinois classique.

Grâce à sa structure spécifiquement conçue et aux normes mises en place pour l'intégration uniforme des textes, notre corpus offre un nouvel outil précieux pour l'étude synchronique ou diachronique du chinois classique.

Cependant, notre recherche présente certaines limitations. Tout d'abord, bien que Unicode nous offre des avantages pour notre projet, cette norme ne prend toujours pas en charge tous les caractères du chinois classique. Cela signifie que dans les dictionnaires comme le *Shuowen Jiezi*, certaines autres formes d'écriture de certains caractères ne peuvent pas être encodées.

Dans le futur, nous souhaitons développer davantage notre corpus en intégrant des textes de différentes sources et en améliorant nos modèles grâce à un ensemble de données d'entraînement plus large. Et selon notre comparaison des modèles d'apprentissage automatique, nous pensons qu'il serait préférable de combiner le CRF avec un modèle pré-entraîné basé sur le Transformer pour travailler sur l'encodage numérique du chinois classique. Les avantages des deux résident dans leur capacité à fournir d'excellentes performances sans nécessiter de grandes quantités de données d'entraînement, en particulier le modèle CRF. Étant donné que ce dernier ne se concentre pas sur la sémantique, il est donc facile pour nous d'étendre les données d'entraînement du modèle en remplaçant simplement les caractères.

En conclusion, malgré les défis rencontrés, notre recherche a abouti à la création d'un outil potentiellement précieux pour l'étude du chinois classique. Nous espérons que notre travail stimulera de nouvelles recherches dans ce domaine et contribuera à une meilleure compréhension du chinois classique.

Références

Bottéro, F. (1996). *Sémantisme et classification dans l'écriture chinoise, les systèmes de classements des caractères par clés du Shuowen jiezi au Kangxi zidian* (I. d. h. é. c. Collège de France, Éd. ; T. 37). Récupérée 24 août 2023, à partir de https://hal.science/hal-02402824

Chen, K.-J., Huang, C.-R., Chang, L.-P., & Hsu, H.-L. (1996). SINICA CORPUS : Design Methodology for Balanced Corpora. *Proceedings of the 11th Pacific Asia Conference on Language, Information and Computation*, 167-176. https://doi.org/http://hdl.handle.net/2065/12025

Denecke, E. b. W., Li, W.-Y., & Tian,
bibinitperiod X. (Éd.). (2020, novembre 1). *The Oxford Handbook of Classical Chinese Literature : (1000BCE-900CE)*. Oxford University Press.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171-4186. https://doi.org/10.18653/v1/N19-1423

Ezen-Can, A. (2020, septembre 11). A Comparison of LSTM and BERT for Small Corpus. Récupérée 26 août 2023, à partir de http://arxiv.org/abs/2009.05451

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation*, *9*, 1735-80. https://doi.org/10.1162/neco.1997.9.8.1735

Lafferty, J. D., McCallum, A., & Pereira, F. C. N. (2001). Conditional Random Fields : Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the Eighteenth International Conference on Machine Learning*, 282-289.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural Architectures for Named Entity Recognition. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, 260-270. https://doi.org/10.18653/v1/N16-1030

Ma, X., & Hovy, E. (2016). End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, 1064-1074. https://doi.org/10.18653/v1/P16-1101

Martin, L., Muller, B., Ortiz Suárez, P. J., Dupont, Y., Romary, L., de la Clergerie, É., Seddah, D., & Sagot, B. (2020). CamemBERT : a Tasty French Language Model. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7203-7219. https://doi.org/10.18653/v1/2020.acl-main.645

Pagel, J., Sihag, N., & Reiter, N. (2021). Predicting structural elements in german drama. *Proceedings of the Second Conference on Computational Humanities Research*.

Pytlik Zillig, B. L. (2009). TEI Analytics : converting documents into a TEI format for cross-collection text analysis. *Literary and Linguistic Computing*, *24*(2), 187-192. https://doi.org/10.1093/llc/fqp005

Radford, A., & Narasimhan, K. (2018). Improving Language Understanding by Generative Pre-Training. Récupérée 26 août 2023, à partir de https://www.semanticscholar.org / paper / Improving - Language - Understanding - by - Generative - Radford - Narasimhan/cd18800a0fe0b668a1cc19f2ec95b5003d0a5035

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, *21*(1), 140 :5485-140 :5551.

Rennie, S. (2000). Encoding a historical dictionary with the TEI. Institut für Maschinelle Sprachverarbeitung. Récupérée 26 août 2023, à partir de https://eprints.gla.ac.uk/72093/

Ruiz Fabo, P., Bernhard, D., Briand, A., & Werner, C. (2023). Computational drama analysis from almost zero electronic text : The case of Alsatian theater. *Computational Drama Analysis : Reflecting Methods and Interpretations*. Récupérée 14 mai 2023, à partir de https://univoak.eu/islandora/object/islandora%3A157880/

Sturgeon, D. (2019). Chinese text project : A dynamic digital library of premodern chinese [Publisher : Oxford University Press]. *Digital Scholarship in the Humanities*, *36*. https://doi.org/10.1093/llc/fqz046

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, décembre 5). Attention Is All You Need. https://doi.org/10.48550/arXiv.1706.03762

Wang, D., Liu, C., Zhu, Z., Feng, J., Hu, H., Shen, S., & Li, B. (2022). Construction and Application of Pre-training Model of “Siku Quanshu” Oriented to Digital Humanities. *Tushuguan luntan* 图书馆论坛, *42*(6), 31-43.

Annexe

Tout le code utilisé est publié sur GitLab : https ://gitlab.huma-num.fr/chi-know-po/all-about-plants

Les corpus sont publiés sur Nakala (collections « CHI-KNOW-PO », « Shuowen Jiezi (full text) » et « Yiwen Leiju (full text) »)

Toutes les librairies d'apprentissage automatique utilisées dans ce projet sont les suivantes :

PyTorch (version 2.0.1) sklearn-crfsuite (version 0.3.6) TorchCRF (version 1.2.0) Transformers (version 4.32.1)