Prédiction de la difficulté de compréhension des contenus audiovisuels

Justine Revol

Prédiction de la difficulté de compréhension des contenus audiovisuels

REVOL

Justine

Sous la direction de Isabelle Ferrané et Tim Van De Cruys

Tuteur universitaire : Olivier Kraif

Réalisé au sein de l'IRIT

Mémoire de master 2 professionnel - Mention Sciences du Langage - 20 crédits

Parcours : Industrie de la Langue

Année universitaire 2019-2020

Prédiction de la difficulté de compréhension des contenus audiovisuels

REVOL

Justine

Sous la direction de Isabelle Ferrané et Tim Van De Cruys

Tuteur universitaire : Olivier Kraif

Réalisé au sein de l'IRIT

Mémoire de master 2 professionnel - Mention Sciences du Langage - 20 crédits

Parcours : Industrie de la Langue

Année universitaire 2019-2020

Remerciements

Je tiens à remercier mes tuteurs Isabelle et Tim qui ont été très présents et ont su m'aider malgré un début de stage difficile en contexte de confinement et dans une autre ville. Ils ont toujours su m'expliquer les choses et m'aider quand il le fallait. Je les remercie aussi de m'avoir fait confiance pour ce stage et d'avoir adapté le sujet pour qu'il me corresponde au mieux. Je remercie également M. Kraif d'avoir été mon tuteur universitaire et d'avoir répondu à mes questions lorsque j'en avais besoin.

Je voudrais aussi remercier mes parents et mon frère pour leur soutien. Mes amis et camarades de classe : Marie, Lucie, Myriam, Rachel, Florent, Vincent et Oussama pour leur écoute et leurs conseils ainsi que pour nos visioconférences de confinement. Je tiens aussi à remercier Constance pour son aide, son soutien et sa bonne humeur pendant tous ces mois de stage.
Je tiens aussi tout particulièrement à remercier mes colocataires Fiona et Kaïs qui m'ont permis de vivre mon aventure toulousaine en toute sérénité.

Merci également à l'ensemble de l'équipe pédagogique du master qui sait répondre présente, et nous encourager malgré nos difficultés.

Résumé

L’apprentissage d’une langue est un exercice qui demande beaucoup d’efforts de la part des apprenants. Les enseignants de langue sont équipés de plusieurs supports et outils pour les aider. Un des supports pédagogiques le plus utilisé est la vidéo avec des extraits de films. Cependant, cet exercice demande aux enseignants de connaître le niveau de difficulté de chaque extrait utilisé.

Pendant mon stage j’ai donc travaillé à l’élaboration d’un système permettant d'annoter automatiquement le niveau de difficulté de grandes quantité de données qui servirait ensuite à développer un système permettant aux enseignants de trouver automatiquement des extraits de films d’un niveau adapté à celui de leurs élèves. Pour cela, j’ai d’abord travaillé à la définition de règles syntaxiques à partir d’ouvrages dédiés à l’apprentissage du français langue étrangère et basés sur les niveaux du Cadre européen commun de référence pour les langues (CECRL). Ensuite, pour pouvoir renvoyer des étiquettes de niveaux de difficulté selon des critères définis, j’ai implémenté ces règles en fonction de labellisation avec un outil de *data programming*. Snorkel. Un ensemble de règles lexicales à part la suite été ajouté avec l’utilisation d’un corpus de fréquences de mots du français par niveau de langue (FLELEX).

Mots Clés : compréhension, prédiction, CECRL (Cadre européen commun de référence pour les langues), étiquettes de niveau, data programming, fréquences de mot, règles syntaxiques et règles lexicales.

Abstract

Learning a language is an exercise that requires a lot of effort on the part of the learners. Language teachers are equipped with several supports and tools to help them. One of the most widely used teaching aids is video with excerpts from films. However, this exercise requires teachers to know the level of difficulty of each extract used.

During my internship, I worked on developing a system to automatically annotate the level of difficulty of large amounts of data which would then be used to develop a system that would allow teachers to automatically find movies extracts of the same level as that of their students. For this, I first worked on the definition of syntactic rules from books dedicated to the learning of French as a foreign language and based on the levels of the Common European Framework of Reference for Languages (CEFRL). Then, in order to be able to return labels of levels of difficulty according to defined criteria, I implemented these rules according to labeling with a data programming tool, Snorkel. A set of lexical rules apart from the rest was added with the use of a corpus of French word frequencies by language level (FLELEX).

Keywords: comprehension, prediction, CEFR (Common European Framework of Reference for Languages), level labels, data programming, word frequencies, syntactic rules and lexical rules.

Sommaire

Remerciements....5
Résumé....7
Abstract....8
Introduction....11
PARTIE 1....12
Chapitre 1 : Contexte du stage....13
Chapitre 2 : Comment mesurer la difficulté de compréhension ?....15
PARTIE 2....18
Chapitre 3 : Paradigme du *Data programming*....19
Chapitre 3.1 : L'outil Snorkel....20
Chapitre 4 : Ressources utilisées....21
4.1 Corpus : OpenSubtitles....21
4.2 Étiquettes du CECRL....23
4.3 Bibliothèque python : SpaCy....23
4.4 Règles syntaxiques....24
4.5 Règle lexicale....25
PARTIE 3....27
Chapitre 5 : Mise en œuvre....28
5.1 Osirim....28
5.2 Implémentation des règles....28
5.3 Erreurs de SpaCy....31
Chapitre 6 : Résultats et analyse....32
6.1 Analyse des premiers résultats de l'entraînement....32
6.1.1 Explication des résultats fournis par Snorkel....32
6.1.2 Analyse des résultats avec les règles syntaxiques....33
6.1.2 Analyse des résultats avec les règles syntaxiques et lexicales....36
6.2 Analyse des résultats avec le corpus test....37
Chapitre 7 : Améliorations possibles....39
PARTIE 4....41
Chapitre 8 : Bilan....42
8.1 Travail réalisé....42
8.2 Bilan personnel....43
Bibliographie....44
Sitographie....46
Annexes....50

Introduction

Ce mémoire présente le déroulement et les résultats de mon stage de fin d'études réalisé au sein de l'Institut de Recherche en Informatique de Toulouse (IRIT) du 16 mars au 11 septembre 2020. L'IRIT est une unité mixte de recherche (UMR) qui regroupe 24 équipes réparties dans 7 départements. Pendant mon stage, j'ai été sous la tutelle de Isabelle Ferrané de l'équipe SAMoVA qui est dans le département « Signaux et Images » et de Tim Van De Cruys de l'équipe MELODI du département d'« Intelligence Artificielle ». Pour cause de pandémie mondiale ce stage s'est déroulé dans des conditions inédites, puisque le confinement et les règles en vigueur ont imposé que je réalise ce stage en télétravail.

Ce stage traite de la *prédiction de la difficulté de compréhension des contenus audiovisuels*. Il s'agit de réaliser un outil permettant de prédire le niveau de difficulté de compréhension que peut représenter un extrait de film pour un apprenant de langue étrangères. Cette étude a été menée sur la base des sous-titres disponibles dans la langue cible (ici le français). Plus précisément, l'objectif de ce stage est de construire un système capable de renvoyer des étiquettes de niveaux de difficulté en fonction d'un texte donné.

Dans notre rapport nous expliquerons d'abord plus précisément le contexte dans lequel s'inscrit le stage, ensuite nous aborderons le travail bibliographique qui a dû être réalisé en amont pour définir par quel prisme aborder le sujet. Dans une deuxième partie, nous expliquerons tous les outils et les ressources que nous avons utilisées pour créer notre système basé sur un ensemble de règles syntaxiques et lexicales. Et nous finirons par expliquer comment nous avons implémenté les règles identifiées puis discuterons de nos premiers résultats.

PARTIE 1

-

CONTEXTUALISATION DU SUJET

Chapitre 1 : Contexte du stage

Dans le cadre de l'apprentissage des langues, les enseignants de Français Langue Étrangère (ou FLE) utilisent fréquemment les films comme outils pédagogiques. Chaque enseignant utilise un film ou une séquence de film qu'il juge intéressant pour l'apprentissage de ses élèves, cependant la plupart utilisent souvent les mêmes extraits. Dans ce contexte le travail réalisé en stage s'inscrit dans un projet plus global qui vise à proposer un système qui permette d'aider les enseignants à trouver de manière automatique des séquences qui soient en accord avec le niveau de leurs élèves. C'est dans cette optique que Estelle Randria, doctorante à l'IRIT, a décidé d'inscrire sa thèse qui porte sur l'évaluation de la compréhensibilité dans les interactions cinématographiques. En effet, ce sont souvent les interactions dans les films qui servent de base à l'apprentissage.

Mon stage s'inscrit dans la même thématique que celle d'Estelle Randria, mais il s'agit de se focaliser sur une approche différente. En effet, celui-ci consiste à développer un système qui permette d'annoter automatiquement de grandes quantités de données qui pourraient être disponibles ensuite pour développer le système permettant aux enseignants de trouver facilement et de manière automatique des extraits de films qui soient adaptés au niveau de leurs élèves pour servir de support d'apprentissage. Une différence existe cependant entre la thèse et mon stage sur les modalités abordées. Pour sa thèse, Estelle se sert de différentes modalités (visuelle, sonore et textuelle), alors que pour ce stage nous nous intéressons uniquement aux sous-titres de films, et donc à l'aspect textuel.

Avant cela, pour disposer d'un système automatique qui puisse déterminer le niveau de difficulté, il faudrait disposer d'un corpus annoté. Or un tel corpus n'est pas disponible. Pour cela, nous allons utiliser une méthode d'apprentissage faiblement supervisée basée sur un corpus de données non annotées. Nous utilisons une approche appelée *data programming* avec l'outil Snorkel pour annoter nos données et leur associer des étiquettes ou labels avec plus ou moins d'erreurs. On parle dans ce cas de corpus faiblement annoté. Mon rôle dans ce stage est de créer des règles pour renvoyer des étiquettes selon des critères syntaxiques précis.

Notre système permettrait de fournir un corpus annoté mais bruité (certaines étiquettes étant potentiellement erronées), qui pourrait ensuite servir de base à l'apprentissage d'un système automatique permettant d'attribuer un niveau de difficulté à des données textuelles issues de sous-titres de films.

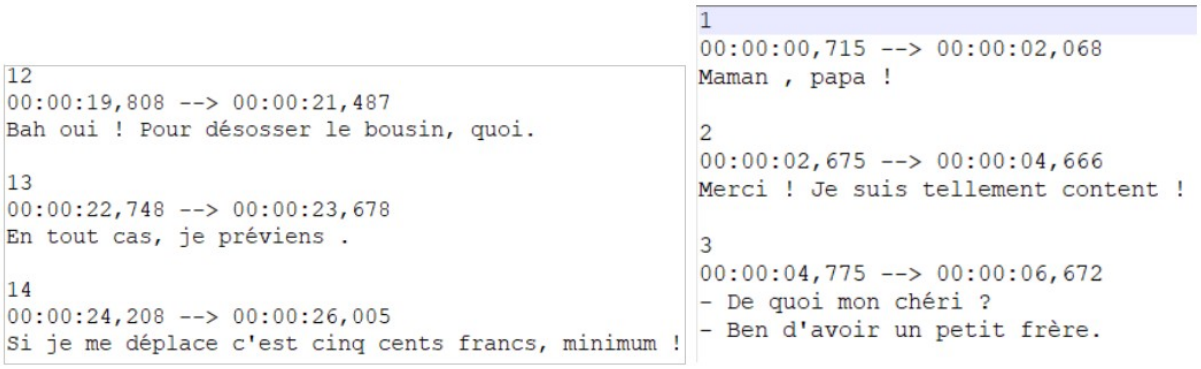


Figure 1: Exemples sur le film "Le petit Nicolas" issus du corpus de Randria et al. (2020)

Par exemple, le système doit être capable de dire lequel des deux extraits ci-dessus est potentiellement le plus difficile à comprendre d'un point de vue lexical et syntaxique (figure 1). En tant qu'humain nous savons que le plus difficile des deux est le premier, car il comporte des mots plus difficiles que le deuxième par exemple. Il va donc falloir entraîner notre système pour qu'il puisse reconnaître les difficultés que les apprenants du français langue étrangère peuvent rencontrer.

Avant cela, il est nécessaire de définir ce que nous entendons par niveau de difficulté et comment cela peut se mesurer, en étudiant la littérature sur le sujet.

Chapitre 2 : Comment mesurer la difficulté de compréhension ?

Afin de cerner les contours de notre sujet dans le but de développer notre système, il semble fondamental de comprendre ce que « prédiction de la difficulté de compréhension » signifie. Il est d'abord question de connaître ce que signifie « compréhension » ainsi que les mécanismes qu'elle implique, associé à cette notion de compréhension nous rencontrons la notion de « lisibilité ». Nous cherchons à savoir quelle mesure permettrait de déterminer si un texte est potentiellement facile ou difficile à comprendre. Les travaux existants ont surtout porté sur la compréhension dans le cadre de la lecture de textes.

Elola et Roubaud (2018) présentent dans leur article deux tests pour mesurer la lisibilité textuelle. Le premier est le test de closure (Taylor, 1953). Il s'agit de faire remplir un texte à trou aux différents sujets. Selon le remplissage effectué par ces sujets, la lisibilité est mesurée. Le deuxième est un test de calcul ou « formule de lisibilité » (Lively et Pressey, 1923). Comme l'explique Elola et Roubaud, il s'agit pour la première formule de calculer la longueur des mots, la longueur de la phrase, le nombre de syllabes. Ensuite, d'autres paramètres ont été ajoutés, comme la fréquence des mots (Lively et Pressey, 1923) ou encore le nombre de mots difficiles présents dans un texte (Gray et Leary, 1935). Dale et Chall (1948) présentent une formule pour prédire la lisibilité. Ils expliquent que la fréquence d'apparition d'un mot donne une indication du niveau de difficulté du texte. Un mot qui va revenir régulièrement dans un texte (donc ayant une haute fréquence), va être considéré comme facilement compréhensible par le lecteur. Ils fournissent également une liste de mots peu (ou non) connus avec leur formule de lisibilité. Cette liste permet de renseigner sur les mots pouvant complexifier le vocabulaire du texte. Il est également mentionné que la longueur des phrases à un impact sur la lisibilité, au-delà de 7 mots, une phrase deviendrait plus difficile à comprendre.

Conquet et Richaudeau (1973) présentent dans leur livre la « formule de lisibilité la plus connue» qui est celle de Flesch. Ils expliquent comment la formule fonctionne, tout d'abord elle se divise en deux partie : la « facilité de lecture » et l'« intérêt humain ». La première se calcule sur des échantillons de 100 mots avec le nombre de syllabes par mots et la longueur moyenne des phrases par mots. La deuxième consiste à calculer sur un même échantillon de 100 mots, le pourcentage de mots personnels (pronoms, mots communs) et le pourcentage de phrases personnelles adressées directement au lecteur.

Toutes ces formules ont été construites sur la langue anglaise. Ce n'est que plus tard, que des linguistes tels que Kandel et Moles (1958) adaptent au français la formule du test de lisibilité de Flesch-Kincaid. Néanmoins, Elola et Roubaud (2018) expliquent, en citant Labasse (2015), que l'utilisation d'une formule de lisibilité seule ne suffit pas à mesurer exactement le niveau puisque le

lecteur passe par plusieurs étapes pour comprendre un texte : "" *la reconnaissance des signaux graphiques* "*(niveau de la mise en forme graphique du texte),* "*le traitement lexical et syntaxique*" *(niveau de la lisibilité),* "*l'établissement des relations entre les éléments d'information*" *(niveau sémantique) et* "*la construction d'un modèle mental de l'état du monde que présente le texte*" (niveau de la compréhension)". Dans leur article, les deux linguistes vont travailler sur le traitement lexical mais aussi syntaxique.

Dans leur article, Randria et al. (2020) citent Henry (1975) qui a créé sa propre formule de closure pour le français (il s'agit d'une formule utilisant un texte à trous identique à celle de (Taylor, 1953)). En plus du nombre de mots par phrases et des mots difficiles, il ajoute dans sa formule le pourcentage de pronoms personnels et le nombre d'indicateurs de dialogue (point d'exclamation, guillemets, noms utilisés seuls).

L'utilisation de la fréquence des mots revient dans de nombreuses études et articles, cependant, les linguistes la calculent différemment. Adelmanetal et al. (2006) proposent de compter la diversité contextuelle (le nombre de contextes dans lesquels un mot apparaît). Ils estiment que mesurer la diversité contextuelle donne de meilleurs résultats quant à l'importance de la fréquence des mots, car elle permet de connaître la place du mot dans le lexique mental du locuteur. Chen et al. (2016) proposent de compléter la fréquence moyenne par l'écart type pour donner de meilleurs résultats. Ils expliquent également que les modèles basés sur la loi de Zipf, c'est-à-dire qui portent sur la fréquence des mots dans un texte, offrent une meilleure performance que ceux formés sur la diversité culturelle.

Collins-Thompson et Callan (2006) expliquent que la plupart des mesures de lisibilité prennent en compte seulement deux facteurs : la familiarité des unités sémantiques utilisées (mots ou phrases) et la complexité syntaxique de la structure de la phrase. Pour le premier, il s'agit tout simplement de la fréquence des mots, le nombre de syllabes par mot ou encore leur longueur. Pour le deuxième facteur, il s'agit uniquement de la longueur des phrases. Ils proposent dans leur article, d'utiliser des modèles de langues statistiques pour la difficulté de compréhension de lecture des documents web et autres textes non traditionnels. Leur modèle statistique nommé « Smoothed Unigram model » se base sur la classification naïve bayésienne. Pour chacun des 12 niveaux scolaires américains, Collins-Thompson et Callan créent un modèle de langue de la manière suivante : la difficulté sémantique d'un texte T est prédite par rapport au niveau scolaire en calculant la probabilité que le mots du texte T soient générés par un modèle de langage de grade (basé sur ce qui est appris à chaque niveau). Les unigrammes, définis par une liste de mots et leur probabilités individuelles, sont utilisés pour calculer la probabilité de chaque mot. Pour tester leur modèle, ils ont comparer le RMS (Root-mean-square) de leur modèle à celui d'un modèle traditionnel. Les résultats montrent alors que leur modèle a obtenu de meilleur résultat sur des passages de moins de 100 mots. De plus, l'erreur du RMS est réduite de 30 à 50 % avec leur modèle.

Cette étude bibliographique a permis de cerner les contours du sujet. Par la présentation de travaux précédemment effectuée, nous avons pris connaissance des points forts de ces études mais aussi de leurs limites. En effet, la plupart de ces études s'intéressent principalement au lexique ou à la statistique en calculant la fréquence des mots par exemple. Cependant, peu s'intéressent ou intègrent la syntaxe dans leur critère de lisibilité. Dans notre projet nous avons décidé d'intégrer, en plus du lexique et de la fréquence, des paramètres syntaxiques pour l'élaboration de notre outil d'annotation faiblement supervisée.

PARTIE 2

-

MÉTHODE : APPROCHE FAIBLEMENT SUPERVISÉE

Chapitre 3 : Paradigme du *Data programming*

Comme nous l'avons mentionné dans le contexte de notre stage, nous disposons d'un corpus non annoté (cf. chapitre 4.1). Même si un corpus peut être annoté de manière manuelle, cela demande beaucoup de temps et surtout des moyens humains particuliers, car il faudrait des professionnels de l'apprentissage des langues qui soient capables de donner des étiquettes de niveaux pour chaque extrait. Nous allons donc annoter nos données de manière automatique grâce au principe du *data programming*.

Le concept de *data programming* introduit par Ratner et al. (2016) consiste à créer un corpus d'apprentissage (données annotées) à partir d'un ensemble de règles d'annotations qui sont définies par l'utilisateur-expert. Ces règles d'annotation représentent des connaissances ou des heuristiques sur le domaine considéré. Elles permettent d'annoter automatiquement les données via des fonctions de labellisations. Dans notre cas, les fonctions de labellisation se basent sur des règles syntaxiques et lexicales que je vais développer à partir d'ouvrage de FLE et du lexique FleLex (cf. chapitre 4.5).

Le data programming va nous permettre de gérer les conflits entre les fonctions de labellisation. Ceux-ci surviennent quand deux fonctions de labellisation annotent un sous-ensemble de données de manière différentes. Prenons l'exemple des deux phrases suivantes :

> *Je mange une pomme*

> *Le parlementaire propose une motion*

Si nous nous intéressons à la syntaxe et au lexique de chacune de ces phrases, nous voyons que la première phrase comporte une syntaxe simple de type sujet-verbe-complément et un lexique simple. Cela signifie que pour cette phrase le système va renvoyer la même étiquette. La deuxième phrase quant à elle possède une syntaxe simple également mais un lexique qu'on peut considérer comme plus difficile avec des mots comme « parlementaire » et « motion ». Pour cette phrase, le système va alors renvoyer deux étiquettes différentes pour une même phrase, il va donc y avoir un conflit. Ainsi, dans une approche d'annotation manuelle l'utilisateur va devoir choisir laquelle des deux étiquettes utiliser ou alors choisir de fusionner les deux, mais cela peut entraîner une perte de précision. Le data programming permet de faire cela automatiquement en apprenant un modèle en prenant en compte les deux étiquettes renvoyées par le système.

Chapitre 3.1 : L'outil Snorkel

Snorkel est un outil de *data programming* créé par Ratner et al (2017) pour permettre à des utilisateurs de pouvoir construire facilement des données d'entraînements. Il a été bâti selon trois critères. Le premier est de pouvoir utiliser toutes les ressources disponibles, c'est-à-dire de permettre à l'utilisateur d'avoir des étiquettes venant de toutes les ressources faiblement supervisées à sa disposition. La deuxième est de créer des données d'entraînement comme interface pour l'apprentissage automatique. Le système doit être capable de modéliser les sources d'étiquette pour induire un seul étiquetage probabiliste pour chaque instance. Ces étiquettes sont utilisées pour entraîner un modèle discriminatif, qui va permettre de généraliser. Enfin le dernier critère est celui de l'efficacité, le système doit être capable de fournir rapidement des résultats avec un taux de fiabilité qui soit comparable à une annotation qui serait réalisée par des humains.

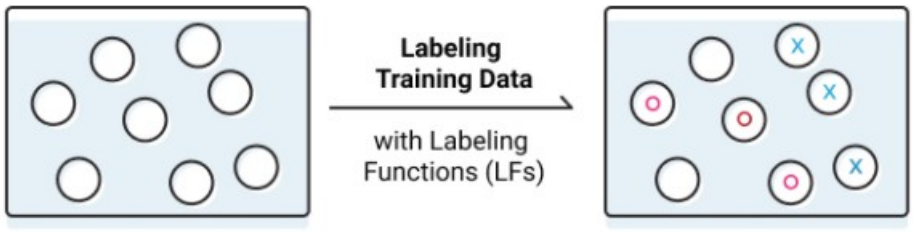


Figure 2: Fonction de labellisation de Snorkel

La figure 2 résume ce que permet de faire Snorkel, c'est-à-dire attribuer des étiquettes à des données non annotées que nous avons présentées en entrée.

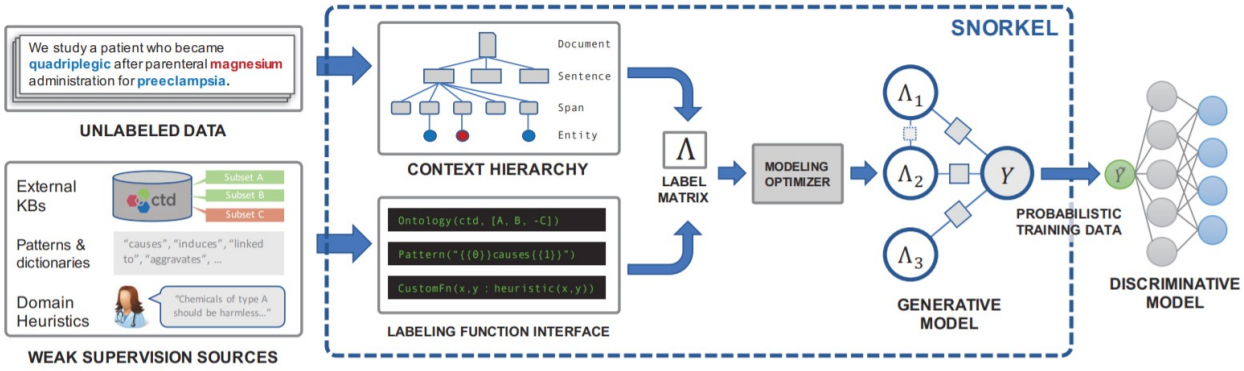


Figure 3: Schéma du fonctionnement de Snorkel (Ratner et al., 2017)

Dans la figure 3, nous pouvons voir ce que Snorkel permet de faire en tant qu'outil de *data programming*. Tout à gauche de la figure, « unlabeled data » et « weak supervision sources » sont ce que l'utilisateur va apporter au système, c'est-à-dire les sources et les règles qui vont être utilisées comme fonction d'étiquetage. Ces dernières vont permettre à Snorkel, grâce à un optimiseur, de travailler et d'apprendre automatiquement un modèle génératif en estimant la précision et la corrélation des étiquettes si celles-ci sont peu nombreuses, ou faisant un vote à la

majorité si il y a un grand nombre d'étiquettes renvoyées, c'est-à-dire garder l'étiquette qui aura était renvoyée par le plus de règle. Pour cela, il va analyser les accords et les désaccords en fonction des étiquettes qui auront été renvoyées par les fonctions d'étiquetage pour un même sous-ensemble de données.

A la suite de la modélisation du modèle génératif, nous obtenons un ensemble d'étiquettes sélectionnées automatiquement à partir de leur probabilité. Ces étiquettes précises vont permettre de former des modèles d'apprentissage. Par exemple, il est possible de faire un modèle discriminatif à partir de nos étiquettes pour garder la même précision mais pouvoir généraliser nos fonctions d'étiquetages.

Chapitre 4 : Ressources utilisées

4.1 Corpus : OpenSubtitles

Pour constituer notre corpus de sous-titres d'extraits de films, nous avons choisi d'utiliser OpenSubtitles 2018 (Lison et Tiedemann, 2016). Cette base de données, construite à partir d'IMDb (Internet Movie Database), est composée de plus de trois millions de fichiers au format XML de sous-titres de films et séries dans 60 langues différentes. Nous avons sélectionné dans ce grand volume de données uniquement les films et séries en langue française puisqu'il s'agit de la langue cible de notre système.

Nous travaillons donc sur une partie seulement de ces films. Pour pouvoir les sélectionner nous disposons d'un fichier « **`OPUS metadata json.pickle`** » qui nous donne accès aux métadonnées des films et séries comprenant notamment la langue, ainsi qu'un identifiant unique par film. A partir de ce fichier et dans le cadre d'un stage précédent (Petiot, 2018), un découpage automatique (Figure 3) a été fait grâce à l'application d'une règle simple qui consiste à segmenter les sous-titres en séquences, dès que deux sous-titres consécutifs sont séparés de plus de 7 secondes de non parole. Ce délai a été défini empiriquement lors de travaux précédents. L'idée est de découper les sous-titres en séquences d'interaction différentes (dialogue entre mêmes participants sur un sujet donné dans un même cadre spatio-temporel) en faisant l'hypothèse qu'une zone de non parole relativement longue (> 7s) peut, dans la plupart des films, séparer deux scènes différentes. Cela a permis d'avoir un ensemble de fichiers de quelques lignes de dialogues.

Nous disposons pour notre corpus de plusieurs dossiers rangés par année de 1910 à 2017. Nous avons fait le choix de sélectionner des films et des séries sur une grande échelle de sortie de films pour avoir plus de données pour entraîner notre corpus. Il faut cependant noter que pour les films anciens, les extraits ne sont pas des dialogues, mais le texte inséré entre deux scènes de films

muets. Ces dossiers classés par année comprennent des dossiers correspondant aux identifiants des films ou séries, eux-mêmes décomposés en plusieurs fichiers de séquences correspondant à des dialogues. Le corpus d'OpenSubtitles2018 sert à entraîner nos règles (Tableau 1). Nous disposons également d'un corpus de test issu d'une étude de Randria et al. (2020) sur l'évaluation de la compréhension dans les interactions cinématographiques. Celui-ci est composé de 55 extraits issus de 12 films français différents dont la difficulté a été évaluée par des enseignants du Français Langue Étrangère (FLE) sur une échelle de 1 à 100. Dans la figure 4, nous voyons que les sous-titres du corpus de test sont formés différemment de ce que nous avons pour notre corpus d'entraînement. Pour pouvoir récupérer seulement les lignes de dialogue sans les *time codes*, nous avons dû utiliser la bibliothèque **`pysrt`** qui permet de passer les lignes qui ne correspondent pas au texte du sous-titre.

En plus du corpus de test (Tableau 1), nous disposons d'un second fichier comportant les notes des enseignants. Il y a trois types de notes pour chaque extrait de films : une note sur la difficulté syntaxique, une note sur la difficulté lexicale ainsi qu'une note globale sur la difficulté de l'extrait. Nous ne pouvons cependant pas donner d'exemple de ce fichier puisque la thèse est encore en cours.

```
1
00:00:09,314 --> 00:00:11,914
Une heure plus tard , au onze boulevard Saint-Martin ,...

2
00:00:12,014 --> 00:00:16,814
... Amélie entre dans un magasin de farces et attrapes, déguisements et cotillons.

3
00:00:16,814 --> 00:00:20,914
Au même instant , au cent huit de la rue Lecourbe un homme quitte son domicile .
```

Figure 4: Exemple du corpus test avec « Le fabuleux destin d'Amélie Poulain » (Randria et al., 2020)

Même si nous utilisons ce corpus pour notre test, il faut cependant noter qu'à la différence de notre système qui ne prend en compte qu'une seule modalité, celle de l'écrit, dans cette étude, les enseignants ont évalué la difficulté par rapport à d'autres modalités puisqu'ils disposaient de l'audio, de la vidéo et du texte. Ces autres modalités ont un impact sur la notation car certains enseignants ont jugé difficile ou facile tel ou tel extrait aussi en prenant en compte les bruits environnant lors d'un dialogue, le contexte ou encore l'accent des personnages. Cependant, ce corpus est celui qui nous semble le plus pertinent pour évaluer notre système, car nous pourrons confronter ce qu'il a trouvé à une évaluation humaine.

Tableau 1: Récapitulatif des données des deux corpus

	Corpus d'entraînement (OpenSubtitles 2018 (Lison et Tiedemann, 2016))	Corpus de test (Randria et al., 2020)
Nombre de films	66373	12
Nombre de séquences	81911	55
Nombre de phrases	1 064 843	900

4.2 Étiquettes du CECRL

Comme nous l'avons dit le principe du *data programming* et son outil Snorkel sont fondés sur l'annotation des données via la création de règles et la définition d'étiquettes. Pour notre système, nous avons besoin d'étiquettes qui reflètent un niveau de difficulté linguistique pouvant être assimilées à un niveau de difficulté de compréhension. Au début, nous avons pensé à choisir des étiquettes du type « facile-moyen-difficile » cependant comment délimiter ces étiquettes ? En effet, ces notions sont beaucoup trop subjectives pour être utilisées dans notre cadre théorique et dans un système que nous voulons utilisable par tous les enseignants de FLE. Nous avons pensé aux niveaux du Cadre Européen Commun de Référence pour les Langues (CECRL : A1, A2, B1, B2, C1 et C2). L'avantage d'utiliser ces niveaux était double. Dans un premier temps, il s'agissait d'utiliser des étiquettes qui soient familières pour les enseignants, car ce sont celles-ci qui représentent les niveaux dans l'enseignement des langues étrangères. Dans un second temps, ce choix a été motivé par l'aspect pratique de ces étiquettes. En effet, il est facile de trouver des ouvrages qui informent sur les éléments qui entrent en compte dans la définition de ces niveaux d'apprentissage selon les niveaux de difficulté. Nous avons limité la caractérisation du niveau de difficulté à des critères lexicaux et syntaxique. Pour obtenir ces informations, nous utilisons la ressource FleLex pour le lexique (cf. chapitre 4.5), et la bibliothèque python **`SpaCy`** pour le traitement des règles syntaxiques.

4.3 Bibliothèque python : SpaCy

`SpaCy` est une bibliothèque python très utilisée en traitement automatique des langues. Elle permet d'attribuer des vecteurs de mots, des catégories (POS ou part-of-speech), des dépendances, mais aussi de faire de la reconnaissance d'entités nommées dans plus de 50 langues différentes.

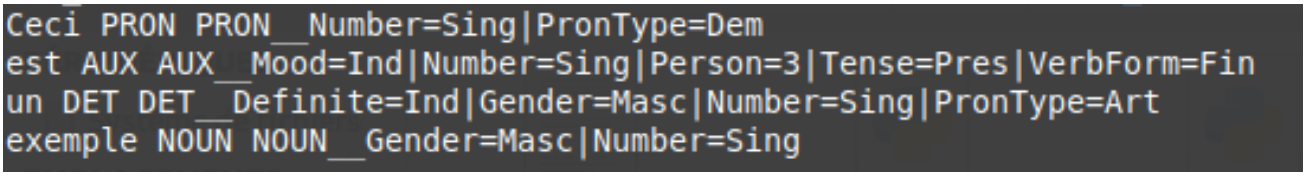


*Figure 5: Exemple de sortie SpaCy où "ceci" est la propriété token.text, "PRON" est la propriété token.pos_ et PRON_Number=Sing|PronType=Dem" est la propriété token.tag_*

Pour cela, **Spacy** possède en français trois modèles statistiques de base entraînés sur différentes données. Un modèle basique «fr_core_news_sm», et deux modèles plus avancés «fr_core_news_md» et «fr_core_news_lg»1. Les trois sont entraînés sur le UD French Sequoia et WikiNER mais à la différence du modèle basique, les deux plus avancés ont en plus les vecteurs de mots entraînés à l'aide de la bibliothèque FastText1 utilisant le modèle CBOW (continuous bag of words) sur les ressources Wikipedia et OSCAR (Common Crawl). Les vecteurs de mots vont permettre de faire des tests de similarité. Or pour notre système nous n'avons pas besoin de cette information (similarité des mots). De plus l'accuracy pour le «fr_core_news_sm»1 est de 98,52 pour les tokens et 94,20 pour les tags contre respectivement 98,52 et 95,72 pour le «fr_core_news_md», la différence étant minime nous avons choisi d'utiliser le modèle basique «fr_core_news_sm »2.

4.4 Règles syntaxiques

Pour rédiger nos règles qui sont l'élément central de notre système, nous nous sommes appuyés sur des ouvrages référentiels du Français Langue Étrangère (Beacco & al., 2004; Beacco & al., 2006; Beacco & al., 2008; Beacco & al., 2011; Riba, 2016). Nous disposions d'un livre par niveaux du CECRL, mais nous nous sommes principalement servi des niveaux A2, B1, et B2. En effet, les niveaux dit « avancés » c'est-à-dire C1 et C2 devenaient beaucoup trop précis et sémantiquement complexes, et donc allaient demander des règles beaucoup trop lourdes. Nous avons donc laissé la sélection des niveaux avancés au lexique dont nous parlerons plus tard. Ces ouvrages nous ont permis d'avoir des règles correctement délimitées selon les niveaux. Cependant, l'utilisation de ce genre de livres peut aussi poser un problème linguistique. La syntaxe de l'oral est différente de celle de l'écrit, or les sous-titres de films sont justement dans une modalité écrite retranscrivant ce qui est dit à l'oral (parfois en le simplifiant, mais nous ferons abstraction de cette question). Ainsi, nous avons pris en compte ce facteur dans la rédaction de nos règles, et certaines règles jugées trop littéraires ont été éliminées de notre script.

La définition d'un premier jeu de règles s'est opérée en considérant des familles de règles. En tout nous disposons de 60 règles comprenant 9 règles adjectivales, 3 règles nominales, 6 règles verbales, 8 règles passives, 13 règles pronominales (avec et sans négation) et 20 règles interrogatives ainsi qu'une règle sur la longueur des phrases. L'objectif n'était pas d'être exhaustif,

mais d'avoir un premier ensemble qui couvre un ensemble significatif de cas. Il y a différentes structures de règles qui sont chacune utilisées selon les informations fournies par **SpaCy**. Certaines vont chercher un pattern ou motif fixe de catégories syntaxiques (il s'agit de la plupart des règles), d'autres vont chercher une expression figée, ou encore d'autres vont chercher les informations dans le tag des catégories. Cependant, même si la structure des règles et les patterns de recherche peuvent changer selon les fonctions, ces dernières ont toutes une structure identique :

Tableau 2: Structure des règles syntaxiques

SI <CONDITION>
ALORS retourner ÉTIQUETTE
SINON retourner S'ABSTENIR

En effet, le principe du data programming consiste à définir des ensembles de règles basiques, plutôt que d'avoir des règles très complexes couvrant de nombreux cas.

4.5 Règle lexicale

Comme nous l'avons dit précédemment, en plus des livres, nous nous sommes servis d'un lexique pour définir les étiquettes de niveaux. Ces informations lexicales sont très importantes, car la difficulté de compréhension passe aussi par le vocabulaire du lecteur. Bien qu'il faille différencier lexique et vocabulaire, les deux niveaux d'informations sont importants. Le premier correspond au regroupement des mots d'une langue (représentant la structure de groupes de mots) tandis que le second représente une partie de ce que connaît le lecteur du lexique de la langue cible langue. Dans un contexte d'apprentissage d'une langue en milieu scolaire, il y a tout de même un socle de vocabulaire à connaître à chaque niveau.

Dans ce contexte, nous avons choisi d'utiliser FLELex CRF (François et al., 2014) qui est un lexique regroupant les fréquences normalisées de plus de 14 000 mots pour chaque niveau du CECRL. Il a été développé conjointement par Le Centre de traitement du langage naturel (CENTAL) de l'Université catholique de Louvain, le Laboratoire Parole et Langue de l'Université Aix-Marseille et la société EarlyTracks. Pour obtenir ce corpus, les chercheurs ont récupéré 2071 textes de différentes manuels scolaires datant d'après 2011. Chaque texte s'est vu attribuer le même niveau que celui de son manuel, puis a été transformé au format XML par des outils de reconnaissance optique de caractères. Après une correction manuelle, un corpus de 777 000 mots a été créé.

Après avoir utilisé deux taggers pour étiqueter les mots, la fréquence d'apparition des mots dans les différents niveaux a été calculée pour pouvoir créer deux lexiques différents basés sur les deux

taggers utilisés. Le premier lexique est FLELex TT qui a basé sa fréquence sur la tokenization et les part-of-speech de Treetagger. Ce lexique compte en tout 14 236 entrées. Le deuxième est FLELex CRF qui lui est basé sur les résultats du tagger CRF et qui compte 17 871 entrées. Nous avons choisi d'utiliser FLELex CRF tout d'abord car il contient un plus grand nombre d'entrées mais aussi car selon François et al (2014), celui-ci fournit une meilleure estimation des fréquences, même si certaines d'entre elles ne sont pas bien étiquetées.

Nous avons décidé de récupérer principalement les fréquences des niveaux B2, C1 et C2, car les deux derniers niveaux ne sont pas représentés dans les règles syntaxiques, ce qui donne ainsi une information complémentaire. Pour cela, nous disposons d'un fichier « **`fle_converted_B2.txt`** » créé à partir des fréquences qui se compose uniquement du lemme, du part-of-speech, et de son niveau de difficulté (figure 7).

```
abonné NOM B2
abord NOM C1
abordable ADJ C1
aboutissement NOM C2
```

Figure 6: Exemple fle_converted_B2.txt

Nous récupérons ensuite l'étiquette du premier niveau où l'on trouve une fréquence. Nous considérons alors que ce niveau est celui qui caractérise la difficulté du mot. Par exemple, pour le mot *évocateur* l'étiquette C2 est renvoyé car dans FLElex (voir figure 8) le mot apparaît au niveau C2, tandis que le niveau B2 est affecté à *évocation*.

word	tag	freq_a1	freq_a2	freq_b1	freq_b2	freq_c1	freq_c2	freq_total
évocateur	A	0.0	0.0	0.0	0.0	0.0	4.1173	0.0665
évocation	N	0.0	0.0	0.0	4.5416	0.0	0.0	0.1037

Figure 7: Exemple de fréquence pour les mots "évocateur" et "évocation" dans FLElex_CRF (François et al., 2014)

Dans cette partie, nous avons exposé les bases du principe du *data programming* ainsi que les ressources que nous avons utilisés pour notre système, nous allons voir maintenant comment nous avons implémenté cela via Snorkel, et analyser les premiers résultats obtenus.

PARTIE 3

-

IMPLÉMENTATION, RÉSULTATS ET ANALYSE

Chapitre 5 : Mise en œuvre

Le corpus de sous-titre dont nous disposons étant très important, il est nécessaire de lancer le traitement sur une machine capable de traiter ce volume de données en un temps raisonnable. Avant de décrire l'implémentation des règles, je vais donner une brève explication sur la plate-forme utilisée pour exécuter mon script.

5.1 Osirim

Osirim (Observatoire des Systèmes d'Indexation et de Recherche d'Information Multimédia) est une plate-forme de l'IRIT disposant d'environ 1 Po de stockage. Elle est composée de deux nœuds : le nœud interactif et le nœud de calcul. Le premier permet à l'utilisateur de se connecter, on peut y accéder grâce à des identifiants via **`ssh`** (Secure Shell) qui est une commande sécurisée de connexion à distance. Un fois connecté, il permet de lancer les calculs sur le second nœud en utilisant un job **`SLURM`** via la commande **`sbatch`**. Pour cela, il est possible de créer un fichier **`shell`** qui va lancer le job lié à l'exécution d'un script donné.

Le temps d'exécution du script peut être surveillé avec la commande **`squeue`**, cependant cette commande va montrer tous les jobs en cours, alors il est possible de surveiller uniquement son calcul avec la commande suivante : **`squeue|grep <nom_utilisateur>`**

L'utilisation de la plate-forme OSIRIM a été un avantage pour pouvoir gagner en performance, et permettre de lancer notre script sur un grand volume de données. C'est aussi sur cette plate-forme que nous avons récupéré notre corpus et toutes les ressources nécessaires pour notre projet.

5.2 Implémentation des règles

Notre système d'annotation faiblement supervisé dispose d'un premier jeu de 60 règles, chacune d'elle est classée dans des catégories différentes : verbales, adjectivales, pronominales, modales ou encore interrogatives.

L'écriture de ces règles s'est faite en plusieurs étapes. La première a été de sélectionner dans les ouvrages les différentes règles (structure syntaxique ou niveaux de l'apprentissage des temps et mode par exemple, cf. chapitre 4.4) selon plusieurs critères. Le premier concerne la complexité de la règle, il ne fallait pas choisir une règle trop complexe à écrire en code car même si en théorie

nous pouvons penser que cela va nous aider à être précis dans ce que nous renvoyons, en pratique il est possible qu'un mot ou une ponctuation vienne « polluer » la structure que nous recherchons et ainsi la rendre inutilisable. Prenons l'exemple des constructions pronominales, pour les niveaux les plus élevés dans certaines structures il pouvait y avoir plus de quatre pronoms dans une même phrase ce qui les rend intéressantes mais beaucoup trop complexes à décrire.

Le deuxième critère est la pertinence, par exemple, nous n'avons pas sélectionné la règle de subordination, c'est-à-dire renvoyer une étiquettes suivant le temps qui suit une subordonnée introduite par « que » ou « qu' », car nous donnons déjà une étiquette avec le mode et le temps verbal. Le dernier critère est les informations fournies par SpaCy qui vont nous aider à décider quelles règles sont réalisables.

Ce premier travail a permis de récupérer les règles qui semblaient les plus judicieuses pour notre système. Une fois cette sélection faite, la deuxième étape a consisté à écrire toutes ces règles en pseudo-code, c'est à ce moment là que le travail en amont sur les informations fournies par SpaCy a été utile. Par exemple, pour ce qui est des règles verbales, l'information du mode et du temps étaient données dans le **`token.tag`** c'est-à-dire les informations du part-of-speech (POS ou catégorie lexicale). Ainsi pour pouvoir renvoyer des étiquettes de niveaux, il fallait chercher dans le **`token.tag`** le mode et le temps du verbe. La troisième étape a été d'écrire les règles en code `python` sur le script Snorkel fourni préalablement par Tim Van De Cruys. Cette étape a été la plus longue, car il fallait gérer les erreurs de SpaCy (que nous aborderons plus tard), mais aussi la syntaxe de Snorkel dans la manière d'articuler les fonctions, et de renvoyer les étiquettes.

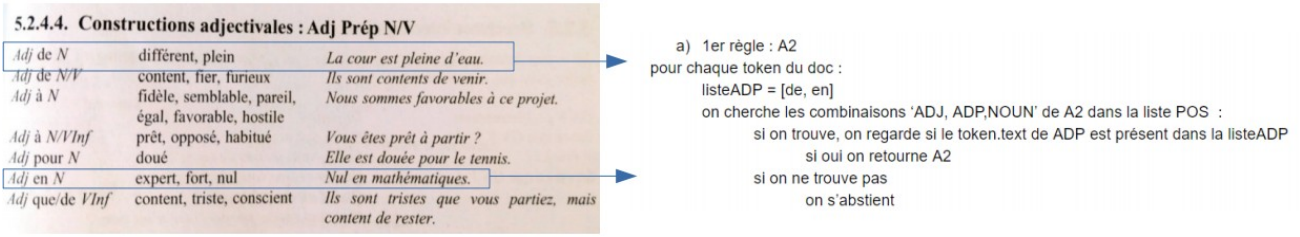


Figure 8: Exemple de règle d'un ouvrage écrite en pseudo-code

Comme nous l'avons indiqué dans la partie de présentation, Snorkel permet de définir des fonctions de labellisation. Dans notre cas, ces fonctions traduisent les règles syntaxiques identifiées. Celles-ci vont permettre de renvoyer des étiquettes de niveaux A1 à C2. Pour chacune des règles, nous avons utilisé une fonction de *preprocessing* de SpaCy, qui permettait d'étiqueter les lemmes, les part-of-speech et les informations syntaxiques des données (textes des sous-titres) pour pouvoir ensuite les traiter. Une fonction de labellisation ne peut renvoyer que deux étiquettes, une sur le niveau si les données répondent à tous les critères (A1, A2, B1, B2, C1 ou C2) et une qui indique que le système s'abstient (ABSTAIN), ce qui signifie que la structure recherchée dans la règle n'a pas été trouvée. Par exemple, pour les règles interrogatives de type **['NOUN','VERB']** où c'est l'intonation à l'oral qui indique la question ("le bébé pleure" / "le bébé pleure ?") , j'ai utilisé la

structure [**'NOUN','VERB','PUNCT'**] pour marquer le point d'interrogation à l'écrit en de fin de structure. La ponctuation pouvant être diverse, j'ai utilisé le **`token.text`** de **SpaCy** pour vérifier si le texte de **'PUNCT'** était un point d'interrogation. Si c'est le cas, la règle va renvoyer un niveau (par exemple A2), sinon elle va s'abstenir.

Il est aussi possible que la règle ne renvoie aucune étiquette de niveau, cela signifie donc que ce que l'on recherche n'est pas présent dans nos données ou alors qu'il y a eu une erreur d'étiquetage (cf. chapitre 5.3). Pour chaque type de règle, la structure est différente, car les informations sont fournies différemment par SpaCy.

```
# RÈGLE 7 : Construction nominale 3
@labeling_function(pre=[spacy_preproc])
def constru_noun3(x) :
    listPos = [i.pos_ for i in x.doc]
    nounStructure = ['NOUN','ADP','VERB']
    result = [(i, i+len(nounStructure)) for i in range(len(listPos)) if listPos[i:i+len(nounStructure)] == nounStructure]
    #print(result)
    if result:
        return B1
    else:
        return ABSTAIN
```

Figure 9: Exemple de règle nominale avec la fonction de construction nominale cherchant la structure ['NOUN','ADP','VERB']

Pour les règles nominales et les règles adjectivales basiques, il s'agit juste de chercher un *pattern* de *part-of-speech* (POS) particulier sans autre contrainte (figure 9). Pour les règles nominales, nous ne recherchons que des combinaisons de type [**'NOUN','ADP','VERB'**] (ex. "Une envie de partir") ou encore [**'NOUN','ADP','NOUN'**](ex. "Un joueur de football") (figure 9). La règle se déclenche et renvoie une étiquette dès que la combinaison est trouvée.

Pour les règles adjectivales plus complexes (c'est-à-dire demandant une préposition particulière), les règles pronominales, et une partie des règles interrogatives et passives, les fonctions cherchent aussi des patterns de POS, mais il y a des conditions en plus à respecter. Prenons l'exemple de la fonction adjectivale cherchant un pattern **['ADJ','ADP','PROPN']** où le ADP (préposition) doit être **["de","en"]** (figure 10). Quand le pattern est trouvé alors il faut que le système regarde le lemme qui correspond au part-of-speech de la préposition, s'il s'agit d'un "de" ou d'un "en" alors il renvoie une étiquette de niveau A2, sinon il s'abstient.

```
@labeling_function(pre=[spacy_preproc])
def constru_adj1a(x) :
    listPos = []
    listText = []
    for word in x.doc:
        listPos.append(word.pos_)
        listText.append(word.text)

    nounStructure = ['ADJ','ADP','PROPN']
    result = [(i, i+len(nounStructure)) for i in range(len(listPos)) if listPos[i:i+len(nounStructure)] == nounStructure]
    boolPresent = False
    for e in result:
        ibegin, iend = e
        if listText[ibegin+1] in ["de","en"]:
            boolPresent = True

    if boolPresent:
        return A2
    else:
        return ABSTAIN
```

Figure 10: Exemple d'une règle plus complexe avec la fonction adjectivale constru_adj1a

Pour les règles verbales, il faut d'abord chercher un verbe ou un auxiliaire en POS puis vérifier dans le **`token.tag_`** le temps et le mode. Prenons l'exemple du subjonctif, avec ce mode le système peut renvoyer deux étiquettes de niveaux différentes : B1 si le temps est au présent et B2 si c'est de l'imparfait. Tout d'abord, il faut créer une liste pour stocker le **`token.tag_`** d'un verbe ou d'un auxiliaire. Pour cela, nous cherchons les tags qui commencent par « ***VERB__Mood=Sub*** » ou «***AUX__Mood=Sub*** ». Par la suite, si il y a plus d'occurrence de verbe au présent que de verbe au subjonctif alors on renvoie l'étiquette B1.

Le script de Snorkel prend en entrée les différentes séquences du corpus d'OpenSubtitles et cherche nos règles sur les mots ou les phrases selon ce que l'on recherche dans notre structure. Les étiquettes sont renvoyées selon ce qui est trouvé par les règles sur les phrases et les mots. Dès qu'une structure est trouvé dans un séquence, la règle se déclenche et renvoie une étiquette pour la séquence concernée.

5.3 Erreurs de SpaCy

Nous savons que SpaCy n'est pas aussi bien entraîné sur le français que sur l'anglais, il est donc possible d'avoir des erreurs d'étiquetage qui peuvent porter préjudice au déclenchement des règles. Par exemple, le pronom « tu » va être étiqueté en tant que déterminant (DET), auxiliaire (AUX) ou encore verbe (VERB) au lieu de pronom (PRON). Autre exemple, les temps et les modes peuvent ne pas être correctement étiquetés. Ainsi nous aurons un temps au présent alors que le verbe est à l'imparfait. Cela peut empêcher une règle de se déclencher ou alors en déclencher une qui n'a pas lieu d'être. Cependant, nous avons choisi d'ignorer ce mauvais étiquetage, car il constitue une infime partie de ce que donne SpaCy et ainsi peut être considéré comme du bruit marginal. Malgré la décision de garder les erreurs, si les règles le permettaient, certains changements ont été apportés pour pallier certains problèmes d'étiquetage syntaxiques de SpaCy.

Chapitre 6 : Résultats et analyse

6.1 Analyse des premiers résultats de l'entraînement

6.1.1 Explication des résultats fournis par Snorkel

Avant d'analyser nos résultats nous allons expliquer ce que Snorkel présente en sortie. Les résultats sont sous forme d'un tableau à six colonnes, dont les quatre dernières représentent des paramètres pour le calcul (polarité, couverture, chevauchement, conflits). La figure 11 illustre un exemple de sortie de Snorkel que nous allons détailler.

	j	Polarity	Coverage	Overlaps	Conflicts
complex_adjective	0	[3]	0.000073	0.000073	0.000073
complex_adjective1	1	[]	0.000000	0.000000	0.000000
complex_adjective2	2	[2]	0.000696	0.000696	0.000696
complex_adjective2a	3	[2]	0.000049	0.000049	0.000049
constru_noun1	4	[0]	0.484636	0.473807	0.463955
constru_noun2	5	[2]	0.408919	0.400007	0.398152
constru_noun3	6	[2]	0.172260	0.171430	0.171100
verb_Sub	7	[3]	0.002320	0.002283	0.002283
verb_Sub1	8	[2]	0.120631	0.120387	0.119947
verb_Cond	9	[2]	0.198935	0.198447	0.197068
verb_Ind1	10	[0]	0.823809	0.771410	0.760069

Figure 11: Exemple de sortie de Snorkel

Chaque règle reçoit en entrée les sous-titres de l'extrait à traiter. Le corpus regroupant l'ensemble des extraits issus d'OPUS (cf. chapitre 4.1) sert à apprendre comment les règles se déclenchent et à obtenir des résultats pour évaluer la pertinence des règles.

Ainsi, après le nom de chaque règle et le numéro associé on trouve :

1) La polarité : Il s'agit de l'ensemble des étiquettes uniques que la fonction de labellisation a produit. Les numéros entre crochet correspondent donc aux étiquettes de niveaux (-1 pour l'abstention, 0 pour A1, 1 pour A2, 2 pour B1, etc, jusqu'à C1). Il faut noter que si les crochets sont vides cela signifie que la règle ne s'est jamais déclenchée.

2) Couverture : Il s'agit de la proportion de données annotées par la fonction de labellisation. Si la valeur est 0,5 alors cette règle a permis d'annoter 50% des extraits traités.

3) Chevauchements : Il s'agit de la proportion des données annotées par la fonction de labellisation considérée et au moins une autre règle. Dans notre cas, si plusieurs règles se sont déclenchées sur une même séquence de film alors elles vont se chevaucher.

4) Conflits : Elle fonctionne de pair avec le chevauchement, puisqu'il s'agit également de la proportion des données annotées par fonction de labellisation considérée et au moins par une autre fonction, mais cette fois-ci les étiquettes ne sont pas les mêmes (désaccord : l'une juge qu'on a un niveau A1 et l'autre C2 par exemple).

Ce sont ces différents critères qui vont nous permettre d'analyser la pertinence de nos fonctions de labellisation. En effet, si une règle ne se déclenche pas, ou a une couverture trop faible, cela signifie qu'elle doit être modifiée ou supprimée.

Il est ensuite possible de définir un modèle génératif à partir des votes produits par les fonctions de labellisation et que l'on peut considérer comme autant de résultats "bruités" par rapport à l'étiquette réelle (Ratner et al., 2017).

6.1.2 Analyse des résultats avec les règles syntaxiques

Nos premiers résultats ont avant tout servi à ajuster l'ensemble de nos règles, c'est-à-dire à modifier les règles qui ne se déclenchaient pas. Par exemple, nous avons modifié une règle interrogative qui ne se déclenchait pas à cause d'un mauvais étiquetage syntaxique de **`SpaCy`**. La règle cherchait le pattern **`['VERB,PRON,SCONJ,PRON,VERB']`** pour des phrases du type "est-ce que tu viens ?", nous avons donc remplacé la recherche de pattern par la recherche de l'expression interrogative **`['est','-ce','que']`**.

Une fois ce travail effectué nous avons commencé à analyser les résultats par le prisme des paramètres fournis par Snorkel. Afin d'éclaircir ce point nous allons illustrer chaque paramètre à l'aide de capture d'écran de nos résultats.

	j	Polarity	Coverage	Overlaps	Conflicts
avg_length	0	[3]	0.087033	0.084177	0.084152
complex_adjective	1	[3]	0.000073	0.000073	0.000073
complex_adjective2	2	[2]	0.000696	0.000696	0.000696
complex_adjective2a	3	[2]	0.000049	0.000049	0.000049
constru_noun1	4	[0]	0.484636	0.474417	0.465530
constru_noun2	5	[2]	0.408919	0.400813	0.399348
constru_noun3	6	[2]	0.172260	0.171552	0.171235
verb_Sub	7	[3]	0.002320	0.002295	0.002283
verb_Sub1	8	[2]	0.120631	0.120411	0.120118
verb_Cond	9	[2]	0.198935	0.198472	0.197324
verb_Ind1	10	[0]	0.823809	0.775561	0.765245

Figure 12: Exemple de sortie Snorkel pour la couverture

La figure 12 permet d'illustrer le paramètre de couverture (ou coverage). Celui-ci montre le pourcentage de couverture de chaque fonction de labellisation. Par exemple, pour les fonctions *verb_Sub* (qui cherche la présence d'un verbe au subjonctif au passé) et ***`complex_adjective`*** (qui cherche une pattern de **`['ADJ','ADJ','NOUN','ADJ','ADJ']`** pour des phrases comme « le joli

petit chat roux tigré »), nous voyons bien qu'elles ont une couverture faible (<1%), ce qui signifie que la fonction n'a pas été déclenchée sur un grand nombre de données. A contrario, si on regarde la couverture d'une fonction comme ***verb_Ind1*** (qui cherche la présence d'un verbe à l'indicatif présent) nous voyons qu'elle a une couverture de plus de 80%. La structure de certaines règles peut expliquer ce résultat. En effet, celles-ci peuvent être très précises, ou alors ce qu'elles recherchent n'est pas courant dans le type de données que nous utilisons (transcription de dialogue) La règle ***verb_Ind1*** recherche, par exemple, des verbes à l'indicatif (mode) présent (temps). Or nous savons que l'indicatif est un mode très utilisé en français et que le présent est aussi beaucoup utilisé dans les dialogues, c'est d'ailleurs pour cela que le conditionnel et le subjonctif, plus rares dans des discussions, ont une couverture plus faible.

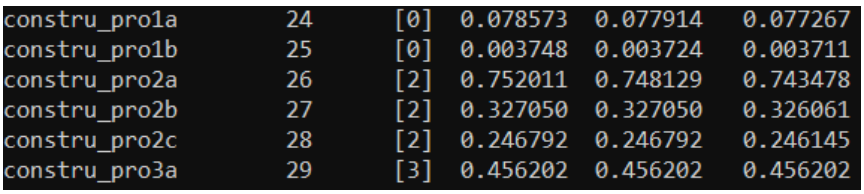


Figure 13: Exemple de sortie Snorkel pour la polarité

La polarité (figure 13) montre l'étiquette renvoyée par la fonction. Dans nos résultats nous n'avons aucune étiquette d'abstention noté [-1] ce qui signifie que chaque fonction a été déclenchée et donc que leur couverture n'est pas nulle. Sur les 6 fonctions présentes ci-dessus, deux renvoient une étiquette A1, 3 renvoient une étiquette B1 et enfin une renvoie une étiquette B2. Si une fonction ne se déclenche pas alors il y aura des crochets vide []. Si c'est le cas, cela signifie que le pattern de recherche n'a pas été trouvé dans les données d'entrée.

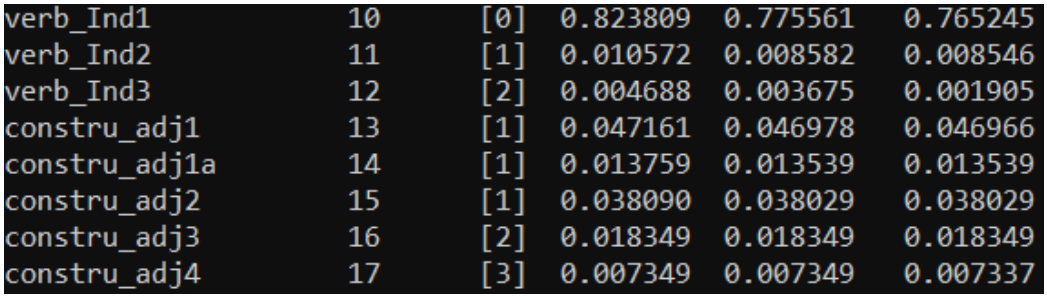
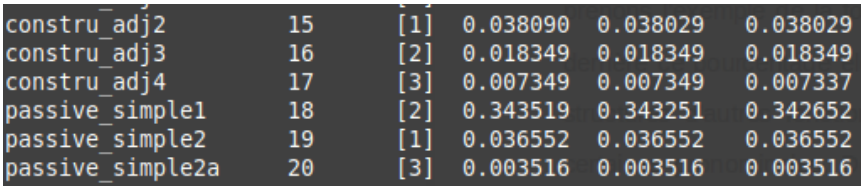


Figure 14: Exemple de sortie pour le chevauchement

Pour ce qui est du chevauchement (ou overlap) dans la figure 14, nous avons dans nos résultats de nombreuses règles qui se chevauchent car les structures de certains motifs englobent les structures des autres. A titre d'illustration, prenons l'exemple de la fonction verbale ***verb_Ind1*** qui a un chevauchement de 77 %, ce pourcentage peut s'expliquer par la présence de verbe dans certaines structures recherchées par d'autres règles, comme les constructions adjectivales (fonction ***constru_adj2*** > [**'ADJ','ADP','VERB'**]) ou encore certaines règles pronominales (***constru_pro3a*** > [**'PRON','PRON', 'VERB'**]). De ce fait, si une des règles verbales se déclenche sur une même phrase que ces deux règles alors il y aura du chevauchement. Par exemple, la

phrase « Tu le lui dis. » peut déclencher la règle ***verb_Ind1*** avec le verbe “dire” et la règle ***constru_pro3a*** avec la structure.



constru_adj2	15	[1]	0.038090	0.038029	0.038029
constru_adj3	16	[2]	0.018349	0.018349	0.018349
constru_adj4	17	[3]	0.007349	0.007349	0.007337
passive_simple1	18	[2]	0.343519	0.343251	0.342652
passive_simple2	19	[1]	0.036552	0.036552	0.036552
passive_simple2a	20	[3]	0.003516	0.003516	0.003516

Figure 15: Exemple de sortie pour le conflit

Avec la figure 15, nous allons illustrer la dernière colonne qui est celle du conflit, elle montre l’accord et le désaccord des fonctions de labellisation dans l’attribution des étiquettes de niveaux. Comme nous l’avons dit dans la partie présentation des résultats fournis par Snorkel, le conflit et chevauchement vont souvent de pair. De ce fait les pourcentages entre les deux dernières colonnes sont, la plupart du temps, les mêmes ou légèrement différents. La fonction ***passive_simple1*** par exemple qui recherche la structure [**‘AUX’,’VERB’**], pour une phrase comme « La réunion est reportée », avec un tag « passive » est en conflit dans 34% des cas. Cela peut s’expliquer par le fait que cette fonction va entrer en confrontation avec d’autres règles comme les verbes ou alors certaines constructions nominales, adjectivales et pronominales. Il suffit que les autres fonctions ne renvoient pas la même étiquette pour qu’il y ait conflit. Cependant, les conflits ne sont pas un problème pour le système, puisqu’il doit lui permettre de faire des choix entre les différentes étiquettes. Ce choix peut se faire en calculant l’*accuracy* et la corrélation des étiquettes ou alors en utilisant un vote à la majorité de l’étiquette la plus retournée par les fonctions.

Les résultats de notre première itération montrent que notre système est capable de fournir un ensemble d’étiquettes de niveaux en fonction des paramètres syntaxiques que nous lui avons donnés sous la forme de règle. En effet, le but de notre entraînement est de pouvoir affiner nos règles en modifiant ou enlevant celles qui ne se déclenchent pas. Or nous voyons que sur nos 60 règles, 6 ne se déclenchent pas, ce qui signifie qu’une grande majorité de nos règles se sont déclenchées sur le corpus d’entraînement. Pour ce qui est de la précision de notre système, nous pourrons la calculer avec notre corpus de test qui est déjà annoté par étiquettes de niveaux de difficulté par des enseignants de FLE. Pour pouvoir améliorer notre modèle, nous allons maintenant ajouter des règles lexicales pour prendre en compte la complexité lexicale des extraits considérés, ce qui peut être reflété par la fréquence des mots employés.

6.1.2 Analyse des résultats avec les règles syntaxiques et lexicales

Pour nos règles lexicales, nous utilisons le lexique FLELex, si un des mots du corpus apparaît dans la liste des mots de B2, C1 et C2 alors le niveau associé à ce mot va être renvoyé.

Cette analyse nous permet de voir les conflits entre les règles lexicales et syntaxiques. En effet, il est possible que la syntaxe d'une phrase soit simple mais que ce soit le lexique qui rende la compréhension difficile. La figure 16, ci-dessous, montre les résultats de l'entraînement des règles syntaxiques et lexicales. Nous voyons que sur les 18 000 mots du corpus FLElex_CRF seulement 5 000 sont sélectionnés, car ils apparaissent dans le lexique à partir du niveau B2.

	j	Polarity	Coverage	Overlaps	Conflicts
avg_length	0	[3]	0.087033	0.084555	0.084360
complex_adjective	1	[3]	0.000073	0.000073	0.000073
complex_adjective2	2	[2]	0.000696	0.000696	0.000696
complex_adjective2a	3	[2]	0.000049	0.000049	0.000049
constru_noun1	4	[0]	0.484636	0.475248	0.467581
...	...	...	...	...	...
keyword_évier_NOM	4965	[4]	0.000037	0.000037	0.000037
keyword_évincer_VER	4966	[5]	0.000061	0.000061	0.000061
keyword_évocateur_ADJ	4967	[5]	0.000012	0.000012	0.000012
keyword_évocation_NOM	4968	[3]	0.000012	0.000012	0.000012
keyword_îlot_NOM	4969	[3]	0.000183	0.000183	0.000183

Figure 16: Exemple de sortie avec la règle lexicale

Avec les 5 dernières lignes du résultat, nous remarquons que la couverture, le chevauchement et le conflit sont les mêmes, ce qui signifie que ces 5 mots se sont retrouvés dans des structures des règles syntaxiques et qu'à chaque fois l'étiquette de niveau renvoyée était différente de celle renvoyé par la règle syntaxique. La couverture de chacun de ces mots est très faible (<1%). On explique ce faible score par l'utilisation peu courante de certains mots dans un contexte non précis. Par exemple, le mot *évier* est un mot rare dans une conversation sauf exception dans un contexte précis (cuisine, salle de bain, pièce d'eau…), il en va de même pour *îlot* qui lui est encore moins courant dans une conversation ordinaire.

L'addition d'un ensemble de règles lexicales à nos règles syntaxiques nous permet de rendre notre système plus complet puisque nous prenons en compte deux paramètres. Après cette première analyse, nous voulons tester notre système sur un corpus test différent de celui d'OpenSubtitle, mais pour lequel nous disposons d'une vérité de terrain (Randria et al., 2020). En effet, pour l'instant nous ne pouvons pas affirmer que l'étiquetage de notre système peut être en accord avec ce que pensent des enseignants du FLE. Pour tester notre système, nous allons donc utiliser le corpus test d'Estelle Randria présenté plus haut (cf. chapitre 4.1).

6.2 Analyse des résultats avec le corpus test

L'analyse des résultats du corpus de test doit permettre de vérifier l'*accuracy* de nos fonctions de labellisations. En effet, Snorkel dispose pour cela de la fonction **`label_model_acc`**. Pour tester notre modèle nous disposons donc du corpus d'Estelle Randria constitué de 55 extraits pour 12 films, ainsi que des niveaux de difficulté donnés par des enseignants de FLE pour chaque extrait. Pour notre test, nous allons ainsi comparer les étiquettes renvoyées par notre système avec celles que les enseignants ont attribuées.

Pour pouvoir travailler avec le corpus d'Estelle Randria qui est composé de fichier srt, nous avons utilisé **`pysrt`**, puis assemblé les notes des enseignants avec le titre des extraits de film et leurs sous-titres dans un dictionnaire, où le titre de l'extrait est la clé et les notes (lexicale ou syntaxique) et les sous-titres sont les valeurs (figure 17). Les notes données par les enseignants étaient de 0 à 100 sur une échelle de difficulté. Pour correspondre à notre propre échelle de niveau, nous avons tout simplement divisé l'échelle des notes des enseignants par 4 pour faire une échelle qui corresponde au 4 niveaux les plus utilisés dans notre système (c'est-à-dire A1, A2, B1 et B2).

```
{'Amelie_poulain_25': ['2', '2', 'Un petit vin chaud avec des Spéculoos .', 'Merci .', "Je crois que j' ai été un peu dur l'autre
'Amelie_poulain_31': ['2', '1', 'Une heure plus tard , au onze boulevard Saint-Martin ,...', '... Amélie entre dans un magasin de
'Amelie_poulain_41-1': ['3', '2', 'Nous sommes le vingt-huit septembre mille neuf cent quatre-vingt-dix-sept , il est exactement
'Cyrano_de_Bergerac_24': ['4', '3', "Hein ? Quoi ? Quel est cet homme ? Et d'où nous tombe t-il ?", '- De la Lune !\n- Comment?',
'Cyrano_de_Bergerac_28-1': ['2', '2', "- Qu'ont-ils donc?\n- Ils ont faim!", "- Et alors? Moi aussi .\n- Dans les oreilles , moi
```

Figure 17: Corpus Randria et al., (2020) après modification pour test

Pour illustrer, nous pouvons prendre l'exemple de l'extrait « Cyrano_de_Bergerac_24 » (Figure 18) qui est donc considéré par les enseignants de FLE comme ayant un niveau B2 en lexique mais un niveau B1 en syntaxe.

```
1
00:00:00,555 --> 00:00:03,415
Hein ? Quoi ? Quel est cet homme ? Et d'où nous tombe t-il ?

2
00:00:03,455 --> 00:00:04,935
- De la Lune !
- Comment?

3
00:00:05,375 --> 00:00:09,135
- Terrien , quelle heure est-il ?
- Qui est-ce?

4
00:00:09,175 --> 00:00:10,895
J'y vois mal . N'a t-il plus sa raison?
```

Figure 18: Extrait « Cyrano_de_Bergerac_24 » du corpus de Randria et al., (2020)

Le résultat obtenu permet de connaître *l'accuracy* des étiquettes que notre système a renvoyé par rapport à celles données par des enseignants de FLE. Nous obtenons une *accuracy* de 76,5 %. Ce qui signifie que 76,5 % de nos étiquettes sont en accord avec les étiquettes des enseignants. Ce pourcentage montre que notre système donne des résultats convenables mais qu'ils faut encore améliorer nos règles.

Chapitre 7 : Améliorations possibles

Comme nous l'avons constaté pour nos résultats, certaines fonctions ne se déclenchent pas pour plusieurs raisons. La première est le caractère « écrit » textuel des règles, car malgré le fait que nous travaillons sur des données écrites textuelles, celles-ci sont avant tout calquées sur de l'oral. Et même s'il y a une certaine mise en forme textuelle des interactions, les sous-titres restent quand même plus proche de la syntaxe orale. De ce fait, nos règles tout droit sorties de livre peuvent parfois être inadaptées à la réalité langagière de l'oral. Ainsi, il serait intéressant de faire un travail pour adapter ces règles à un contexte oral, pour pouvoir réduire les erreurs et le nombre de règles non déclenchées.

Une deuxième amélioration possible serait d'utiliser un autre annotateur que SpaCy pour limiter le nombre d'erreurs d'étiquetage, car nous savons que certaines de nos règles ne se sont pas déclenchées à cause de problème de ce genre. D'autres ont renvoyé des ABSTAIN, car les critères que nous cherchions dans les tags n'étaient pas toujours présents. Par exemple, pour le passif, il y a des fois où dans le tag du verbe la mention « voice=passive » n'est pas présente. S'il n'est pas aisé de changer d'annotateur, car avec snorkel, SpaCy est le mieux intégré, il est cependant possible d'adapter certaines règles pour qu'elles prennent en compte les erreurs d'étiquetage et de manque d'information. Pour illustrer les erreurs de SpaCy nous pouvons prendre l'exemple de la fonction interrogative ***struc_interro2a*** qui cherchait la structure ['**VERB**', '**PRON**','**SCONJ**','**PRON**','**VERB**'] (figure 19) mais nous avons dû changer car la construction de la liste des part-of-speech pour rechercher la structure ne se faisait pas bien.

```
# est-ce que tu viens ?
@labeling_function(pre=[spacy_preproc])
def struct_interro2a(x) :
    listPos = []
    listTag = []
    listText = []
    for word in x.doc:
        listPos.append(word.pos_)
        listTag.append(word.tag_)
        listText.append(word.text)
    print(listPos)
    print(listText)
    interroStructure = ['VERB','PRON','SCONJ','PRON','VERB']
    result = [(i, i+len(interroStructure)) for i in range(len(listPos)) if listPos[i:i+len(interroStructure)] == interroStructure]
    if result:
        return A2
    else:
        return ABSTAIN
```

Figure 19: Fonction interrogative struct_interro2a avant modification

En effet, il manquait le '**PRON**', de ce fait comme nous cherchions des phrases interrogatives commençant par « est-ce que », nous avons choisi de remplacer la structure par l'expression interrogative ['**est**','**-ce**','**que**'] (figure 20).

```
@labeling_function(pre=[spacy_preproc])
def struct_interro2a(x) :
	listText = []
	listPos = []
	for word in x.doc:
		listText.append(word.text)
		listPos.append(word.pos_)
	# ~ print(listPos)
	# ~ print(listText)
	interroStructure = ['est', '-ce', 'que']
	result = [(i, i+len(interroStructure)) for i in range(len(listText)) if listText[i:i+len(interroStructure)] == interroStructure]
	# ~ print(result)
	if result:
		return A2
	else:
		return ABSTAIN
```

Figure 20: Fonction interrogative struct_interro2a après modification

Une amélioration est également possible pour le temps d'exécution du script. En effet, pour ce qui est du script avec les règles syntaxiques seules, le temps d'exécution est d'une 1h30. Par contre, avec l'ajout du paramètre lexical avec ses 18 000 fréquences de mots, le temps est passé à plus de 10h d'exécution. Or nous voulons créer un système qui puisse donner des réponses aux enseignants assez rapidement, un temps d'exécution de 10h est beaucoup trop long. Pour cela, il est possible de sauvegarder les résultats de l'entraînement dans un fichier **`pickle`**.

PARTIE 4

-

BILAN DU STAGE

Chapitre 8 : Bilan

8.1 Travail réalisé

Pour ce stage notre objectif était de créer un système qui permet d'annoter automatiquement de grandes quantités de données selon des niveaux de difficulté.

La première étape a donc été de définir ces niveaux de difficulté, notre choix s'est porté sur les étiquettes du CECRL. A partir de là, j'ai pu définir, implémenter et lancer des règles syntaxiques et lexicales.

Pour les règles syntaxiques, je me suis basé sur des ouvrages du Français Langue Étrangère des différents niveaux du CECRL. J'ai sélectionné dans ces ouvrages des structures pour construire mes règles, cette sélection s'est faite en lien avec les informations fournies par SpaCy et la pertinence des règles dans un contexte de transcription de l'oral (sous-titre). Une fois les structures sélectionnées, j'ai travaillé à l'implémentation des règles d'abord sur un script utilisant uniquement SpaCy puis une fois la règle mise au point et fonctionnelle, je suis passée sur le script Snorkel permettant de renvoyer les étiquettes. Ce script a été fourni par mon tuteur Tim Van De Cruys au début de mon stage, il a été modifié et amélioré à quelques reprises pendant mon stage(par exemple pour récupérer uniquement les films dont les sous-titres sont français, gérer les problèmes d'identifiants de certains films). Chacune des règles syntaxiques a demandé un travail de réflexion sur la langue, c'est-à-dire se questionner sur la réalité linguistique de l'oral et de la transcription. Pour les règles lexicales, il a juste fallu ajouter au script Snorkel le fichier des fréquences d'apparition des mots par niveaux constitué comme ceci : mot, part-of-speech, niveau.

Une fois ce travail fait, j'ai lancé le script Snorkel depuis le serveur Osirim, analyser les résultats et apporter des modifications à certaines règles si nécessaire. Quand la majorité des règles ont été déclenchées par le système pour le corpus d'entraînement, nous avons décidé de le lancer sur le corpus test.

Pour la partie test, Estelle Randria nous a fourni son corpus (fichiers srt) ainsi que les notes des enseignants (fichier csv). Pour le fichier csv, ma tutrice Isabelle Ferrané s'est occupé de modifier le fichier pour que les notes des enseignants correspondent à nos niveaux de difficulté.

J'ai donc récupéré le corpus et le fichier des notes, et grâce à un script j'ai pu stocker les informations dans un dictionnaire python. Celui-ci va permettre ensuite de récupérer plus facilement le corpus et les notes pour pouvoir tester notre système puis comparer nos étiquettes de niveaux avec les étiquettes données par les enseignants.

8.2 Bilan personnel

Mon stage s’est déroulé pendant une période un peu particulière puisqu’il s’est fait en première partie en télétravail. De ce fait, je pense ne pas avoir avancé autant que je le voulais. Le cadre d’un travail à la maison sans pouvoir échanger avec d’autres stagiaires ou les personnes de l’équipe par exemple a été compliqué pour moi. En effet, en dehors de mes encadrants, je n’ai pas pu bénéficier d’une aide suivie extérieure sur des problèmes techniques en informatique, ou des échanges concernant des problèmes de règles linguistiques. De plus, la plupart des réunions se passant en visioconférence, il est difficile d’expliquer un problème ou de se débattre sur la structure d’un code.

Cependant ce stage a quand même été très bénéfique pour moi puisque j'ai dû travailler en autonomie du fait du contexte de crise sanitaire et ainsi me perfectionner sur l'installation de divers outils et programmes.

Tout d'abord lorsque j'ai voulu installer Anaconda sur mon ordinateur personnel je me suis rendu compte que mon système d'exploitation LINUX était obsolète. Il a donc fallu que j'installe la dernière version de Linux Mint (la version 19.3 « Tricia ») et refaire toutes mes installations à commencer par Python et Anaconda.

Une fois les installations basiques faites grâce à Conda, qui s'est installé avec Anaconda, j'ai pu créer un environnement virtuel qui peut être lancé avec la commande <conda activate my_env> et installer par la suite, dans cet environnement, les différents outils dont j'avais besoin pour mon stage (snorkel, spacy, panda, pickle...).

Sur un aspect plus linguistique, le travail que j'ai effectué pour la recherche de règles a été très intéressant puisque j'ai dû réfléchir à la manière dont fonctionnait la langue française. J'ai ainsi pu mettre en pratique ce que j'avais appris en licence de linguistique, notamment en syntaxe et dans certains cours où nous abordions l'apprentissage des langues.
De plus, ayant un profil de linguiste, travailler avec Isabelle et Tim a été très enrichissant et stimulant pour moi, puisque leur domaine est celui de l'informatique. J'ai ainsi appris à travailler sur les serveurs à distance, pratique que je n'avais jamais eue jusqu'alors, et j'ai également appris le fonctionnement du *data programming* à travers l'outil Snorkel.

Bibliographie

Beacco, J., Lepage, S., Porquier, R., & Riba, P. (2008). *Niveau A2 pour le français (utilisateur / apprenant indépendant) niveau intermédiaire*. Paris: Didier.

Beacco, J., Blin, B., Houles, E., Lepage, S., & Riba, P. (2011). *Niveau B1 pour le français (utilisateur / apprenant indépendant) niveau seuil*. Paris: Didier.

Beacco, J., Bouquet, S., & Porquier, R. (2004). *Niveau B2 pour le français (utilisateur / apprenant indépendant)*. Paris: Didier.

Chen, X., & Meurers, D. (2016, June). Characterizing text difficulty with word frequencies. In *Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications* (pp. 84-94).

Collins-Thompson, K., & Callan, J. (2005). Predicting reading difficulty with statistical language models. *Journal of the American Society for Information Science and Technology*, *56*(13), 1448-1462.

Conquet, A., & Richaudeau, F. (1973). Cinq méthodes de mesure de la lisibilité. *Communication & Langages*, *17*(1), 5-16.

Dale, E., & Chall, J. S. (1948). A formula for predicting readability: Instructions. *Educational research bulletin*, 37-54.

De Clercq, B. (2016). Le développement de la complexité syntaxique en français langue seconde: complexité structurelle et diversité. In *SHS Web of Conferences* (Vol. 27, p. 07006). EDP Sciences.

François, T., Gala, N., Watrin, P., & Fairon, C. (2014, May). FLELex: a graded lexical resource for French foreign learners.

Elola, M., & Roubaud, M. N. (2018). La compréhension du lexique dans les manuels scolaires d'histoire: Un exemple de recherche en lisibilité linguistique.

Lison, P., & Tiedemann, J. (2016). Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles.

Pilán, I., Volodina, E., & Zesch, T. (2016, December). Predicting proficiency levels in learner writings by transferring a linguistic complexity model from expert-written coursebooks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* (pp. 2101-2111).

Randria, E., Fontan, L., Le Coz, M., Ferrané, I., & Pinquier, J. (2020, May). Subjective evaluation of comprehensibility in movie interactions. In *Proceedings of The 12th Language Resources and Evaluation Conference* (pp. 2348-2357).

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., & Ré, C. (2017, November). Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases* (Vol. 11, No. 3, p. 269). NIH Public Access.

Riba, P. (2016). *Niveaux C1 - C2 pour le français (utilisateur expérimenté) niveau autonome / maîtrise*. Paris: Didier.

Sitographie

spaCy · Industrial-strength Natural Language Processing in Python. (s. d.). spaCy. Consulté 27 avril 2020, à l'adresse https://spacy.io/

Snorkel. (s. d.). Snorkel. Consulté 24 mars 2020, à l'adresse https://www.snorkel.org/

Index des figures

Figure 1: Exemples sur le film “Le petit Nicolas” issus d*u* corpus de Randria et al. (20*20*).....14
Figure 2: Fonction de labellisation de Snorkel.....20
Figure 3: Schéma du fonctionnement de Snorkel (Ratner et al., 2017).....20
Figure 4: Exemple du corpus test avec « Le fabuleux destin d’Amélie Poulain » (Randria et al., 2020).....22
Figure 5: Exemple de sortie SpaCy où “ceci” est la propriété token.text, “PRON” est *la propriété* token.pos_ et PRON_Number=Sing|PronType=Dem” est la propriété token.tag_.....24
Figure 6: Exemple fle_converted_B2.txt.....26
Figure 7: Exemple de fréquence pour les mots "évocateur" et "évocation" dans FLElex_CRF (François et al., 2014).....26
Figure 8: Exemple de règle d'un ouvrage écrite en pseudo-code.....29
Figure 9: Exemple de règle nominale avec la fonction de construction nominale cherchant la structure [‘NOUN’,‘ADP’ ,‘VERB’].....30
Figure 10: Exemple d'une règle plus complexe avec la fonction adjectivale constru_adj1a.....31
Figure 11: Exemple de sortie de Snorkel.....32
Figure 12: Exemple de sortie Snorkel pour la couverture.....33
Figure 13: Exemple de sortie Snorkel pour la polarité.....34
Figure 14: Exemple de sortie pour le chevauchement.....34
Figure 15: Exemple de sortie pour le *conflit*.....35
Figure 16: Exemple de sortie avec la règle lexicale.....36
Figure 17: Corpus Randria et al., (2020) après modification pour test.....37
Figure 18: Extrait « Cyrano_de_Bergerac_24 » du corpus de Randria et al., (2020).....37
Figure 19: Fonction interrogative struct_interro2a avant modification.....39
Figure 20: Fonction interrogative struct_interro2a après modification.....40

Index des tableaux

Tableau 1: Récapitulatif des données des deux corpus.....23
Tableau 2: Structure des règles syntaxiques.....25

Index des annexes

1. Tableau récapitulatif des règles.....50
2. Exemple de structure dans les ouvrages de FLE.....56
3. Exemple de quelques règles syntaxiques issues du script Snorkel.....58

Annexes

1. Tableau récapitulatif des règles

	REGLE - INTITULE	PSEUDO-CODE	PYTHON	Script Snorkel	Informations recherchées	Niveaux
	REGLE X (nom de la fonction)	à faire / en cours / fait	en cours / fait	testé ou pas		A1 / A2 / B1 / B2
Règle longueur phrase						
1	Règle 1 (avg_length)	fait	fait	testé	Phrase (mot>13)	B2
Règles adjectivales						
2	Règle 2 (complex_adjective)	fait	fait	testé	'ADJ','ADJ', 'NOUN', 'ADJ', 'ADJ'	B2
3	Règle 3 (complex_adjective1)	fait	fait	testé	'ADJ','ADJ', 'NOUN', 'ADJ','CONJ', 'ADJ'	B2
4	Règle 4 (complex_adjective2)	fait	fait	testé	'ADJ','ADJ','NOUN','ADJ'	B1
5	Règle 5 (complex_adjective2a)	fait	fait	testé	'ADJ','PUNCT','ADJ','NOUN','ADJ'	B1
6	Règle 9 (constru_adj1)	fait	fait	testé	'ADJ','**ADP**','NOUN' **["de" ou "en"]**	A2
7	Règle 10 (constru_adj1a)	fait	fait	testé	'ADJ','**ADP**','PROPN' **["de" ou "en"]**	A2
8	Règle 11 (constru_adj2)	fait	fait	testé	'ADJ','**ADP**','VERB' **["de" ou "à"]**	A2
9	Règle 12 (constru_adj3)	fait	fait	testé	'ADJ','**ADP**','VERB' **"à"**	B1

10	Règle 13 (constru_adj4)	fait	fait	testé	'ADJ','**ADP**','NOUN' **"pour"**	B2
Règles nominales						
11	Règle 6 (constru_noun1)	fait	fait	testé	'NOUN', 'NOUN'	A1
12	Règle 7 (constru_noun2)	fait	fait	testé	'NOUN','ADP','NOUN'	B1
13	Règle 8 (constru_noun3)	fait	fait	testé	'NOUN','ADP','VERB'	B1
Règles verbales						
14	Règle 14 (verb_Sub)	fait	fait	testé	VERB__Mood=Sub AUX__Mood=Sub Tense=Past	B2
15	Règle 15 (verb_Sub1)	fait	fait	testé	VERB__Mood=Sub AUX__Mood=Sub Tense=Pres	B1
16	Règle 16 (verb_Cond)	fait	fait	testé	VERB__Mood=Cond AUX__Mood=Cond	B1
17	Règle 17 (verb_Ind1)	fait	fait	testé	VERB__Mood=Ind AUX__Mood=Ind Tense=Pres	A1
18	Règle 18 (verb_Ind2)	fait	fait	testé	VERB__Mood=Ind AUX__Mood=Ind Tense=Imp	A2
19	Règle 19 (verb_Ind3)	fait	fait	testé	VERB__Mood=Ind AUX__Mood=Ind Tense=Fut	B1
Règles passives						
20	Règle 20 (passive_simple1)	fait	fait	testé	'AUX', 'VERB'	B1
21	Règle 21 (passive_simple2)	fait	fait	testé	'a', 'été'	A2

22	Règle 22 (passive_simple2a)	fait	fait	testé	'avait', 'été'	B2
23	Règle 23 (passive_simple2b)	fait	fait	testé	'ont', 'été'	B2
24	Règle 24 (passive_simple2c)	fait	fait	testé	'avaient', 'été'	B2
25	Règle 25 (passive_complex)	fait	fait	testé	'AUX','AUX','VERB','ADP'	B1
Règles passives pronominales						
26	Règle 26 (passive_pro1)	fait	fait	testé	'PRON','VERB','VERB','ADP'	B2
27	Règle 27 (passive_pro2)	fait	fait	testé	'PRON','VERB','VERB'	B1
Règles pronominales						
28	Règle 28 (constru_pro1a)	fait	fait	testé	'**VERB**','PRON' **« Mood=Imp »**	A1
29	Règle 29 (constru_pro1b)	fait	fait	testé	'**VERB**','PUNCT','PRON' **« Mood=Imp »**	A1
30	Règle 30 (constru_pro2a)	fait	fait	testé	'PRON','VERB'	B1
31	Règle 31 (constru_pro2b)	fait	fait	testé	'PRON','VERB','**VERB**' **VerbForm=Inf**	B1
32	Règle 32 (constru_pro2c)	fait	fait	testé	'VERB','PRON','VERB' **VerbForm=Inf**	B1
33	Règle 33 (constru_pro3a)	fait	fait	testé	'PRON','PRON','VERB'	B2
34	Règle 34 (constru_pro3b)	fait	fait	testé	'VERB','PRON','PRON','**VERB**' **VerbForm=Inf**	B2
35	Règle 35 (constru_pro3c)	fait	fait	testé	'**VERB**','PRON','PRON' **Mood=Imp**	B2

36	Règle 36 (constru_pro_neg1a)	fait	fait	testé	'**ADV**','PRON','VERB','**ADV**' **Polarity=Neg**	B1
37	Règle 37 (constru_pro_neg1b)	fait	fait	testé	'**ADV**','PRON','VERB','**ADV**','***VERB***' **Polarity=Neg** ***VerbForm=Inf***	B1
38	Règle 38 (constru_pro_neg1c)	fait	fait	testé	'**ADV**','PRON','***VERB***' **Polarity=Neg** ***VerbForm=Inf***	B1
39	Règle 39 (constru_pro_neg2a)	fait	fait	testé	'**ADV**','PRON','PRON','***VERB***' **Polarity=Neg** ***VerbForm=Inf***	B1
40	Règle 40 (constru_pro_neg2b)	fait	fait	testé	'**ADV**','PRON','PRON','VERB','**ADV**' **Polarity=Neg**	B2
Règles interrogatives						
41	Règle 41 (struct_interro1a)	fait	fait	testé	'**PRON**','VERB','***PUNCT***' **PronType=Int** ***«?»***	A1
42	Règle 42 (struct_interro1a1)	fait	fait	testé	'**PRON**','VERB','ADV','***PUNCT***' **PronType=Int** ***«?»***	A1
43	Règle 43 (struct_interro1a2)	fait	fait	testé	'**PRON**','VERB','DET','NOUN','***PUNCT***' **PronType=Int** ***«?»***	A1
44	Règle 44 (struct_interro1a3)	fait	fait	testé	'**PRON**','VERB','PROPN','***PUNCT***' **PronType=Int** ***«?»***	A1
45	Règle 45 (struct_interro1b)	fait	fait	testé	'NOUN','VERB','**PUNCT**' **«?»**	A1
46	Règle 46 (struct_interro1b1)	fait	fait	testé	'NOUN','VERB','DET','NOUN', '**PUNCT**'	A1

					«?»	
47	Règle 47 (struct_interro1c)	fait	fait	testé	'PROPN','VERB','**PUNCT**' **«?»**	A1
48	Règle 48 (struct_interro1c1)	fait	fait	testé	'PROPN','VERB','VERB','ADV','**PUNCT**' **«?»**	A1
49	Règle 49 (struct_interro1c2)	fait	fait	testé	'PROPN','VERB','ADV','**PUNCT**' **«?»**	A1
50	Règle 50 (struct_interro2a)	fait	fait	testé	'est', '-ce', 'que'	A2
51	Règle 51 (struct_interro2b)	fait	fait	testé	'Est', '-ce', 'que'	A2
52	Règle 52 (struct_interro2c)	fait	fait	testé	'**ADV**','PRON','VERB','DET','NOUN' **PronType=Int** **PronType=Rel**	A2
53	Règle 53 (struct_interro2d)	fait	fait	testé	'**ADV**','PRON','VERB','PROPN' **PronType=Int** **PronType=Rel**	A2
54	Règle 54 (struct_interro2e)	fait	fait	testé	'**PRON**','VERB','***PUNCT***','PRON' **PronType=Int** **PronType=Rel** ***« - »***	A2
55	Règle 55 (struct_interro2f)	fait	fait	testé	'**ADV**','PRON','VERB','***PUNCT***' **PronType=Int** **PronType=Rel** ***« ? »***	A2
56	Règle 56 (struct_interro3a)	fait	fait	testé	'**PRON**','VERB','***PUNCT***' **PronType=Int** **PronType=Rel** ***« ? »***	B1
57	Règle 57 (struct_interro3b)	fait	fait	testé	'**PRON**','AUX','VERB','***PUNCT***' **PronType=Int** **PronType=Rel**	B1

					« ? »	
58	Règle 58 (struct_interro4a)	fait	fait	testé	'SCONJ','**VERB**' **VerbForm=Inf**	B2
59	Règle 59 (struct_interro4b	fait	fait	testé	'**ADV**','ADV','ADV','VERB' **PronType=Int**	B2
60	Règle 60 (struct_interro4c)	fait	fait	testé	'PRON','VERB','PRON','**PUNCT**' **« ? »**	B2

2. Exemple de structure dans les ouvrages de FLE

Issus du B2 p 160 et 162

5.2.4.4. Constructions adjectivales : Adj Prép N/V

Adj de *N*	différent, plein	*La cour est pleine d'eau.*
Adj de *N/V*	content, fier, furieux	*Ils sont contents de venir.*
Adj à *N*	fidèle, semblable, pareil, égal, favorable, hostile	*Nous sommes favorables à ce projet.*
Adj à *N/VInf*	prêt, opposé, habitué	*Vous êtes prêt à partir ?*
Adj pour *N*	doué	*Elle est douée pour le tennis.*
Adj en *N*	expert, fort, nul	*Nul en mathématiques.*
Adj que/de *VInf*	content, triste, conscient	*Ils sont tristes que vous partiez, mais content de rester.*

5.2.4.5. Constructions pronominales

(Voir 5.1.2.2.)

VImp Pro	*Écoutez-moi.*
Pro, GN Pro V	*Lui, tout le monde le connaît.*
Pro, Pro V	*Elles, elles sont efficaces.*
Prép Pro	*Avec lui, ça va plus vite.*
C'est *Pro*	*C'est moi !*
GN Pro V	*Je lui parle.*
GN Pro V VInf	*On m'a fait entrer par là.*
GN V Pro VInf	*Je veux le voir, ce film.*
Nég Pro VImp Nég	*Ne leur dis absolument rien de tout ça.*
Nég Pro VImp Nég VInf	*Ne les laisse pas partir.*
Nég Pro VInf	*Ne pas couvrir.*
C'est ***Pro*** qui/que	*C'est moi qui ai appelé hier.*

5.2.5. Structures interrogatives

Int.	**mot interrogatif comme *qui, pourquoi…* ou *GPrép* incluant un mot interrogatif (*à quelle heure, avec qui…*)**
GN V […]/intonation/	*Tu dors ?*
Int GN V	*Où tu es ?*
Est-ce que *GN V* […]	*Est-ce que tu viens ?*
V Pro […]	*Voulez-vous un café ?*
GN V Pro	*Les secours arriveront-ils à temps ?*
Int est-ce que *GN V*	*Quand est-ce qu'on mange ?*
Int V	*Qui accepte ?*
Int V GN	*Où s'arrête ce bus ?*
V GN […] *Int*	

3. Exemple de quelques règles syntaxiques issues du script Snorkel

```
# RÈGLE 1
@labeling_function(pre=[spacy_preproc])
def avg_length(x):
    """Example based on average sentence length; sentences with more than 10 words are more difficult"""
    lengthList = [len(s) for s in x.doc.sents] # doc.sents = phrase du document
    avgLength = sum(lengthList) / len(lengthList)
    if avgLength > 13:
        return B2
    else:
        return ABSTAIN

# RÈGLE 2 : ADJECTIF B2
@labeling_function(pre=[spacy_preproc])
def complex_adjective(x):
    listPos = [i.pos_ for i in x.doc]
    #print(listPos)
    adjStructure = ['ADJ','ADJ', 'NOUN', 'ADJ', 'ADJ']
    result = [(i, i+len(adjStructure)) for i in range(len(listPos)) if listPos[i:i+len(adjStructure)] == adjStructure]
    #print(result)
    if result:
        return B2
    else:
        return ABSTAIN

# RÈGLE 3 : ADJECTIF B2
@labeling_function(pre=[spacy_preproc])
def complex_adjective1(x):
    listPos = [i.pos_ for i in x.doc]
    #print(listPos)
    adjStructure = ['ADJ','ADJ', 'NOUN', 'ADJ','CONJ', 'ADJ']
    result = [(i, i+len(adjStructure)) for i in range(len(listPos)) if listPos[i:i+len(adjStructure)] == adjStructure]
    #print(result)
    if result:
        return B2
    else:
        return ABSTAIN

# RÈGLE 4 : ADJECTIF B1
@labeling_function(pre=[spacy_preproc])
def complex_adjective2(x):
    listPos = [i.pos_ for i in x.doc]
    adjStructure = ['ADJ','ADJ','NOUN','ADJ']
    result = [(i, i+len(adjStructure)) for i in range(len(listPos)) if listPos[i:i+len(adjStructure)] == adjStructure]
    if result:
        return B1
    else:
        return ABSTAIN
```

```
# # RÈGLE 23
# # PASSIF 2
@labeling_function(pre=[spacy_preproc])
def passive_simple2b(x) :
    listText = []
    listTag = []
    for word in x.doc:
        listText.append(word.text)
        listTag.append(word.tag_)
        # ~ print(word.tag_)
    # ~ print(listTag)
    passiveStructure = ['ont', 'été']
    result = [(i, i+len(passiveStructure)) for i in range(len(listText)) if listText[i:i+len(passiveStructure)] == passiveStructure]
    if result:
        return B2
    else:
        return ABSTAIN

# # RÈGLE 24
# # PASSIF 2
@labeling_function(pre=[spacy_preproc])
def passive_simple2c(x) :
    listText = []
    listTag = []
    for word in x.doc:
        listText.append(word.text)
        listTag.append(word.tag_)
        # ~ print(word.tag_)
    # ~ print(listTag)
    passiveStructure = ['avaient', 'été']
    result = [(i, i+len(passiveStructure)) for i in range(len(listText)) if listText[i:i+len(passiveStructure)] == passiveStructure]
    if result:
      return B2
    else:
      return ABSTAIN
```