Faculté des **Langues**
Université de Strasbourg

Master Technologies des Langues

2021-2023

Ressources et outils pour l'évaluation des systèmes de simplification automatique des textes

Jessy Nierichlo

Mémoire de fin d'études

Sous la direction de
Erreur ! Signet non défini.**Mme le Professeur Amalia Todirascu**

Sommaire

I. Introduction

I.1 La simplification automatique

La simplification automatique des textes, un domaine du traitement automatique des langues, a pour but de rendre plus lisibles/accessibles des textes dits linguistiquement « complexes », tout en gardant leur contenu originel, pour ceux qui n'ont pas le bagage terminologique suffisant ou pour ceux qui ont des difficultés à comprendre la langue (personnes dyslexiques, enfants, deuxième langue, etc.) Il est important pour une personne d'avoir une capacité de compréhension écrite dans un monde où la plupart des informations sont transmises par écrit. Un exemple simple est celui de la prescription médicale, comme expliqué par (Brouwers et al., 2012), il est important que le ou la patient.e comprenne comment prendre le médicament (posologie, voie d'administration, etc.) indépendamment de sa condition (si FLE, dyslexique, enfant, etc.).

Depuis quelques années maintenant, les chercheurs ont pu constater une hausse de la fréquentation du site Wikipédia en anglais simplifié. Ce qui montre un intérêt croissant pour les textes simplifiés. Ainsi, le nombre d'articles de recherches sur la simplification automatique n'a cessé d'augmenter depuis la fin des années 90. (32 000 articles en 2021, source : Google Scholar)

La simplification automatique d'un texte peut être différenciée sous trois aspects : la simplification syntaxique, la simplification lexicale et la simplification discursive.

I.1.1 Simplification syntaxique

La simplification syntaxique aura pour but de rendre la phrase moins complexe en la coupant en plusieurs éléments en fonction des clauses à l'intérieur des phrases, ou en retirant simplement les parties trop longues des phrases selon un ensemble de règles de transformation prédéfinies. Elle vise à réduire la complexité grammaticale d'une phrase.

Comment cela fonctionne-t-il ?

Pour la simplification automatique, il est usuel de trouver un processus en trois étapes d'après (Shardlow, 2014) :

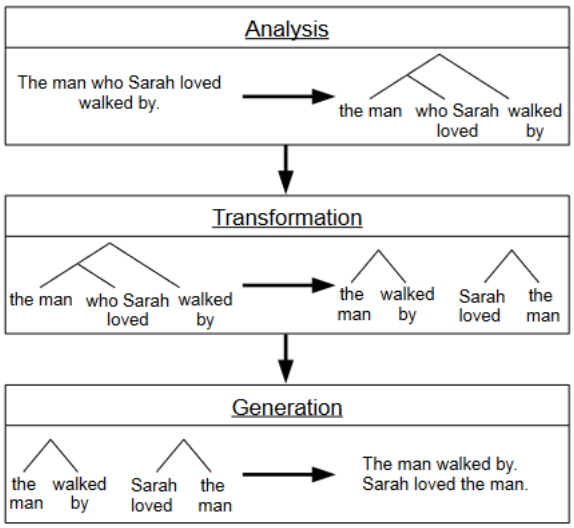


Figure 1 : Le procédé de simplification automatique Source : (Shardlow, 2014, p. 62)

D'après (Shardlow, 2014), la première étape d'analyse consiste en l'analyse de la structure syntaxique de la phrase. Ensuite, lors de l'étape de transformation, les règles préétablies permettent de couper la phrase à partir de celles-ci. Ainsi, nous nous retrouvons avec plusieurs arbres. Ces arbres sont ensuite retransformés en phrases lors de l'étape de génération.

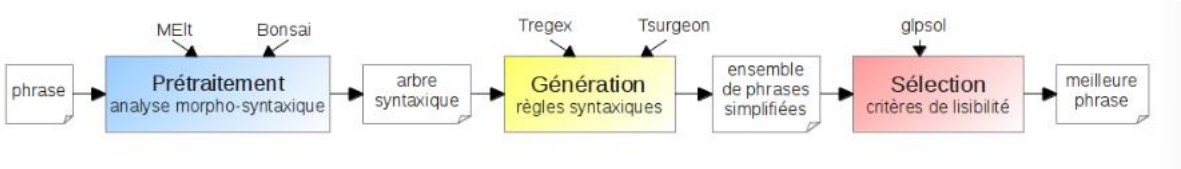
Ci-dessous, un exemple de simplification syntaxique (source : cours d'outils d'annotation, Amalia Todirascu) :

Phrase originale : « L'oreille humaine ne l'entend pas vraiment et pourtant, on perçoit "dans le fond de l'air" quelque chose de très sourd. »

Phrase simplifiée : « L'oreille humaine ne l'entend pas vraiment. Pourtant, on perçoit "dans le fond de l'air" quelque chose de très sourd. »

Dans cet exemple, nous pouvons observer que la phrase a été coupée en deux, la conjonction « et » a été supprimée dans le processus, mais cela n'enlève pas le lien logique. La phrase a donc été simplifiée syntaxiquement, mais elle reste toujours plutôt difficile à comprendre à cause des guillemets à l'intérieur de celle-ci. La simplification syntaxique peut être en effet une première étape dans la simplification.

La simplification syntaxique a déjà été étudiée pour le français dans l'article de (Brouwers et al., 2012). L'organisation de celle-ci est expliquée par le schéma suivant :



(Brouwers et al., 2012) 1 : Organisation du système de simplification syntaxique

Comme nous pouvons le constater, c'est une manière différente de procéder, différente de celle de Shardlow.

Une phrase non simplifiée est prise en entrée puis elle est annotée syntaxiquement selon le schéma de (Brouwers et al., 2012). Un arbre syntaxique de la phrase est dessiné, ce qui permet de mieux appliquer les règles syntaxiques, puisque les arbres syntaxiques mettent en valeur les liens entre les différents mots dans la phrase. L'application de chaque règle génère une phrase qui est gardée en mémoire puis elle sera sélectionnée si elle est jugée comme pertinente par le modèle de programmation linéaire d'après (Brouwers et al., 2012, p. 217). En plus de juger la

pertinence de la phrase, le modèle sélectionne la meilleure phrase en fonction des critères de lisibilité : longueur des mots, longueur de la phrase, etc.

Cependant, nous allons davantage nous intéresser à la simplification discursive puis lexicale, qui, elle, vise principalement la simplification de mots dits complexes qui sont difficiles à comprendre pour des locuteurs rencontrant des difficultés avec la langue.

I.1.2 Simplification discursive

La simplification discursive est une autre branche complémentaire de la simplification automatique de textes. Ce domaine se concentre sur la simplification des chaînes de référence (Schnedecker, 1997) qui indiquent des relations d'identité référentielles. Ce concept peut être illustré avec les relations anaphoriques ou les inférences. Deux contextes dans lesquels les enfants faibles lecteurs rencontrent des difficultés (Wilkens et Todirascu, 2020). La simplification discursive peut se limiter à remplacer les pronoms anaphoriques (« *il* » et « *elle* » par exemple) par leurs référents, ce qui aide à la compréhension globale du texte pour des enfants faibles lecteurs, voire dyslexiques (Quiniou et Daille, 2018). Néanmoins, ce n'est pas la branche de la simplification automatique sur laquelle nous allons nous concentrer. Nous allons nous interroger sur l'approche lexicale de la simplification automatique (II.1.3 Simplification lexicale) et les différentes mesures pour l'évaluer (p.8).

II.1.3 Simplification lexicale

La simplification lexicale se concentre sur la simplification d'un vocabulaire complexe. Les mots complexes sont remplacés par des synonymes plus simples. Les synonymes du mot en question seront classés du plus simple (et moins ambigu) au plus complexe (et potentiellement plus ambigu). Dans l'article de Shardlow, nous pouvons retrouver ce schéma qui présente un des processus possibles pour la simplification lexicale automatique d'une phrase : (Shardlow, 2014).

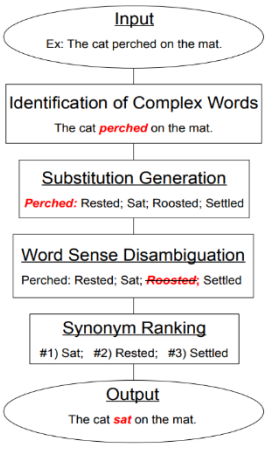


Fig. 2. The lexical simplification pipeline. Many simplifications will be made in a document concurrently. In the worked example the word 'perched' is transformed to sat. 'Roosted' is eliminated during the word sense disambiguation step as this does not fit in the context of 'cat'.

Figure 2 : (Shardlow, 2014) le cheminement de la simplification lexicale.

La simplification lexicale peut être accompagnée par un FrameNet pour comprendre un des sens possibles du mot dans un contexte et ainsi proposer un équivalent qui ferait perdre le moins de sens possible à la phrase.

D'après (Shardlow, 2014, p. 60), il est nécessaire de passer par la tâche de désambiguïsation des mots (WSD) pour pouvoir évaluer quel synonyme est le meilleur, donc quel synonyme est le plus simple et garde au mieux le sens originel.

Il est également important de se poser la question suivante dans le cadre de la simplification lexicale : qu'est-ce qu'un mot complexe ?

La lisibilité d'un texte dans le but de le simplifier est possible grâce à des formules de lisibilité comme Flesch (1948), mais elle est aussi possible si l'on détermine la difficulté des mots d'un texte.

D'après (François et al., 2016), on parle de prédiction de la difficulté lexicale ; ici, la difficulté lexicale est une valeur qui situe l'unité lexicale sur une échelle de complexité de lecture et de compréhension par rapport à des termes sémantiquement équivalents.

La complexité lexicale est liée à diverses caractéristiques du lexique (fréquence, longueur des mots, polysémie, etc.), mais elle dépend aussi du niveau de l'individu comme nous allons le voir avec FLELex.

Les mots : bleu, contusion et ecchymose ; sont tous les trois synonymes (exemple repris de [François et al., 2016]). Bleu est classifié comme simple (classe 1) dans ReSyf et les deux autres comme classe 3 donc complexe. Le problème est que nous ne pouvons pas faire la distinction entre les deux.

Par conséquent, beaucoup de critères rentrent en compte quand on essaie de déterminer la complexité d'un lemme. D'après (Gala et al., 2020 b), la régularité des graphèmes dans une unité lexicale la rend plus facile à lire. Les syllabes simples et fréquentes sont alors privilégiées au détriment des structures plus complexes (CVCC, CCVC, etc.). Les syllabes simples/complexes sont décrites dans le tableau ci-dessous (C = consonne, V= voyelle).

Difficulté	Caractéristiques, type de syllabe	Syllabe
1	Simple, fréquente	CV, V
2	Complexe, fréquente	CVC
3	Complexe, moins fréquente	CVCC, CCVC, CYVC, CYV, CCV, VCC, CVY
4	Complexe, rare	VC, YV, VCCC, CCYC, CCVC

Source : (Gala et al., 2020, p. 8)

Toujours d'après (Gala et al., 2020), la longueur des unités lexicales est intrinsèquement liée à leur fréquence et donc à leur complexité. Les syllabes simples permettent un temps de lecture généralement inférieur et des erreurs moindres lors de la lecture. De même, les mots avec suffixes/les mots d'origine latine ou grecque sont considérés comme des mots plus complexes du fait de leur fréquence plus faible.

Des tentatives de classification des mots en fonction de leur niveau ont pris forme grâce aux statistiques. FLELex (François et al., 2014), Manulex (Lété, Sprenger-Charolles et Colé, 2004), sont les deux ressources que nous allons prendre comme exemple pour définir ce qu'est un mot complexe, en nous fondant sur les statistiques lexicales pour déterminer le niveau de difficulté d'un mot.

Tout d'abord, concentrons-nous sur FLELex. D'après (Gala et Javourey-Drevet, 2020), la ressource utilise des manuels pour les apprenants FLE (*Français Langue Étrangère)* identifiés à un niveau comme corpus. Par exemple : si un mot se trouve dans un manuel pour apprenants FLE au niveau A2, le corpus sera alors *ipso facto* considéré comme A2. FLELex est doté d'une interface qui permet de visualiser la fréquence d'une unité lexicale (de A1 à C2 selon la gradation du Cadre Européen Commun de Référence pour les Langues, CECRL).

Quant à Manulex (base lexicale téléchargeable), les corpus sont identifiés à partir des niveaux scolaires (CP, CE1) ; chaque mot présent dans Manulex est associé à une fréquence estimée d'usage en fonction du niveau scolaire. Exemple : le mot « orteil » a une fréquence estimée d'usage de 39,79 au CP ; 5,26 en CE1 ; 16,40 pour fréquence globale.

Ainsi, d'après (Gala et Javourey-Drevet, 2020), les fréquences obtenues avec Manulex et le niveau obtenu avec FLELex nous permettent déjà de donner un niveau à certains mots. Donc, si un apprenant de la langue française connaît seulement des mots de niveau A1, A2, on pourrait dire qu'un mot est complexe s'il est de niveau B1 par rapport à son niveau. Il en va de même si l'on change A1, A2 par un niveau scolaire (CP, CM2, etc.).

Néanmoins, tous deux fonctionnent sous le principe de désambiguïsation grammaticale, non pas sémantique. C'est-à-dire que le mot « échelle » (objet) ne pourra pas être différencié de « échelle » (utilisé en mathématiques). Les mots polysémiques sont donc possiblement mal représentés par ces deux ressources.

La simplification lexicale peut aussi s'appuyer sur des outils comme FLAIR (Akbik et al., 2019) qui est un outil, utilisant le langage python, capable d'entraîner des modèles de langue. Son interface permet aux chercheurs de construire n'importe quel type de plongement lexical [contextuel ou non] dans une seule et même architecture ; FLAIR permet aussi de télécharger des jeux de données prêts à l'utilisation dans un cadre de recherche (Akbik et al., 2019). La simplification lexicale peut aussi s'appuyer sur BERT (Devlin et al., 2019) ainsi que ses dérivés. Tous les deux fonctionnent avec un système de vectorisation des mots. C'est-à-dire que l'algorithme détermine, en fonction du contexte, le sens le plus probable du mot (par exemple, la chaîne de caractères « Nancy » sera traitée différemment selon qu'elle désigne la ville ou un prénom). Ainsi, avec cette méthode, un vecteur est ensuite utilisé pour déterminer le mot qui est le candidat le plus probable pour le remplacement dans un contexte donné. (Shardlow, 2014.)

Nous détaillerons l'utilisation de ces ressources dans la partie consacrée à BERT.

I.2 Mesures pour l'évaluation de la simplification automatique d'un texte

La simplification automatique demande d'être évaluée par des mesures, afin de de nous permettre de nous rendre compte des changements syntaxiques ou lexicaux qu'apporte réellement la simplification du texte original. L'évaluation nous permet de mesurer la qualité de la simplification automatique.

Certains chercheurs, comme Shardlow dans sa publication de 2014, affirment que les mesures automatiques n'ont pas d'intérêt, car certaines modifications pourraient être faites pour avantager certaines mesures. (Shardlow, 2014) prend l'exemple d'une mesure qui donnerait un score haut aux phrases courtes ; alors, réduire toutes les phrases en phrases de 3-4 mots donnerait un score haut au texte même si celui-ci était difficile à comprendre.

Néanmoins, les dernières mesures comme SAMSA (Sulem, Abend, et Rappoport, 2018a) ont démontré qu'il n'était pas impossible de se rapprocher de la qualité de l'évaluation manuelle de la simplification automatique. SAMSA, comme nous le verrons dans la partie consacrée à

UCCA, est une mesure qui prend en compte les changements dans la sémantique structurelle, et ce, principalement grâce au schéma UCCA. La prise en compte des changements dans la sémantique structurelle était une première pour une mesure consacrée à l'évaluation de la simplification automatique.

Les prochaines mesures présentées sont seulement automatiques, c'est-à-dire qu'elles compareront automatiquement à l'aide d'un corpus de bonne qualité référencé manuellement par un humain. Il est nécessaire de préciser que les mesures présentées sont plutôt spécifiques, c'est-à-dire qu'elles fonctionnent bien dans certains domaines (tels que l'évaluation de la traduction automatique dans le cas de BLEU), mais possèdent des lacunes dans d'autres domaines, comme la simplification automatique (toujours dans le cas de BLEU).

I.2.1 BLEU

BLEU (Papineni et al., 2002) est présenté par ses fondateurs comme un indicateur numérique de la « proximité » de la traduction grâce à un corpus de traductions de bonne qualité référencées manuellement.

D'après les mêmes auteurs (*ibidem*) BLEU fonctionne de la manière suivante : à partir d'un corpus de traductions de référence, il va comparer les n-grammes d'une traduction dite « candidate » avec les n-grammes d'une traduction de référence. À partir de là, l'indicateur comptera les concordances entre les deux. Ainsi, plus la traduction candidate compte de concordances avec la traduction référence, mieux elle sera notée.

L'indicateur BLEU s'étend de 0 à 1. Force est de constater que peu de traductions atteindront un score de 1, à moins qu'elles ne soient identiques à une traduction de référence (ce qui est, évidemment, plutôt rare). Pour cette raison, même un traducteur humain n'obtiendra pas toujours la note de 1. BLEU est donc une mesure pour évaluer la traduction automatique d'un texte, mais il n'est pas rare de la trouver dans d'autres contextes, qui eux ne lui correspondent pas autant. Néanmoins, d'autres mesures sont plus adaptées à cette tâche, comme SARI (Xu et al., 2016).

D'après (Sulem, Abend, et Rappoport, 2018 b), la mesure BLEU ne convient pas à l'évaluation de la division des phrases, la principale opération de simplification structurelle.

Ainsi, BLEU ne réussirait pas à prédire la simplicité d'une phrase d'après (Sulem, Abend, et Rappoport, 2018 b) et aurait tendance à donner des scores hauts aux phrases ressemblant à la phrase originale, ce qui est contreproductif dans le cas d'une simplification. On veut que la phrase soit différente, plus compréhensible.

Mais, il obtiendrait une plus haute corrélation dans la grammaticalité et dans la préservation du sens (voir SARI [Xu et al., 2016]) ; ce qui confirme bien l'hypothèse que chaque mesure est « dédiée » à l'évaluation d'un domaine spécifique de la simplification automatique.

Il en est conclu que BLEU n'est pas pertinent, la mesure est même qualifiée de trompeuse (Sulem, Abend, et Rappoport, 2018 b) dans l'évaluation de la simplification automatique. Les

tâches pour lesquelles la mesure fonctionne correctement sont l'évaluation de la division et la reformulation automatique d'une phrase.

Les mesures sont-elles complémentaires, c'est-à-dire qu'elles évaluent toutes correctement un domaine, ainsi pour avoir une évaluation globale d'une simplification, faudrait-il utiliser chacune des mesures ?

I.2.2 SARI (Xu et al., 2016)

SARI est une mesure crée par (Xu et al., 2016). SARI a été conçue spécialement pour la simplification automatique de textes, à la différence de BLEU. SARI est une mesure monolingue, qui se concentre sur une seule langue à la fois. L'objectif principal de cette mesure est d'améliorer l'évaluation de la simplification lexicale d'une phrase et, dans une moindre mesure l'évaluation de la simplification syntaxique.

Les nouveaux indicateurs créés (FKBLEU, SARI) par les chercheurs montrent une corrélation bien plus forte avec les notes de simplicité attribuées par les humains sans pour autant négliger la grammaticalité tout en préservant le sens de la phrase.

Leur mesure SARI (Xu et al., 2016) présente la corrélation la plus élevée avec les jugements humains de simplicité, mais l'indicateur BLEU présente des corrélations plus élevées sur la grammaticalité et le maintien du sens. Les critères de simplicité peuvent se résumer à : la longueur des phrases/mots (plus la phrase/mot est longue, plus elle est difficile à comprendre) ; la complexité du vocabulaire ; complexité syntaxique ; présence d'anaphores ; présence de constructions au passif ; utilisation d'abréviations ; etc. Pour maintenir au mieux le sens de la phrase lors de la simplification, il faut choisir un synonyme du mot complexe qui enlève le moins de sens à la phrase (de même pour la construction), même s'il en enlèvera toujours un peu.

Exemple :

Phrase originale : La file d'attente est interminable.

Phrase simplifiée 1 : La file d'attente est longue.

Phrase simplifiée 2 : La file d'attente est sans fin.

Ici, la phrase simplifiée 2 enlève le moins de sens à la phrase originale. La locution adverbiale « sans fin » permet de garder le jugement du locuteur. Il est exaspéré par la longueur de la file d'attente. La phrase simplifiée 1 est plus objective, on enlève le jugement du locuteur. On dit simplement que la file est longue, mais aucun sentiment d'exaspération n'en ressort, contrairement à l'adjectif « interminable » qui donne ce sentiment. La grammaticalité de la phrase est un des autres critères concernant la simplicité d'une phrase, car un texte sera *de facto* plus complexe s'il comporte des erreurs dans la grammaire. Ainsi, pour simplifier le texte, il faut se débarrasser des erreurs de grammaire.

Voici quelques recommandations selon (Saggion, 2017) pour rédiger un texte « simple » :

- use simple and direct language;
- use one idea per sentence;
- avoid jargon and technical terms;
- avoid abbreviations;
- structure text in a clear and coherent way;
- use one word per concept;
- use personalization; and
- use active voice.

Contrairement aux précédentes mesures, le corpus utilisé pour les expériences de la mesure SARI n'est pas la version simplifiée de Wikipédia. (Xu et al., 2016) n'ont pas choisi d'utiliser ce corpus, car il s'avère qu'il possède beaucoup de simplifications erronées qui ne diffèrent pas de la phrase originale. Ainsi, comme l'expliquent (Xu et al., 2016), ces simplifications erronées seraient la cause du dysfonctionnement [ou de l'incapacité à évaluer correctement les simplifications automatiques] de la part des précédentes mesures.

SARI entend inventer une nouvelle manière d'évaluer les simplifications automatiques, une manière qui corrèle plus avec le jugement humain.

Comment fonctionne-t-elle ?

SARI compare les données en sortie de simplification avec les données de référence ET les données du texte source. La mesure sera à même de juger si les mots ajoutés par l'outil de simplification automatique sont « bons », de même pour les suppressions effectuées par l'outil.

Contrairement à la plupart des autres mesures, SARI récompense les ajouts dans le score. C'est-à-dire que si un mot est ajouté dans la phrase de sortie, par rapport à la phrase source, SARI récompensera cet ajout dans le score s'il est jugé pertinent.

La formule suivante explique son fonctionnement :

$$p_{add}(n) = \frac{\sum_{g \in O} \min\left(\#_g(O \cap \bar{I}), \#_g(R)\right)}{\sum_{g \in O} \#_g(O \cap \bar{I})}$$

$$r_{add}(n) = \frac{\sum_{g \in O} \min\left(\#_g(O \cap \bar{I}), \#_g(R)\right)}{\sum_{g \in O} \#_g(R \cap \bar{I})}$$

(Xu et al., 2016)

Les formules P_{add} (n) et T_{add} (n), selon (Xu et al., 2016), expliquent comment la mesure SARI « récompense » l'introduction de nouveaux mots pertinents. Le *O* représente la sortie (*output*) ; la phrase source correspond à *I* (*input)* et *R* correspond aux phrases de référence. Pour finir, #g est un indicateur binaire qui indique s'il y a une occurrence du *n*-Gram g dans un set donné.

La mesure SARI récompense également quand la phrase de sortie garde des mots qui sont gardés dans les phrases références. Cette méthode de récompense est expliquée par les formules P_{keep} (n) et T_{keep} (n) de (Xu et al., 2016) :

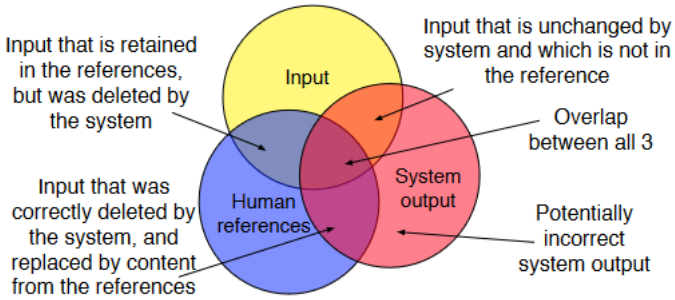
$$p_{keep}(n) = \frac{\sum_{g \in I} \min\left(\#_g(I \cap O), \#_g(I \cap R')\right)}{\sum_{g \in I} \#_g(I \cap O)}$$

$$r_{keep}(n) = \frac{\sum_{g \in I} \min\left(\#_g(I \cap O), \#_g(I \cap R')\right)}{\sum_{g \in I} \#_g(I \cap R')}$$

where

$$\#_g(I \cap O) = \min\left(\#_g(I), \#_g(O)\right)$$
$$\#_g(I \cap R') = \min\left(\#_g(I), \#_g(R)/r\right)$$

(Xu et al., 2016)



(Xu et al., 2016, p.404).

L'intersection entre les trois cercles est l'optimum, ce qui est recherché. SARI compare la sortie avec les phrases de référence ainsi que l'entrée. Une phrase obtiendra un score haut si elle se situe dans l'intersection entre les trois cercles. Nous pouvons noter une différence : la mesure SARI compare la sortie avec les phrases références ET l'entrée, ce que ne font pas les mesures qui évaluent la traduction automatique, étant donné qu'ici l'entrée et la sortie sont dans la même langue contrairement à la TA.

Ensuite, SARI a également une formule pour la suppression, appelée P_{del} (n) :

$$p_{del}(n) = \frac{\sum_{g \in I} \min\left(\#_g(I \cap \overline{O}), \#_g(I \cap \overline{R'})\right)}{\sum_{g \in I} \#_g(I \cap \overline{O})}$$

where

$$\#_g(I \cap \overline{O}) = \max\left(\#_g(I) - \#_g(O), 0\right)$$
$$\#_g(I \cap \overline{R'}) = \max\left(\#_g(I) - \#_g(R)/r, 0\right)$$

(Xu et al., 2016)

Peu de suppressions sont effectuées parce qu'il vaut mieux, pour le sens de la phrase, ne pas en faire que trop en faire.

Ainsi, SARI évalue la simplicité d'une phrase en combinant ces trois formules :

$SARI = d_1F_{add} + d_2F_{keep} + d_3P_{del}$

where $d_1 = d_2 = d_3 = 1/3$ and

$$P_{operation} = \frac{1}{k} \sum_{n=[1,...,k]} p_{operation}(n)$$

$$R_{operation} = \frac{1}{k} \sum_{n=[1,...,k]} r_{operation}(n)$$

$$F_{operation} = \frac{2 \times P_{operation} \times R_{operation}}{P_{operation} + R_{operation}}$$

$$operation \in [del, keep, add]$$

where k is the highest n-gram order and set to 4 in our experiments.

Formule provenant de (Xu et al., 2016, p.405)

Comme le démontrent (Xu et al., 2016), BLEU (Papineni et al., 2002) reste une bonne mesure dans les domaines liés à la traduction automatique donc dans la grammaire et la préservation du sens. Néanmoins, c'est dans le cas de l'évaluation de la simplification automatique que la mesure BLEU n'arrive pas à rejoindre le jugement humain contrairement à la mesure SARI. SARI obtient un score de 0,343 pour la simplicité (Xu et al., 2016, p. 410) comparé au score de BLEU qui est de 0,151. Ces scores ont été obtenus sur les corpus MechanicalTurk (contributions anonymes afin de simplifier le corpus) ; Wikipédia et Wikipédia simplifié. Ce score représente les corrélations entre le jugement humain et celui de la mesure. Nous pouvons expliquer ces corrélations par le fait que SARI est directement conçue pour évaluer la simplification automatique à partir de techniques héritées de l'évaluation de la simplification automatique.

SARI pénalise les modèles qui ne font pas assez de changement.

I.2.3 FKBLEU

FKBLEU (Xu et al., 2016) mesure la lisibilité d'un texte et est complémentaire avec SARI. Prendre en compte les deux indicateurs permet d'avoir une meilleure idée de la qualité de la simplification, car les deux indicateurs ont des critères différents.

FKBLEU est un mélange de la mesure iBLEU (Sun et Zhou, 2012) qui est connue pour l'évaluation de la production de paraphrase ; avec l'indicateur de lisibilité Flesch-Kincaid [FK] (Kincaid et al., 1975).

D'après (Xu et al., 2016), plus le score de FKBLEU est haut, plus la simplification automatique d'un texte est jugée lisible et plus la simplification est jugée comme pertinente ou bonne [grâce à la combinaison de FK et iBLEU].

La formule pour calculer FKBLEU est la suivante :

D'abord, il faut calculer iBLEU [*I, R, O* ont la même signification que dans la partie SARI] :

iBLEU = α × BLEU (*O*, *R*) — (1 — α) × BLEU (*O*, *I*).

Formule tirée de (Xu et al., 2016, p. 403)

Ensuite, il est nécessaire de calculer FK :

$$FK = \begin{matrix} 0{,}39 \times \left(\frac{words}{sentences}\right) \\ +11{,}8 \times \left(\frac{syllables}{words}\right) - 15{,}59 \end{matrix}$$ (Kincaid et al., 1975)

Pour ainsi calculer FKBLEU, formule tirée de (Xu et al., 2016, p. 403) [FKdiff calcule la différence entre la phrase *O* et *I*] :

FKBLEU = iBLEU (I, R, O) × FKdiff (I, O)

FKdiff = sigmoid (FK(O) − FK [I])

Néanmoins, il est nécessaire de préciser que les formules ci-dessus sont uniquement pour l'anglais car l'indicateur Flesch Kincaid est dépendant de l'anglais. On pourrait donc dire que FKBLEU n'est pas utilisable pour le français, dans sa formule actuelle.

Pour conclure, nous avons étudié un ensemble de mesures automatiques (BLEU, SARI et FKBLEU). Ces mesures ont montré leurs faiblesses ainsi que les domaines dans lesquels elles fonctionnaient le mieux. SARI (Xu et al., 2016) s'est révélée être la mesure la plus probante, celle qui se rapproche le plus du jugement humain lors de l'évaluation de la simplicité d'une phrase. L'évaluation de la simplicité d'une phrase est un exercice difficile tant il est subjectif. Un annotateur *X* peut trouver que la phrase simplifiée *x* est plus simple que la phrase simplifiée *y* alors qu'un annotateur *Y* peut penser l'inverse. C'est pour cela qu'un nombre de règles est imposé aux annotateurs, pour essayer de neutraliser ces différences subjectives.

Après avoir vu les différentes mesures évaluant la simplicité d'une phrase, nous allons maintenant étudier les outils permettant l'annotation sémantique, ainsi qu'une autre mesure d'évaluation de la simplicité qui fonctionne grâce à l'outil d'annotation UCCA.

II. Outils pour l'annotation sémantique

II.1 UCCA : Universal Cognitive Conceptual Annotation

UCCA est un outil d'annotation sémantique qui vise à prendre en compte les distinctions sémantiques. C'est un schéma d'annotation qui s'appuie sur la théorie linguistique typologique et linguistique BLT : Basic Linguistic Theory par (Dixon, 2009). UCCA peut être utilisé pour annoter le français et plusieurs autres langues, mais a été seulement testé sur l'anglais dans l'article de présentation. L'article (Sulem, Abend et Rappoport, 2015) se concentre sur la portabilité de l'UCCA sur la paire de langues anglais-français et sa stabilité ; c'est-à-dire sa capacité à préserver la structure à travers les traductions.

Quelle serait la potentielle application d'UCCA ?

Grâce à sa dimension sémantique, l'outil permet d'améliorer le fonctionnement des applications sémantiques telles que les applications de type questions-réponses. UCCA permet de montrer les différences sémantiques entre les phrases, ce qui pourrait être utile dans ce genre d'applications ; jusqu'alors seules les distinctions syntaxiques étaient prises en compte dans les applications de questions/réponses (par exemple) alors que celles-ci ne peuvent pas faire la différence entre deux phrases au sens quasi identique.

De plus (Abend et Rappoport, 2013), démontrent dans leur article que l'outil est facilement adaptable et accessible. Même les annotateurs et annotatrices n'ayant pas d'expérience en linguistique rattrapent rapidement — dans la qualité des annotations — les annotateurs et annotatrices dotés d'une expérience linguistique.

La lettre U, dans le sigle UCCA, fait référence au mot « universel » qui lui-même fait référence à la capacité d'UCCA de s'adapter à un ensemble très riche de distinctions sémantiques et à sa capacité de fournir toutes les informations sémantiques nécessaires à l'apprentissage de la grammaire.

L'approche d'UCCA se fonde sur une approche typologique de la grammaire BLT (Basic Linguistic Theory [Dixon, 2005 ; 2010 a ; 2010 b ; 2012] cité par [Abend et Rappoport, 2013]), ce qui permet la description d'une grande variété de langues.
Plus précisément, l'approche linguistique "BLT" explore les structures fondamentales du langage humain, transcendant les spécificités linguistiques. Elle s'intéresse aux liens entre formes grammaticales et à la manière dont les éléments se combinent pour créer des énoncés. Tout en recherchant des traits universels, cette approche reconnaît également les différences inhérentes à chaque langue. Elle a été conçue par le linguiste R. M. W. Dixon pour analyser une variété de langues à travers des catégories fondamentales et des relations syntaxiques. Cette théorie contribue à une compréhension approfondie du langage, en harmonisant l'universalité et la diversité linguistique.

De même (Abend et Rappoport, 2013), ont démontré qu'UCCA était très insensible à la variation de la syntaxe d'une phrase — et donc insensible à la paraphrase. Ce qui est utile pour

la simplification automatique dans le sens où l'outil se concentre seulement sur la sémantique. Dans les exemples donnés par les chercheurs, différentes phrases qui sont en anglais sont montrées puis traduites dans deux autres langues. Chaque fois, le résultat est le même : bien que la syntaxe soit différente, la sémantique ne l'est pas. Il y a toujours un participant quand il n'y en a qu'un dans la phrase source, toujours une négation dans la phrase cible quand il y en a une dans la phrase source, etc. Son mode de fonctionnement est comparable à FrameNet même si UCCA est un peu moins précis que FrameNet [voir [Baker et al., 1998] pour FrameNet]. UCCA privilégie une annotation qui couvre plus de textes que les annotations de FrameNet. Un corpus de textes dits « naturels » annoté avec UCCA est fourni contrairement à FrameNet d'après [Abend et Rappoport, 2013, p. 236].

Comment fonctionne réellement UCCA ?

Premièrement, on peut distinguer plusieurs couches. En premier lieu, nous allons nous concentrer sur la couche dite « élémentaire » qui constitue la majorité de la première analyse du texte.

Le texte est vu, par la couche élémentaire, comme une collection de scènes. D'après [Abend et Rappoport, 2013, p. 229], chaque scène peut représenter une action, un mouvement, ou un état temporellement persistant. Les scènes ont généralement une dimension temporelle et spatiale, qui peuvent être spécifiques à un temps et à un lieu particulier, mais peuvent aussi décrire un état schématisé particulier, un événement qui fait référence à de nombreux événements en mettant en évidence une composante de sens commun. Une phrase peut contenir plusieurs scènes et évidemment plusieurs relations dans ces scènes, mais aussi entre les scènes.

Qu'est-ce qu'une relation ? Il existe des relations de dépendance entre les différents syntagmes d'une phrase. Par exemple, quelle relation entretient le participant « Baptiste » avec le verbe « prendre » dans le contexte « Baptiste prend sa douche » ?

Baptiste est le sujet du verbe « prend ». Le sujet « Baptiste » et le verbe « prendre » sont donc en relation, et UCCA pourrait leur consacrer une scène.

La richesse des distinctions sémantiques d'UCCA permet aux unités de participer à plus d'une relation, c'est ce qui le différencie des autres outils. UCCA est conçu pour s'adapter.

D'après (Abend et Rappoport, 2013), UCCA préconise une approche qui traite la syntaxe comme une couche cachée lors de l'apprentissage de la correspondance entre la forme, le langage et le sens, alors que les approches syntaxiques existantes visent à la modéliser manuellement et explicitement. Se détacher de la syntaxe lui permet d'être quasiment insensible à la paraphrase. Même si les deux phrases ont une syntaxe différente, UCCA les traitera comme une seule et même phrase puisqu'elles véhiculent le même sens.

Le schéma UCCA fonctionne avec des graphes orientés acycliques qui sont des graphes orientés sans circuits.

D'après [Abend et Rappoport, 2013], p229-230], ces graphes sont très utiles pour la représentation des mots dans la phrase. Ici, ils sont utilisés pour représenter les structures sémantiques. Chaque nœud est représenté par une unité, qui peut soit être un terminal, soit

plusieurs éléments regroupés en une seule entité en fonction de leur considération sémantique ou cognitive.

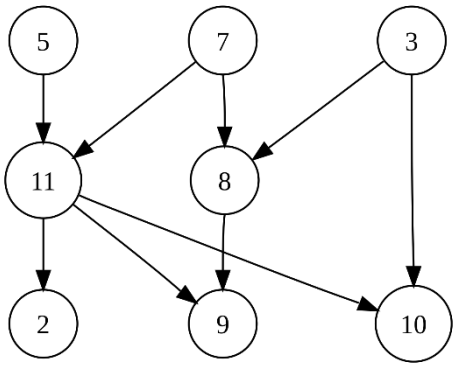
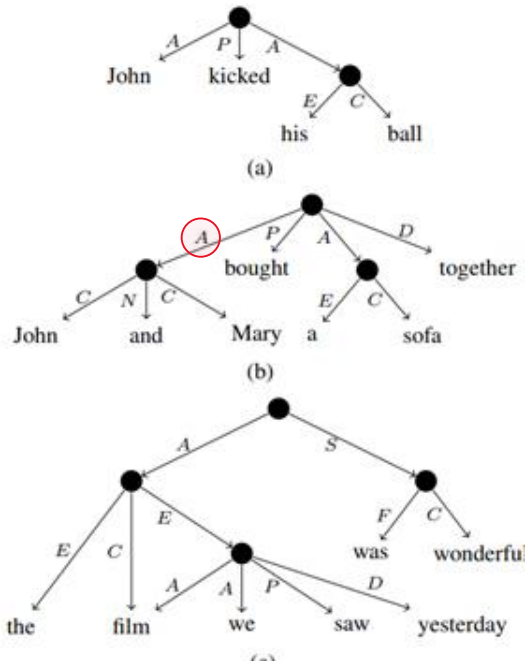


Figure 3 : https://upload.wikimedia.org/wikipedia/commons/thumb/0/03/Directed_acyclic_graph_2.svg/1280px-Directed_acyclic_graph_2.svg.png

Exemple de graphe orienté sans circuit. Ce genre de graphes permet une représentation par niveaux et également une meilleure lisibilité avec les numéros.

Une des caractéristiques de ces graphes est qu'ils permettent une seule et unique interprétation sémantique du texte. Il peut alors découler différentes annotations d'une même phrase. C'est également pour ça que calculer l'accord inter-annotateur sur un corpus annoté par différents annotateurs n'est pas simple.



Chaque trait représente une relation et à côté de chaque trait nous pouvons voir une lettre. Chaque lettre désigne une catégorie en particulier (cf. Tableau 1).

Figure 4 : exemple annoté de schéma UCCA

(Abend et Rappoport, 2013, p. 231)

Abb.	Category	Short Definition
	Scene Elements	
P	**Process**	The main relation of a Scene that evolves in time (usually an action or movement).
S	**State**	The main relation of a Scene that does not evolve in time.
A	**Participant**	A participant in a Scene in a broad sense (including locations, abstract entities and Scenes serving as arguments).
D	**Adverbial**	A secondary relation in a Scene (including temporal relations).
	Elements of Non-Scene Units	
C	**Center**	Necessary for the conceptualization of the parent unit.
E	**Elaborator**	A non-Scene relation which applies to a single Center.
N	**Connector**	A non-Scene relation which applies to two or more Centers, highlighting a common feature.
R	**Relator**	All other types of non-Scene relations. Two varieties: (1) Rs that relate a C to some super-ordinate relation, and (2) Rs that relate two Cs pertaining to different aspects of the parent unit.
	Inter-Scene Relations	
H	**Parallel Scene**	A Scene linked to other Scenes by regular linkage (e.g., temporal, logical, purposive).
L	**Linker**	A relation between two or more Hs (e.g., "when", "if", "in order to").
G	**Ground**	A relation between the speech event and the uttered Scene (e.g., "surprisingly", "in my opinion").
	Other	
F	**Function**	Does not introduce a relation or participant. Required by the structural pattern it appears in.

Tableau 1 : Set complet des catégories dans la couche élémentaire de l'UCCA (Abend et Rappoport, 2013, p. 230)

Les exemples plus haut (cf. Figure 4 ; p.17) montrent des scènes dites « simples » qui ne contiennent qu'une relation principale. À l'intérieur de ces scènes simples, nous pouvons distinguer deux types de scènes : les scènes statiques (représentées par le « S »), pour les états permanents.

Ex. : Baptiste est incroyable. (S)

Et les scènes montrant un processus (représentées par le « P »).

Ex. : Baptiste prend sa douche. (P)

Les éléments « C » qui renvoient à « centre » constituent une partie importante de la phrase au niveau sémantique. Sans cet élément C, il est difficile de comprendre le type sémantique de la phrase, c'est-à-dire la nature de sa dénotation (si le sens est saturé ou non).

Comme nous l'avons déjà vu, UCCA a été pensé pour s'adapter. Ici, nous pouvons le constater encore une fois puisqu'une unité peut participer à plus d'une scène (cf. Figure 4, p.17 ; troisième exemple).

Ex. : Baptiste a demandé à Marie de le rejoindre. (Abend et Rappoport, 2013, p. 231)

Marie participe à la fois dans la scène « a demandé » et « le rejoindre ».

UCCA annote également les relations des scènes entre elles (cf. Figure 4, p.*17* ; entourée en rouge). Quelle est l'utilité pour UCCA d'avoir plusieurs couches ?

Les autres couches de l'UCCA servent à affiner le travail effectué par la couche élémentaire. La catégorisation d'une relation est un exemple d'affinage de la part des autres couches. Les chercheurs ont également montré que ces couches pouvaient étendre le graphe d'annotations, en ajoutant des relations entre des unités déjà définies, en peaufinant celles qui existent déjà ou en ajoutant des unités intermédiaires entre l'unité parent et ses descendants. Ils donnent

l'exemple « gave up ». D'après (Abend et Rappoport, 2013, p. 232) si la fonction d'une couche donnée est d'annoter le sens, alors celle-ci cassera l'expression en deux. Elle se retrouvera alors avec « gave » qui sera l'unité porteuse du temps [de la scène].

Après avoir vu une partie de la théorie autour de l'UCCA, il est nécessaire de voir un cas pratique. Nous allons nous concentrer sur SAMSA : une mesure se fondant sur le schéma UCCA.

II.1.1 SAMSA (Sulem, Abend, et Rappoport, 2018a) : un exemple d'application de l'UCCA

SAMSA (Sulem, Abend, et Rappoport, 2018a) est une mesure qui ne se concentre pas seulement sur les aspects lexicaux, mais principalement sur l'évaluation de l'aspect structurel de la phrase. C'est ce qui la différencie de beaucoup d'autres mesures, notamment de celles abordées dans la partie II. 2 (p. 8).

Dans l'article de (Sulem, Abend, et Rappoport, 2018a), les chercheurs comparent le fonctionnement de la mesure SARI (Xu et al., 2016) qu'ils décrivent comme un grand pas en avant dans l'évaluation de la simplification automatique. Toutefois, la mesure SARI a ses limites : elle se concentre sur l'évaluation de la simplification au niveau lexical, au détriment de l'évaluation de l'aspect structurel d'une simplification.

SAMSA est la première mesure à s'intéresser à la sémantique structurelle, qui est l'étude des relations de sens. Ce qui est en concordance directe avec le schéma UCCA et explique pourquoi la mesure se fonde presque entièrement sur le schéma.

SAMSA utilise UCCA pour définir les structures sémantiques, car l'efficacité du schéma a été démontrée de nombreuses fois dans le domaine de la traduction automatique (Sulem, Abend, et Rappoport, 2015) ; il peut donc très bien s'appliquer à la simplification automatique, un domaine voisin. Elle intègre UCCA grâce à l'analyseur multilingue TUPA (Hershcovich, Abend, et Rappoport, 2017) qui permet la représentation sémantique.

Le fonctionnement de la mesure est simple : un score bas sera donné par SAMSA dans les cas où la relation dans la phrase source a été altérée par la simplification.

Ex. 1 :

** phrase source : Baptiste a mangé et a lavé la vaisselle.

** phrase simplifiée 1 : Baptiste a mangé. Baptiste a lavé la vaisselle. (Score élevé attribué par SAMSA)

** phrase simplifiée 2 : *Baptiste a mangé et a lavé. La vaisselle a été lavée. (Score bas attribué par SAMSA ; agrammaticale, verbe transitif direct qui n'a pas de complément)

La mesure SAMSA est appliquée dans des corpus annotés automatiquement et dans des corpus annotés manuellement : EventS (phrases simplifiées automatiquement par le système EventSimplify TS (Glavaš et Štajner, 2015) provenant du journal News-Brief ; EncBrit :

phrases originales et simplifiées automatiquement par le système ATS (Štajner, Béchara, et Saggion, 2015) provenant de Encyclopædia Britannica (Barzilay and Elhadad, 2003) ; LSLight : composé de phrases provenant de Wikipédia dans sa version anglophone puis les versions simplifiées des phrases par différents systèmes de simplifications automatiques.

Les résultats obtenus par SAMSA sont encourageants : il existe une corrélation élevée avec le jugement humain, ce qui est un avantage contrairement aux autres mesures qui peinent à prévoir le jugement humain lorsque de la simplification structurelle a été effectuée sur la phrase source.

La combinaison de l'évaluation de plusieurs tâches inhérentes à la simplification automatique est un atout clé pour une mesure d'évaluation. D'après (Xu et al., 2016), les mesures de 2016 peinaient à évaluer les ressources sur différentes tâches en même temps, et donc la qualité de leur évaluation en souffrait. De plus, les mesures étaient souvent importées de l'évaluation de la traduction automatique, elles n'étaient donc pas entièrement consacrées à évaluer toutes les tâches de la simplification automatique. Une exception est SARI, qui a été entièrement conçue pour l'évaluation de la simplification automatique.

Avant SAMSA, et même SARI, il n'y avait pas une mesure qui sortait du lot. Certaines comme SARI et FKBLEU (Xu et al., 2016) s'en sortaient mieux dans les jugements de la simplicité, car elles obtenaient une corrélation forte avec le jugement humain, mais se faisaient complètement dépasser par BLEU dans les autres domaines, comme ceux de la préservation du sens. En effet, BLEU est avant tout une mesure pour l'évaluation de la traduction automatique (Sulem et al., 2018).

SAMSA est donc unique dans le domaine, car elle prend en compte l'aspect structurel de la phrase. De plus, les mesures qui prennent en compte la structure de la phrase ont tendance à avoir une corrélation élevée avec le jugement humain.

Nous pouvons citer un équivalent de SAMSA : HUME [Human UCCA-based Evaluation of Machine Translation] (Birch et al., 2016). Cette mesure se fonde aussi sur le schéma UCCA, la différence avec SAMSA est que l'évaluation est manuelle.

SAMSA change de ce qui a été fait auparavant. Dans le passé, les structures syntaxiques étaient davantage étudiées contrairement aux structures sémantiques.

Les découpages incohérents, comme nous l'avons vu dans l'exemple plus haut (cf. phrase simplifiée 2), sont identifiés par SAMSA, et reçoivent un score bas pour conséquence.

Dans une phrase source [input] contenant traditionnellement un événement, il est nécessaire que la phrase simplifiée [output] contienne un seul événement, pas plus ; traditionnellement, quand on parle d'événement, on fait référence au schéma UCCA, et à sa définition d'événement (cf. tableau 1 ; partie UCCA). Dans le cas où la phrase simplifiée contiendrait deux événements, SAMSA le détecterait grâce au schéma UCCA et ainsi sanctionnerait le score de la simplification.

Cependant, on évitera la référence à plusieurs événements au sein de la même phrase (cf. ex1 : phrase source, phrase simplifiée 1), sinon le score sera pénalisé, car la phrase sera jugée comme plus complexe.

À travers cette partie consacrée à l'étude du schéma UCCA et de son fonctionnement, ainsi qu'à l'étude d'une application du schéma à travers la mesure SAMSA, nous pouvons tirer quelques conclusions. UCCA est polyvalent. Il peut s'appliquer à plusieurs paires de langues, sans trop perdre en efficacité. Grâce à SAMSA, nous avons pu observer que le schéma pouvait être utilisé de manière automatique pour l'évaluation de la simplification automatique d'une phrase.

Par conséquent, il n'est pas inconcevable d'utiliser le schéma UCCA, à travers la mesure SAMSA, pour l'évaluation de la simplification automatique des textes en français. Nous allons voir dans la partie suivante le projet ASFALDA, une ressource pour le français ; ressource qui peut être utile dans le cas où UCCA ne serait pas totalement compatible pour l'évaluation de la simplification automatique des textes en français.

II.2 FrameNet

FrameNet est une ressource électronique se fondant sur des *cadres* (représentations schématiques) qui permettent de prédire des cadres sémantiques possibles. Ces *cadres* sont accompagnés par des FEs (*frame elements ; c.-à-d. les participants aux situations)* Cette ressource a été développée pour l'anglais d'abord (Baker, Fillmore, et Lowe, 1998) et pour d'autres langues.

Il existe un FrameNet pour le français depuis la mise en œuvre du projet ASFALDA (Djemaa et al., 2016) qui s'est terminé en 2016.

Un FrameNet français serait utile pour l'analyse sémantique automatique des phrases. En effet, il permet d'extraire le sens global d'une phrase donnée et peut même le comparer à d'autres phrases similaires ayant d'autres perspectives.

Les FEs permettent de rajouter une nouvelle dimension sémantique aux *cadres,* en décrivant les participants, le contexte, l'endroit, etc. Les FEs peuvent également contenir des éléments syntaxiques, qui décrivent le rôle des participants dans la phrase.

EXPORTING		
Def:	An *Exporter* moves *Goods* across a border from an *Exporting_area* to an *Importing_area.*	
Roles:	*Exporter*	The conscious entity, generally a person, that moves the *Goods* across a border out of the *Exporting_area*
	Exporting_area	The place where the *Goods* are initially, before the the *Exporter* moves them.
	Goods	The items of value whose location is changing.
	Importing_area	The place that the *Goods* end up as a result of motion.
FEEs:	export.n, export.v, exportation.n	

Figure 5 : frame pour le verbe anglais « exporting »

Source : (Djemaa et al., 2016, p. 3795)

D’après (Djemaa et al., 2016), il existe également des relations entre les cadres. Ces relations permettent de définir des relations d’identité entre les rôles de deux cadres qui sont liées.

Comme nous l’avons précisé précédemment, il existe un FrameNet dédié au français qui a été développé sous le nom de “projet ASFALDA”. Il a donc été intéressant d’étudier brièvement le fonctionnement du FrameNet référence pour l’anglais pour pouvoir ensuite étudier celui dédié au français. Ainsi, dans la sous-partie suivante, nous allons étudier ce FrameNet en profondeur afin de mettre en lumière les différentes caractéristiques de ce FrameNet.

II.3 Le projet ASFALDA

Le but du projet ASFALDA (Candito et al., 2014) était d’organiser un FrameNet et un lexique pour le français. Les unités à l’intérieur de ce lexique seront associées à des cadres dans le FrameNet créé dans le cadre du projet ASFALDA (exemple : Figure 5). Un FrameNet est aussi accompagné par des unités lexicales, qui elles correspondent à des tokens (mots, groupe de mots, etc.). Chaque mot est associé à un contexte. Chaque cadre possède une phrase qui met le mot dans son contexte.

Ainsi, avoir un FrameNet pour le français permettrait de développer des applications pour l’extraction d’informations ou même des applications pour annoter sémantiquement le contenu d’un corpus.

Le FrameNet a été annoté grâce aux corpus français : « Sequoia Treebank » et « the French Treebank » et a pour base le FrameNet anglais (Baker et al., 1998) ; il reprend donc sa structure et on peut se demander si la structure anglaise est adaptable au français.

Le projet ASFALDA avait également pour but de développer un analyseur sémantique pour le français. Ayant pour base FrameNet, il a été choisi pour son orientation un peu plus sémantique que les autres bases comme PropBank (Kingsbury et Palmer, 2002).

Il est également nécessaire de noter que les caractéristiques syntaxiques d’un cadre ne sont pas pour autant négligeables, car ce sont des indices essentiels pour prédire les cadres sémantiques et FEs (frame éléments : participants des situations.)

(Candito et al., 2014) ont préféré se contenter de quelques domaines afin de rendre l’annotation manuelle plus simple et moins coûteuse dans le cadre de l’annotation des deux corpus français cités plus haut. Ils ont donc opté pour une annotation domaine par domaine. Il y avait d’autres possibilités comme l’annotation lemme par lemme, mais elle est beaucoup plus coûteuse et difficile pour les annotateurs ; de même, ils auraient pu adopter une approche cadre par cadre, mais cette approche aurait augmenté la polysémie. D’après (Candito et al., 2014), les domaines du FrameNet sont les suivants :

1) Transactions commerciales

2) Communication verbale

3) Évaluation/Jugement : évaluation ou jugement positif ou négatif d'une entité correspondant à une norme

4) Positions cognitives

5) Relations spatiales : relations locatives des cadres et les cadres de mouvement (en laissant de côté les cadres du mouvement du corps et les causalités de mouvement)

6) Relations temporelles : durée, etc.

7) Causalité

Seuls 4 des 7 domaines ci-dessus [1, 2, 4, 7] ont pu être annotés dans la première version (Djemaa et al., 2016)

Pour chaque domaine, deux experts se sont prononcés ensemble, à l'exception des cadres interdomaines, qui ont été jugés par des spécialistes des domaines concernés. Cette tâche a souvent conduit à enrichir et à clarifier les caractéristiques distinctives de certaines paires de cadres, et aussi dans certains cas à modifier les champs lexicaux des cadres.

Ils utilisent l'outil Salto (Burchardt et al., 2006 cité par [Candito et al., 2014]) qui permet aux annotateurs et annotatrices externes de choisir si le sens de x lemmes rentre dans un des contextes proposés. Les résultats sont ainsi analysés par des spécialistes du domaine afin de confirmer le vrai sens, c'est ce qu'on appelle la désambiguïsation, nous l'avons déjà vu dans le cadre du Word Sense Disambiguation (WSD : a pour but d'identifier le sens d'un mot dans une phrase donnée).

Cependant, si l'on compare le schéma UCCA (p. 17) qui peut être transposé au français, on voit que le FrameNet n'est pas le meilleur outil pour l'annotation. Comme nous l'avons déjà vu, il est possible pour n'importe qui d'annoter (avec un peu d'entraînement) avec le schéma UCCA alors que le FrameNet est plus difficile à prendre en main même s'il est plus précis dans l'étude des relations. UCCA reste cependant capable de détecter quand deux phrases sont sémantiquement similaires et ne consacrera qu'un seul schéma à ces deux phrases. Néanmoins, dans le cas de FrameNet, deux cadres seront consacrés à ces deux phrases. Les deux schémas d'annotation ont alors tous les deux leurs points forts : la précision pour FrameNet et une bonne capacité d'adaptation à plusieurs langues pour UCCA. Cependant, FrameNet n'est pas encore assez précis et simple d'utilisation pour l'appliquer à un outil qui annoterait automatiquement des textes passés en entrée.

Après avoir vu les outils pour l'annotation sémantique, nous allons maintenant étudier les ressources lexicales disponibles pour l'analyse sémantique. Dans ces ressources, nous trouverons principalement des dictionnaires. Ces dictionnaires seront des ressources utiles pour les outils d'annotation sémantique que nous venons d'étudier.

III. Ressources lexicales pour l'analyse sémantique

La conception d'un outil d'évaluation nécessite de s'appuyer sur des ressources qui permettent d'évaluer la pertinence d'une simplification automatique. Ainsi, il est nécessaire de faire un historique des ressources potentiellement utilisables dans le cadre de l'évaluation de la simplification automatique. Nous allons principalement étudier des dictionnaires qui sont soit syntaxiques, soit sémantiques, soit les deux [TreeLex++ ; FrameNet].

III.1 DICOVALENCE :

Quelle est la définition de valence ?

« a) Tesnière : Nombre d'actants qu'un verbe est susceptible de régir. Valence verbale. Les verbes intransitifs ont nécessairement la valence 1 (Mounin1974).

b) Valence lexicale : Indice de sélection égal au pouvoir d'un mot à se substituer à d'autres en contexte (Mounin, 1974) »

Source : (cnrtl.fr)

D'après (Mertens, 2010), DicoValence est un dictionnaire syntaxique se fondant seulement sur un vocabulaire non technique. Il permet ainsi de montrer les cadres de valences (c.-à-d. ce qui entoure le verbe ; voir définition cnrtl.fr) tout en décrivant avec précision tous les aspects des verbes proposés. Le dictionnaire comporte 3700 verbes dits « pleins ». Parmi ces 3700 verbes, certains possèdent plusieurs sens, donc plusieurs cadres de valence, c'est pour cela que le dictionnaire comporte environ 8000 entrées. L'utilisation de ces cadres de valence peut également s'appeler « l'approche pronominale ».

Il peut se retrouver utile pour les logiciels de TAL dans le cadre d'un besoin d'une analyse syntaxique poussée.

Quel type de structure a le dictionnaire DicoValence ?

Nous pouvons observer la structure dans l'exemple 1 (Mertens, 2010) :

```
VAL$     supprimer: P0 P1
VTYPE$   predicator simple
VERB$    SUPPRIMER/supprimer
NUM$     80500
EG$      nous avons supprimé tous les obstacles à la publication de ce dico
TR_DU$   afschaffen, opheffen, intrekken, weghalen, weglaten, schrappen, doen verdwijnen
TR_EN$   suppress
FRAME$   subj:pron|n:[hum], obj:pron|n:[hum,nhum,?abs]
P0$      que, qui, je, nous, elle, il, ils, on, ça, celui-ci, ceux-ci
P1$      que, qui, te, vous, la, le, les, se réfl., se réc., en Q, ça, ceci, celui-ci,
         ceux-ci, l'un l'autre
RP$      passif être, se passif, se faire passif
AUX$     avoir
```

Explication des entrées dans l'exemple 1 (Mertens, 2010, p. 5-6) :

VAL $ = entrée

VTYPE $ = type de verbe

VERB $ = le verbe en question

Num $ = nombre d'occurrences ?

EG % = exemple de phrase dans laquelle le verbe apparaît

TR_DU$ = traduction en néerlandais

TR_EN$ = traduction en anglais

FRAME $ = cadre valenciel

P0 $ = paradigme 0

P1 $ = paradigme 1

RP $ = reformulation passive

AUX $ = auxiliaire rattaché au verbe

Ce dictionnaire décrit donc syntaxiquement les verbes, avec leurs constructions, et ceci grâce à des schémas valentiels. DicoValence peut donc être utile comme ressource pour l'analyse syntaxique et sémantique d'une simplification automatique.

III.2 TreeLex++ (Kupść et al., 2019):

TreeLex++ est une extension de TreeLex qui est un lexique syntaxique pour le français. La source principale du lexique est le corpus FTB : corpus de journaux français (Le Monde, etc.).

D'après (Kupść et al., 2019), le lexique contient 1161 verbes qui sont enrichis par des cadres (qui apportent des précisions sur les propriétés syntaxiques des verbes) **sémantiques** (c.-à-d. leurs aspects lexicaux, ce qui le différencie *de facto* de DicoValence, qui a principalement un aspect syntaxique ; même si l'on peut y trouver une dimension sémantique, car il est possible d'ajouter des rôles aux arguments [agents, patients, bénéficiaires]). Chaque verbe est accompagné de son cadre, de son aspect lexical, du nombre d'exemples trouvés dans la FTB et de leur liste complète. TreeLex n'est ni syntaxiquement ni sémantiquement équilibrée, mais ce serait dû au contenu du corpus FTB. Avec TreeLex++ (Kupść et al., 2019) ont réglé les différents problèmes du TreeLex original, comme quand un mot est polysémique : plusieurs cadres lui sont attribuées alors que ses différents sens pourraient être contenus dans un seul et même cadre. Cela est possible grâce au fait qu'ils ont rajouté une dimension sémantique au lexique. TreeLex n'était qu'un lexique syntaxique.

Un tri dans les verbes présents dans TreeLex (format XML) est effectué. 5 catégories aspectuelles sont créées, les verbes sont ainsi placés dedans : état ; activité ; accomplissement ; résultat ; réussite.

Ci-dessous, nous pouvons observer un exemple d'entrée dans le lexique TreeLex :

(2) a. voler: SUJ:NP,OBJ:NP,A-OBJ:NP
b. voler: SUJ:NP,DE-OBJ:NP
c. voler: SUJ:NP

(Kupść et al., 2019)

Dans cet exemple nous pouvons voir le verbe « voler » trois fois. Cela peut être expliqué par le fait que « voler » est un verbe polysémique et nécessite donc plusieurs cadres. Dans les cadres a. et b il est dans sa forme « voler », dans le sens « prendre quelque chose à quelqu'un sans son consentement ». Dans c., il apparaît dans son sens voler dans les airs (aéronefs, etc.). On peut observer que chaque cadre possède son lot d'étiquettes (ex : SUJ:NP, l'étiquette veut dire que le verbe peut être le sujet d'un syntagme nominal [noun phrase en anglais].

Il est important de noter que la polysémie verbale n'a été prise en compte que si des significations différentes apparaissaient dans le corpus. Ainsi, il est possible qu'un verbe polysémique soit pris en compte comme verbe monosémique.

TreeLex++ est un lexique qui regroupe les propriétés syntaxiques et sémantiques de plus de mille verbes illustrés par des exemples attestés. Néanmoins, les verbes polysémiques n'ont plus qu'un seul cadre dans la version ++, les chercheurs ont préféré ne pas désambiguïser les verbes. Ils conseillent de coupler LVF et TreeLex++.

La ressource est disponible librement à l'adresse suivante (format CSV) :

http://redac.univ-tlse2.fr/lexiques/treelexPlusPlus.html

III.3 LVF : Les Verbes Français

LVF (Dubois et Dubois-Charlier, 1997) a pour but de classifier syntaxiquement les verbes français grâce à des schèmes syntaxiques qui seront regroupés à l'intérieur du dictionnaire. Ces schèmes syntaxiques seront ensuite regroupés dans des catégories sémantiques (cf. Figure 6). Par conséquent, une fois classés syntaxiquement, le but est de les distinguer par leurs catégories sémantiques, ce qui fait que LVF est un dictionnaire sémantico-syntaxique.

LVF comporte 25 610 entrées (disponibles en format XML), ce qui correspond à environ 12 310 verbes différents.

Il est normal de trouver plus d'entrées que de verbes, car un seul et même verbe peut avoir jusqu'à plus de 10 entrées différentes. Le verbe *passer* est une valeur dite « aberrante », car il comporte 61 entrées différentes, donc 61 contextes dans lesquels il a un sens différent.

À quoi correspond une entrée ?

Une entrée est une occurrence d'un verbe dans un contexte.

Les 25 160 entrées sont regroupées dans 14 classes dites « génériques » :

C	communication,	N	munir, démunir
D	don, privation	P	verbes psychologiques
E	entrée, sortie	R	réalisation, mise en état
F	frapper, toucher	S	saisir, serrer, posséder
H	états physiques et comportements	T	transformation, changement
L	locatif	U	union, réunion
M	mouvement sur place	X	verbes auxiliaires

Figure 6 : Liste des 14 classes génériques pour les verbes

Source : Dubois et Dubois-Charlier, 1997

D'après (Dubois et Dubois-Charlier, 1997), ces classes sont elles-mêmes divisées en sous-classes. Nous pouvons les identifier à l'aide d'un chiffre comme « L1 » par exemple, elles constituent les classes sémantico-syntaxiques. Il existe également des sous-classes syntaxiques qui vont s'ajouter après le chiffre, ce qui peut éventuellement donner « C1a ». Elles ont pour but de faire la distinction entre les classes F1 et F3 par exemple, qui sont toutes les deux le sens concret de F. Selon ces auteurs (*ibid.*), il constitue un corpus lexicographique.

Le dictionnaire LVF est disponible en libre-accès au format XML sur le site de l'Université de Montréal (Dubois-Charlier, [s.d.]). Le contenu du dictionnaire sera accessible à travers des classes. Chaque verbe peut être consulté et ils sont triés par ordre alphabétique. Les expressions régulières peuvent être utilisées pour rechercher un verbe ou son lemme.

Exemple pour le verbe « abaisser » :

---

9 entrées pour abaisser

Version XML

1 **mot**	no:*1* = abaisser 01
domaine	LOC
domaineEnClair	locatif, lieu
operateur	predicat:*r/d*, complement:*bas qc* = (#) [r/d] bas qc
classe	generique:*transformation, changement*, semantico-syntaxique:*non-animé propre*, construction-syntaxique:*c* = T3c
sens	baisser
phrases	**phrase:** On a~ le rideau de fer, le store. **phrase:** Le rideau du magasin s'a~.
conjugaison	groupe:*1*, sous-groupe:*baisser, pleurer, etc.*, auxiliaire:*avoir (sauf si pronominal ou entrée en être)* = 1bZ
construction	**scheme:** type:*transitif direct*, sujet:*humain*, objet:*chose*, circonstant:*instrumental, moyen* = T1308 **scheme:** type:*pronominal*, sujet:*chose*, circonstant:*instrumental, moyen* = P3008
derivation	1-- -1 --RA --
der-able	positif seul (parfois avec modification du radical)
der-ment	formation directe sur la conjugaison
der-eur	eur, ion
nom	-I
lexique	desc:*dictionnaire de base*, nbmots:*15000* = 2

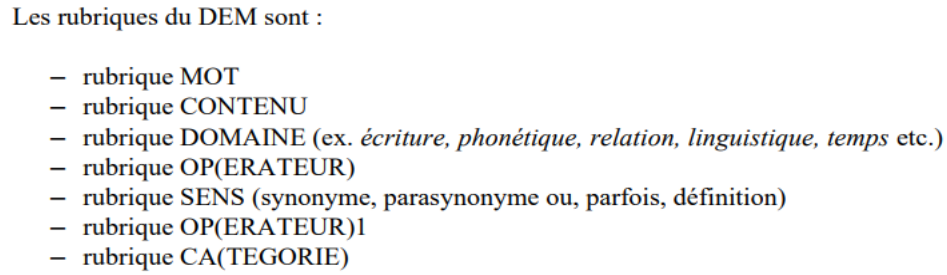
LVF : verbe « abaisser » *source : http://rali.iro.umontreal.ca/LVF+1/alphabetique/A/abaisser.html*

LVF est donc un dictionnaire très complet qui aurait pu être utile si nous avions dû réaliser un outil en partant de zéro. Cependant, vu que ce n'est pas le cas, nous n'utiliserons pas LVF dans l'outil d'annotation et d'évaluation sémantique nous réaliserons.

III.4 DEM : Dictionnaire Électronique des Mots

Le dictionnaire DEM (Dubois et Dubois-Charlier, 2014) est complémentaire de LVF et LOCVERB (qui est un dictionnaire regroupant les locutions verbales et qui lui est complémentaire à LVF.)

Il y a 145 333 entrées dans le dictionnaire. Le dictionnaire est composé de mots de toutes les catégories. Ci-dessous, nous pouvons trouver les catégories du DEM.



Les rubriques du DEM sont :

- rubrique MOT
- rubrique CONTENU
- rubrique DOMAINE (ex. *écriture, phonétique, relation, linguistique, temps* etc.)
- rubrique OP(ERATEUR)
- rubrique SENS (synonyme, parasynonyme ou, parfois, définition)
- rubrique OP(ERATEUR)1
- rubrique CA(TEGORIE)

Figure 7 : Rubriques du dictionnaire DEM | source : (Dubois et Dubois-Charlier, 2014)

On peut retrouver des parties interconnectées entre les trois dictionnaires ; comme la rubrique OP, qui est commune aux trois dictionnaires.

(Dubois et Dubois-Charlier, 2014) ont pour projet de fusionner les trois dictionnaires LVF, LOCVERB et DEM dans le but de former un ensemble cohérent qui puisse être utilisé par les outils de TAL comme les outils d'annotation sémantique par exemple.

Le dictionnaire électronique des mots est une ressource qui peut être utilisée dans le cadre de mon mémoire pour le développement d'un outil d'évaluation qui pourra s'appuyer sur ces trois dictionnaires [DEM, LOCVERB, LVF].

III.5 FRILEX

Le lexique FRILEX (Guillaume et al., 2014) est un regroupement du dictionnaire DicoValence (Mertens, 2010) et de LVF (Dubois et Dubois-Charlier, 1997). Il repose sur le fonctionnement de DicoValence car il est très précis (par exemple, il contient les possibles restrictions sémantiques du verbe et bien d'autres informations). Les chercheurs ont tenté de le regrouper avec LVF qui est beaucoup plus exhaustif (25 610 entrées contre 3700 dans DicoValence). Les deux [DicoValence, LVF] reposent sur des théories linguistiques très différentes ce qui rend le fusionnement des dictionnaires difficile (Guillaume et al., 2014). Le résultat est que les chercheurs (*ibid.*) ont partiellement réussi à fusionner automatiquement les deux lexiques. Cela nous permet d'avoir une nouvelle ressource potentiellement utilisable.

Ainsi, nous avons étudié différentes ressources potentiellement utilisables pour mon projet. Certaines comme LVF et DicoValence sont complémentaires ainsi que TreeLex++ avec LVF. Le tout sera d'évaluer quelle ressource — ou combinaison de ressources — est la plus qualitative afin de servir de base pour mon projet de créer un outil d'annotation sémantique.

Après avoir évalué les ressources de type lexique, nous allons maintenant étudier le fonctionnement de BERT. C'est une ressource construite à base de corpus qui sera le principal constituant de cet outil étant donné son architecture (transformer) et ses dérivés pour le français comme CamemBERT et FlauBERT, qui nous donnent une base pour mon outil d'annotation.

IV. Bi-LSTM : Bidirectional Long Short-Term Memory

IV.1. Introduction aux Réseaux de Neurones Récurrents :

Pour comprendre les modèles LSTM, il faut d'abord comprendre ce qu'est un RNR[1] :

Un RNR est un type de réseau neuronal qui a la capacité de traiter des données séquentielles en tenant compte du contexte antérieur. Pour mieux comprendre ce qu'est un RNR, nous pouvons nous appuyer sur l'analogie décrite par Christopher Olah (Olah, [s.d.]). Imaginez que vous lisez un texte. Chaque mot que vous lisez est compris en se basant sur votre compréhension des mots précédents. Votre cerveau garde en mémoire les mots que vous avez déjà lus, ce qui vous permet de comprendre le sens global du texte. C'est un peu comme si votre pensée avait de la persistance et n'était pas réinitialisée à chaque mot. Un RNR fonctionne de manière similaire. Il est conçu pour traiter des séquences de données, comme des phrases, du texte, des séquences temporelles, etc. Chaque élément dans la séquence (par exemple, un mot dans une phrase) est traité en tenant compte des éléments précédents grâce à des boucles internes dans le réseau.

Dans un RNR, à chaque étape de la séquence, le réseau prend en compte l'élément actuel (par exemple, un mot) et aussi l'information provenant de l'étape précédente (ce qui correspond au contexte antérieur). Cette information est propagée à travers les différentes étapes grâce à ces boucles internes, permettant ainsi de conserver un état interne représentant le contexte passé.

Passons maintenant de l'introduction aux Réseaux de Neurones Récurrents à une exploration plus approfondie des modèles LSTM, souvent considérés comme une amélioration significative par rapport aux RNR classiques.

IV.2. Les modèles LSTM (Hochreiter et Jürgen, 1997):

Les modèles LSTM sont présentés (Hochreiter et Jürgen, 1997) comme le remède aux problèmes d'apprentissage des réseaux de neurones récurrents générés par les méthodes traditionnelles comme BPTT « Back-Propagation Through Time » ou RTRL « Real-Time Recurrent Learning » qui ne transmettaient pas correctement les signaux d'erreurs lors de l'apprentissage, et ce pour deux raisons :

- 1) Explosion des signaux d'erreur : dans certains cas, les signaux d'erreur augmentent de manière exponentielle à mesure qu'ils sont propagés en arrière dans le réseau. Cela peut provoquer des oscillations dans les poids du réseau, ce qui rend l'apprentissage difficile.
- 2) Disparition des signaux d'erreur : dans d'autres cas, les signaux d'erreur diminuent exponentiellement à mesure qu'ils sont propagés en arrière. Cela signifie que l'apprentissage pour relier des informations distantes dans la séquence prendrait un temps prohibitif ou ne fonctionnerait pas du tout

Hochreiter et Jürgen (1997) suggèrent que ces problèmes sont liés à la taille des poids dans le réseau de neurones récurrents. Plus précisément, lorsque les poids sont trop grands, les signaux d'erreur ont tendance à exploser, tandis que lorsque les poids sont trop petits, les signaux d'erreur ont tendance à disparaître. Ces problèmes rendent l'apprentissage difficile, en

[1] RNR : Réseau de Neurones Récurrents. « Recurrent Neural Network » en anglais (RNN).

particulier lorsqu'il est nécessaire de modéliser des dépendances à long terme dans les données, comme cela peut être le cas dans le traitement de séquences temporelles.

Les modèles LSTM sont connus pour leur capacité à reconnaître les relations sémantiques lointaines, et de ce fait les dépendances, entre les mots ce qui est évidemment crucial lors de la compréhension d'un texte écrit. Cette capacité à mémoriser les informations se caractérise avec des cellules de mémoire spécialement conçues pour le bon fonctionnement du réseau. Précédemment, les RNN fonctionnaient avec des couches cachées qui pouvaient rendre le traitement d'informations lointaines complexe (Huang, Xu, et Yu, 2015)

Les modèles bi-LSTM (bidirectional long short-term memory) sont un type de réseau de neurones récurrents qui peuvent traiter un texte dans les deux sens. Selon (citer le gars BILSTM), l'avantage des réseaux Bi-LSTM (Long Short-Term Memory bidirectionnel) est leur capacité à exploiter à la fois les caractéristiques passées et futures pour chaque instant de temps dans la tâche d'étiquetage de séquences. Cette approche permet au modèle de prendre en compte un contexte plus large et d'améliorer ainsi sa compréhension des séquences, ce qui peut conduire à de meilleures performances dans cette tâche. Ils peuvent donc être utilisés dans un large éventail de tâches dans le domaine du TAL[2] comme :

- Étiquetage morphosyntaxique (Part of Speech tagging)
- La classification de textes, par exemple : analyse de sentiments.
- La traduction automatique
- La reconnaissance d'entités nommées

Maintenant que nous avons étudié les modèles bi-LSTM, plongeons dans une application spécifique qui exploite les avantages du bi-LSTM dans le domaine du TAL, le cas de TUPA.

IV.3. Exemple d'une application d'un modèle bi-LSTM : TUPA (Hershcovich, Abend, et Rappoport, 2017)

TUPA utilise un modèle bi-LSTM entraîné sur un corpus monolingue pour annoter des textes avec le schéma UCCA que nous avons étudié précédemment (p. 15). C'est un exemple concret d'utilisation puisque TUPA réalise un étiquetage de séquences de textes avec le schéma UCCA.

TUPA utilise un modèle bi-LSTM pour représenter les caractéristiques des données en plus des caractéristiques d'incorporation denses[3]. Le modèle utilisé analyse les tokens en entrée dans les deux directions (avant/arrière)[4] dans le but de créer une représentation vectorielle. Celle-ci est combinée avec les résultats du parseur qui sont ici représentés par les étiquettes du schéma UCCA. Ainsi, nous pouvons retrouver en sortie de l'outil, TUPA ici, une représentation

[2] TAL : Traitement Automatique des Langues

[3] Les caractéristiques d'incorporation dense (ou dense embeddings) sont une représentation numérique dense et continue de mots ou de symboles dans un modèle de langue. Contrairement aux caractéristiques d'incorporation éparses (sparse embeddings), qui sont généralement des vecteurs creux avec beaucoup de zéros, les caractéristiques d'incorporation denses attribuent à chaque mot ou symbole un vecteur de nombres réels de dimensions relativement élevées.

[4] Cela signifie qu'il examine les mots ou les symboles de la séquence d'entrée à la fois en commençant par le premier mot et en se déplaçant vers la fin de la séquence (direction avant) et en commençant par le dernier mot et en se déplaçant vers le début de la séquence (direction arrière).

vectorielle des mots annotée avec le schéma. Nous pouvons représenter le modèle bi-LSTM comme suit :

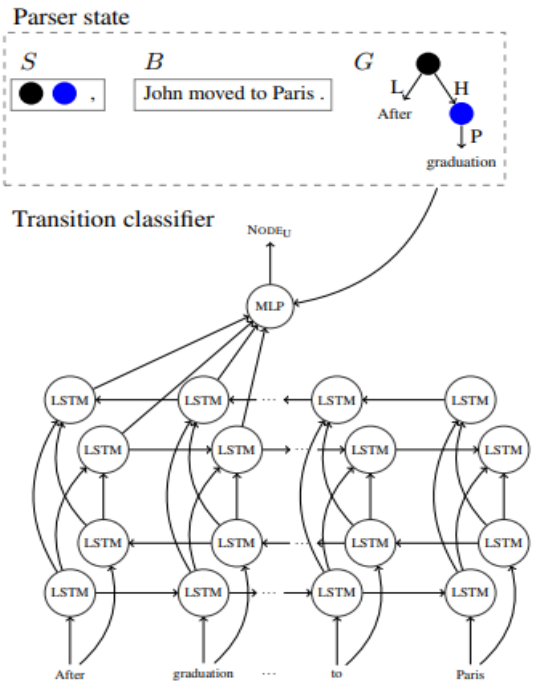


Figure 8: Illustration du modèle TUPA bi-LSTM (Hershcovich, Abend, et Rappoport, 2017) p.5

Cette image résume le fonctionnement de TUPA lors de la phase d'entraînement. D'après (Hershcovich, Abend, et Rappoport, 2017), l'« état du parseur » (« Parser State » en anglais) avec la lettre S correspond aux nœuds du graphe qui sont en cours d'analyse ; la lettre B représente les tokens qui doivent être analysés ; et la lettre G représente les graphes construits à partir du texte.

Ensuite, nous pouvons retrouver la représentation de l'architecture de TUPA bi-LSTM. D'après (Hershcovich, Abend, et Rappoport, 2017, p.5), les tokens en entrée sont traités par deux couches LSTM bidirectionnelles. Les vecteurs sont ainsi concaténés avec les résultats obtenus avec l'état du parseur pour ensuite alimenter le MLP[5] pour déclencher la prochaine transition.

Ce procédé d'entraînement permet d'obtenir des graphes qui correspondent à des représentations vectorielles des mots d'une séquence de textes donnée. Un exemple est disponible sur GitHub[6] avec une phrase annotée provenant du texte « dragonne_minuit » provenant du corpus ALECTOR. Dans cet exemple, nous pouvons voir que chaque mot est relié par un vecteur, le reliant à une ou plusieurs scènes qui englobent la séquence de texte.

[5] D'après (Hershcovich, Abend, et Rappoport, 2017), $Tupa_{MLP}$ est un réseau neuronal feedforward (ou réseau neuronal à propagation avant). C'est un type de réseau où l'information circule dans une seule direction, de l'entrée vers la sortie, sans boucles ni connexions récurrentes. Ultimement, cela permet d'encoder des informations riches sur le sens, la relation et le contexte des mots insérés en entrée.

[6] https://github.com/jessy3ric/git_memoire/blob/main/corpus/corpus_annoté_bilstm/visualisations/dragonne_minuit_bilstm.png

Maintenant que nous avons étudié le fonctionnement des modèles bi-LSTM en détail avec un exemple concret (TUPA), il est intéressant d'étudier un autre type de modèle fonctionnant avec l'architecture « transformers » : BERT. Nous allons premièrement étudier en détail son fonctionnement puis nous allons effectuer une comparaison entre les deux types de modèles (BERT et bi-LSTM) à travers le cas de TUPA. Ensuite, nous allons étudier BERT dans une tâche précise du TAL, ici la substitution lexicale, pour conclure sur une analyse des dérivés de BERT pour le français.

V. BERT : Bidirectional Encoder Representations from Transformers

V.1 Qu'est-ce que BERT ? (Devlin et al., 2019)

BERT représente une structure proposant des modèles de langue préentraînés. Ces modèles peuvent être affinés pour s'appliquer à des tâches spécifiques telles que : réponse à des questions, compréhension écrite, etc. Tout comme FLAIR, c'est une ressource qui représente les mots en contexte.

Les données sont un facteur clé pour BERT, car il va être entraîné sur ces données pour faire des prédictions. Ainsi, meilleures les données sont, meilleures les prédictions seront.

BERT est un convertisseur (transformer en anglais). D'après (Vaswani et al., 2017), un convertisseur se fonde sur des mécanismes d'attention. Pour n'en citer qu'un, celui sur lequel les convertisseurs reposent : self-attention. Le principe de ce mécanisme est de relier différentes positions d'une même séquence afin de calculer une représentation de la séquence. Cela implique que l'outil tienne compte d'un contexte plus large du mot (de tous les mots qui le précèdent et même ceux qui suivent) comparé à d'autres, on peut donc l'appliquer à tout moment.

Le mécanisme de self-attention peut être utile dans les domaines qui nécessitent de tenir compte des contextes plus larges (extraction d'entités nommées, traduction automatique etc.).

Ci-dessous un schéma représentant les deux étapes principales pour BERT. La première étant le préentraînement, étape dans laquelle BERT attribue un poids à la phrase masquée A et également à la phrase masquée B. BERT choisit les cellules à cacher et les remplace par des tokens<MASK >.

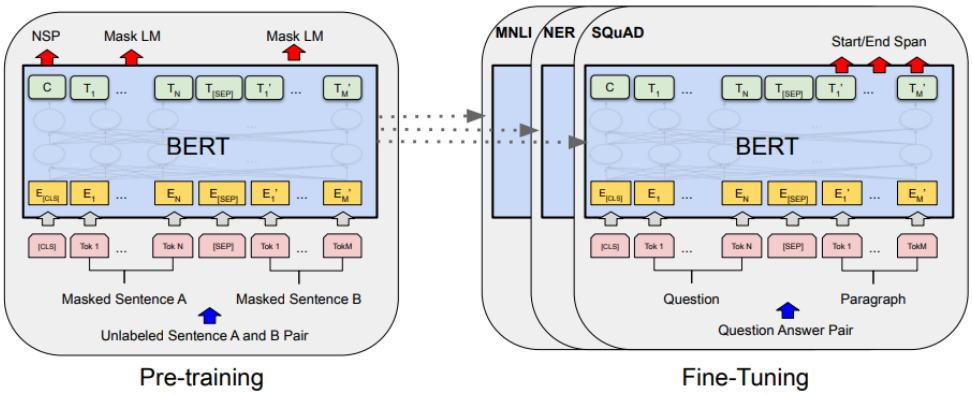


Figure 9 : Procédures de préentraînement et de peaufinage par BERT (Devlin et al., 2019)

La deuxième étape concerne le peaufinage de la ressource. Par conséquent, BERT sera amené à se spécialiser dans des tâches comme la NER (*reconnaissance automatique des entités nommées*, en anglais Named Entity Recognition), il sera donc entraîné à reconnaître des entités

nommées par exemple, mais également pour nombre d'autres tâches différentes comme les questions-réponses, la substitution lexicale (cf. Partie II : Bert appliqué à la substitution lexicale (Zhou et al., 2019).

Un des indicateurs pouvant évaluer les résultats de BERT est GLUE (the General Language Understanding Evaluation : Évaluation de la Compréhension Générale du Langage/de la langue ; ma traduction) qui consiste en l'évaluation avec des mesures de la compréhension du modèle à travers différentes tâches. D'après (Wang et al., 2019), GLUE rassemble un panel d'outils conçus pour évaluer la performance des modèles de CLN (Compréhension du Langage Naturel) qui ont pour but de comprendre les intentions et les significations profondes du texte écrit ou oral. GLUE évalue ces ressources sur plusieurs tâches en même temps en leur donnant le moins de données d'entraînement possibles, afin d'évaluer leur plasticité. GLUE repose sur différents corpus provenant de jeu de données disponibles sur internet. Chaque corpus a un thème différent. Les modèles seront donc évalués sur leur performance dans ces corpus. Les tâches sur lesquelles les modèles sont évalués sont les suivantes :

1. Tâche sur une seule phrase qui consiste à donner au modèle une phrase qu'il devra considérer comme grammaticale ou non.
2. Tâche de paraphrase et de similarité : le modèle devra deviner si la phrase donnée est une paraphrase ou s'il existe seulement des similarités dans le sens.
3. Tâche d'inférence : le modèle doit déterminer, d'après une phrase prémisse et une phrase hypothèse, si la phrase prémisse confirme, infirme ou reste neutre à l'égard de la phrase hypothèse.

Au moment de l'écriture de leur article (Devlin et al., 2019) montraient que $\text{BERT}_{\text{LARGE}}$ obtenait les meilleurs résultats pour un modèle parmi ceux qui étaient disponibles. Depuis, beaucoup de nouveaux modèles ont fait leur apparition. La plupart reposent sur BERT ou un de ses dérivés (RoBERTa par exemple).

V.2 Comparaison de plongements statiques et approches dynamiques de BERT

Les plongements statiques sont une partie cruciale dans le fonctionnement des modèles comme Bert. Ils offrent une représentation numérique des mots, capturant leurs significations et relations sémantiques. Deux approches majeures ont émergé : la vectorisation des mots non contextualisée et la vectorisation des mots contextualisée.

La première approche, représentée par Word2Vec (Mikolov et al., 2013), a été largement adoptée. Elle implique la création de vecteurs de mots en se basant sur les co-occurrences statistiques dans un grand corpus textuel. Cette méthode a été mentionnée par (Loureiro et Jorge, 2019) comme une référence en matière de vectorisation non contextualisée. De même, GloVe (Pennington, Socher, et Manning, 2014) est une méthode populaire qui combine la co-occurrence avec des informations de décomposition matricielle, offrant des résultats comparables.

D'autre part, la vectorisation des mots contextualisée a ouvert de nouvelles perspectives grâce à des modèles tels qu'ELMo (Peters et al., 2018) et FLAIR (Akbik et al., 2019). Contrairement aux méthodes non contextualisées, ces approches prennent en compte le contexte dans lequel un mot apparaît, offrant ainsi des représentations plus riches et nuancées. Elles sont particulièrement utiles pour résoudre le problème de polysémie, où un mot a plusieurs sens en

fonction du contexte. Cela rend les plongements contextualisés idéaux pour des tâches telles que la classification de sentiments et la compréhension de la syntaxe.

L'intégration de plongements statiques dans la recherche en TAL ouvre la voie à des applications diverses et innovantes. En combinant des approches non contextualisées et contextualisées, les chercheurs peuvent explorer de manière plus approfondie la richesse sémantique des mots et améliorer la performance de diverses tâches liées au traitement du langage naturel.

Les plongements statiques ont joué un rôle fondamental dans le développement de modèles de traitement du langage naturel préentraîné tels que BERT (Bidirectional Encoder Representations from Transformers) et CamemBERT. Ces modèles exploitent les avantages des plongements pour capturer les informations sémantiques et contextuelles des mots, ce qui améliore considérablement leur capacité à comprendre et générer du texte.

Quant à BERT, il utilise une approche bidirectionnelle où il prend en compte le contexte avant et après chaque mot dans une phrase. Les plongements statiques, dans ce cas, sont utilisés comme une étape de prétraitement. BERT intègre ensuite ces plongements avec des mécanismes de transformation de type Transformer pour capturer les relations complexes entre les mots dans un texte. Cela permet à BERT de comprendre le sens global d'une phrase en fonction du contexte.

De même, CamemBERT, une version adaptée de BERT pour la langue française, exploite des plongements statiques spécifiquement conçus pour le français. Ces plongements fournissent une représentation numérique de chaque mot, qui est ensuite utilisée comme entrée pour les couches de transformation du modèle. La combinaison des plongements et des transformations permet à CamemBERT de comprendre les subtilités du français et d'accomplir une gamme de tâches de traitement du langage naturel.

Après avoir étudié en profondeur les plongements statiques, nous allons aborder une tâche sur laquelle Bert a pu être entraîné : la substitution lexicale. Cela nous permettra d'étudier de nouvelles caractéristiques sur ce modèle ainsi que ses dérivés.

V.3 Analyse comparative de BERT et des modèles bi-LSTM :

Les modèles BERT et les modèles bi-LSTM représentent deux approches majeures dans le TAL. Alors que BERT est un modèle massivement préentraîné basé sur des Transformers, les modèles bi-LSTM sont des réseaux de neurones récurrents (RNR) bidirectionnels, comme nous l'avons vu dans la partie précédente.

Commençons par BERT. BERT est largement reconnu pour sa capacité à capturer les relations contextuelles entre les mots d'une phrase. En se basant sur un grand corpus de texte, BERT génère des représentations riches pour chaque mot, en tenant compte du contexte de manière bidirectionnelle. Ces représentations sont ensuite utilisées pour des tâches spécifiques (classification de textes, étiquetage morphosyntaxique, etc.), souvent par fine-tuning.

Les modèles bi-LSTM, d'autre part, sont des réseaux de neurones récurrents bidirectionnels. Ils sont particulièrement adaptés pour le traitement des séquences grâce à leur capacité à capturer les dépendances complexes dans les données séquentielles. Un aspect important est qu'ils sont moins gourmands en ressources que BERT, ce qui les rend plus accessibles et plus faciles à entraîner, même avec des ensembles de données plus modestes. Cependant, cette accessibilité est au détriment des performances, car souvent les modèles Transformers comme BERT sont vraiment plus efficaces quand ils sont fine-tuned que ce soit pour une langue et/ou une tâche précise. En effet, BERT excelle dans la contextualisation des mots, car il prend en considération le contexte global de chaque mot dans la phrase, ce qui le rend redoutable dans l'étiquetage avec le schéma UCCA, puisqu'il demande beaucoup de contexte pour déterminer les dépendances entre les différents mots.

Reprenons l'exemple de TUPA pour explorer plus en détail ses performances et les avantages liés à l'utilisation de différents modèles, notamment les modèles bi-LSTM et Bertm (la version multilingue de BERT). TUPA est un outil disponible avec des modèles bi-LSTM entraînés sur des corpus monolingues (français, anglais, allemand) ainsi qu'avec Bertm.

Dans une étude précédente (Hershcovich et Arviv, 2019), il a été démontré que TUPA présente des performances plus élevées dans sa représentation avec le schéma UCCA lorsqu'il utilise les plongements statiques inhérents à ce type de modèle, comme nous l'avons exploré précédemment. Cette amélioration de performance est particulièrement notable lorsque la langue d'entraînement n'est pas l'anglais.

SemEval 2019	All	Prim.	Rem.
English-Wiki (open)			
TUPA (w/o BERT)	73.5	73.9	53.5
TUPA (w/ BERT)	77.8	78.3	57.4
Jiang et al. (2019)	**80.5**	**81.0**	**58.8**
English-20K (open)			
TUPA (w/o BERT)	68.4	69.4	25.9
TUPA (w/ BERT)	74.9	75.7	**44.0**
Jiang et al. (2019)	**76.7**	**77.7**	39.2
German-20K (open)			
TUPA (w/o BERT)	79.1	79.6	59.9
TUPA (w/ BERT)	81.3	81.6	**69.2**
Jiang et al. (2019)	**84.9**	**85.4**	64.1
French-20K (open)			
TUPA (w/o BERT)	48.7	49.6	2.4
TUPA (w/ BERT)	72.0	72.8	**45.8**
Jiang et al. (2019)	**75.2**	**76.0**	43.3

Figure 10 (Hershcovich et Arviv, 2019, p.36)

Le tableau illustré dans la Figure 10 (Hershcovich et Arviv, 2019, p.36) montre les F-scores[7] obtenus lors de l'évaluation des résultats de TUPA sur différents corpus, en utilisant à la fois le

[7] F-scores : Le f-score englobe une famille de mesures couramment utilisée dans le domaine du Machine Learning et en Recherche d'informations pour évaluer la précision et le rappel d'un modèle. Il représente la moyenne harmonique de la précision et du rappel, fournissant une mesure équilibrée de la performance.

modèle bi-LSTM et le modèle Bertm. Une différence significative de performances est observée sur le corpus français, appelé « French-20K ». Plus spécifiquement, en considérant toutes les connexions et relations sémantiques dans les données annotées avec le schéma UCCA (colonne « all »), TUPA avec BERT (TUPA w/ BERT) obtient un F-score de 72,0, tandis que la version avec un modèle bi-LSTM obtient un score beaucoup plus faible, soit 48,7. Cette variation de performances est notable et souligne l'impact du choix du modèle sur les résultats, en particulier dans des langues autres que l'anglais.

Cependant, pour un corpus plus vaste et en anglais, tel que l'« English Wiki Corpus », la différence de performances entre TUPA bi-LSTM (F-score de 73,5) et TUPA avec Bertm (F-score de 77,8) est moins prononcée. Ces résultats mettent en évidence l'influence de la langue et de la complexité du corpus sur les performances des modèles.

Ainsi, après cette analyse comparative entre BERT et les modèles bi-LSTM, nous nous tournerons vers d'autres applications de BERT dans le TAL. Nous débuterons par examiner l'application de BERT à la substitution lexicale avant d'explorer les dérivés de BERT spécifiquement conçus pour le français.

V.4 BERT appliqué à la substitution lexicale (Zhou et al., 2019)

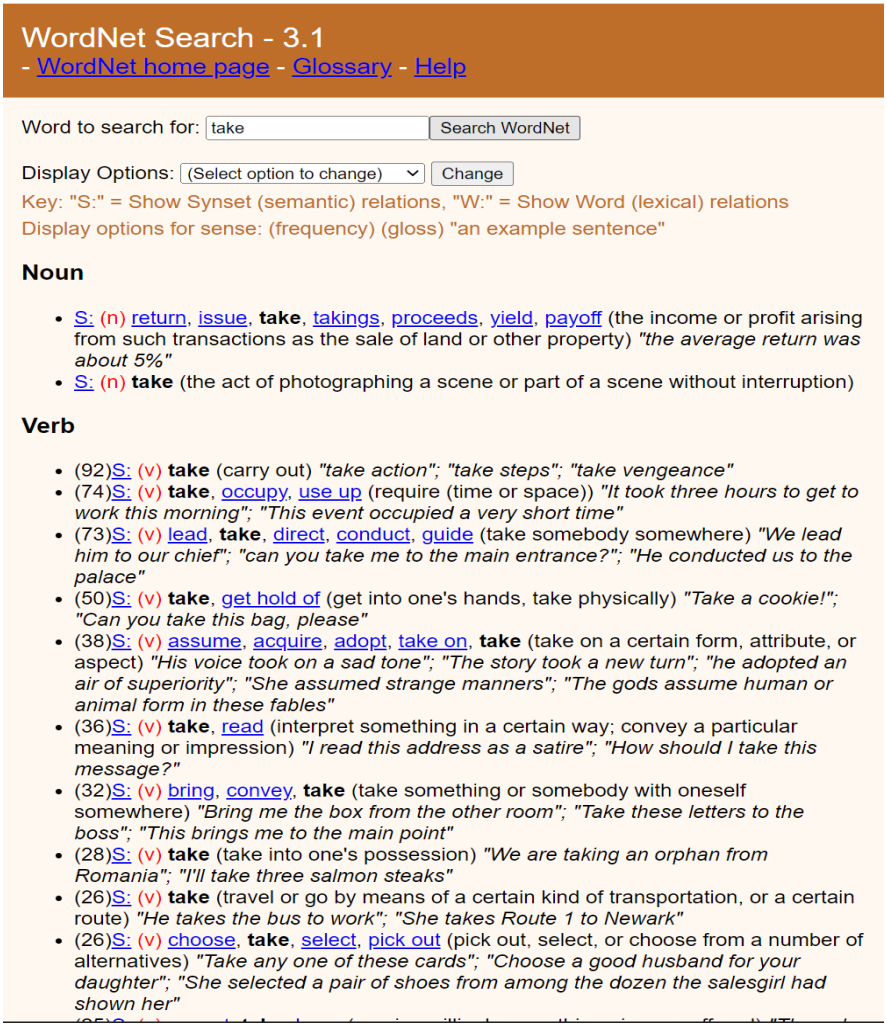
D'après (Zhou et al., 2019), BERT peut s'appliquer au principe de substitution lexicale expliqué par McCarthy (McCarthy et Navigli, 2009).

Le principe de la substitution lexicale est de remplacer un mot « cible » par un substitut qui ne change pas le sens de la phrase (Zhou et al., 2019), ce qui serait utile dans la simplification automatique. BERT peut accomplir cette tâche, même s'il est plus efficient en anglais, même dans son dérivé mBert (BERT mutilingue). Ainsi, comme nous le verrons plus tard, il existe des dérivés pour le français (cf. CamemBERT (Martin et al., 2020), Flaubert (Le et al., 2020)) qui sont des modèles spécialisés pour une seule langue, cela a tendance à optimiser les résultats de l'outil.

Les chercheurs ont démontré que BERT obtenait de meilleurs résultats que WordNet qui est un thésaurus. Comment fonctionne WordNet ?

Tout d'abord, WordNet, qui adopte une approche symbolique c'est-à-dire un travail manuel de réflexion, est également une ressource pour la représentation sémantique des mots. Cette méthode est à différencier de la méthode employée par word2vec par exemple, qui s'appelle la représentation des mots statiques. Cette méthode est très limitée dans le sens où les *embeddings (plongements lexicaux)* ne pourront pas capturer la polysémie des mots. WordNet, quant à lui, est un répertoire à sens. D'après (Loureiro et Jorge, 2019, p. 5684), tous les mots enregistrés dans WordNet sont regroupés dans des domaines de concepts généraux qui eux-mêmes sont liés par des relations comme la synonymie ou l'hyperonymie. La ressource WordNet se fonde principalement sur les synsets [ensembles de mots synonymes construits manuellement], ce qui le différencie de BERT par exemple, car BERT se fonde sur des corpus alors que WordNet est une ressource lexicographique, construite manuellement. De plus, BERT ne s'appuie pas sur des synsets. Ces mêmes synsets possèdent différents attributs : premièrement la glose du mot,

donc sa définition en quelques mots. Ensuite, sa relation hyperonymique avec d'autres synsets. Exemple : fruit est l'hyperonyme de fraise. Ces synsets sont ainsi rangés dans des « lexnames », qui sont des regroupements syntaxiques et logiques des mots.



Source : http://wordnetweb.princeton.edu/perl/webwn

Ci-dessus un exemple de recherche dans le thésaurus WordNet. Plusieurs options sont disponibles pour l'affichage. Il est possible, par exemple, de montrer la fréquence de l'usage du mot « take » en verbe dans sa signification « carry out ». Dans la capture d'écran ci-dessus, l'option est sélectionnée. Ainsi, nous pouvons observer le nombre « 92 » entouré de parenthèses pour le premier cas de « take » dans sa forme verbale.

D'après (Zhou et al., 2019), il est nécessaire de cacher partiellement le mot cible afin d'avoir les meilleurs résultats possibles. En effet, lorsqu'ils ne cachaient pas partiellement (c'est-à-dire cacher le mot, mais montrer son sens à l'outil) BERT n'arrivait pas à substituer le mot cible correctement. Cela permet, toujours selon (Zhou et al., 2019), à BERT de produire des mots qui sont sémantiquement différents du mot cible. Et retomberait à 99,99 % dans le mot original si celui-ci n'était pas masqué.

BERT prend également en compte les changements causés par la substitution du mot dans la phrase ; ce que ne font pas les ressources comme WordNet.

Comment fonctionne BERT ?

Il se sert du contexte autour du mot cible pour trouver le meilleur substitut.

Premièrement, l'établissement d'un score de proposition (proposal score) selon la formule suivante :

$$s_p(x'_k|\boldsymbol{x}, k) = \log \frac{P(x'_k|\widetilde{\boldsymbol{x}}, k)}{1 - P(x_k|\widetilde{\boldsymbol{x}}, k)}$$

(Zhou et al., 2019) (p.3370)

S = score de proposition avec l'argument x'$_k$ qui correspond au possible substitut pour le mot x_k

P correspond à la probabilité pour le k^e mot prédit par BERT étant donné x, $\acute{x}$ est similaire, mais la position du mot est partiellement masquée.

La proposition des substituts au mot cible est la première étape. Ensuite, il faut valider ces substituts pour voir lequel est le plus apte à remplacer le mot cible tout en changeant le moins possible le sens de la phrase. Comment procèdent-ils ? Ils comparent la représentation contextualisée de la phrase source (représentation de la phrase dans son contexte pour avoir son sens complet contrairement à la méthode de plongement lexical) avec la phrase modifiée.

BERT utilise une méthode différente du plongement lexical (aussi vu comme vectorisation des mots) grâce à son mécanisme d'attention, qui permet de représenter les contextes et d'en tenir compte dans le cadre de la substitution par exemple.

Ainsi, un score de validation est calculé avec la formule suivante :

$$s_v(x'_k|\boldsymbol{x}, k) = \textsc{Sim}(\boldsymbol{x}, \boldsymbol{x'}; k)$$

(Zhou et al., 2019)

SIM représente la similarité de x et x », sa définition est la suivante :

$$\textsc{Sim}(\boldsymbol{x}, \boldsymbol{x'}; k) = \sum_{i}^{L} w_{i,k} \times \Lambda(\boldsymbol{h}(x_i|\boldsymbol{x}), \boldsymbol{h}(x'_i|\boldsymbol{x'}))$$

(Zhou et al., 2019)

D'après (Zhou et al., 2019, p. 3370), l'impact de la substitution x_k à x'$_k$ peut donc être mesuré par s_v (x'$_k$, x, k) dans un cadre sémantique. Ainsi, les substituts ayant un score de validation s_v bas ne seront plus considérés comme candidats. Ceux ayant un score s_v élevé par rapport aux autres synonymes candidats, quant à eux, seront les favoris. H est la représentation du i^{ème} token dans la phrase x. Λ (a, b) est le cosinus de similarité du vecteur a et b $wi_{i,k}$ est la moyenne du score d'attention de toutes les têtes dans toutes les couches du i^{ème} token à la k^{ième} position dans x. Ce score est utilisé comme poids pour chaque position selon sa dépendance sémantique à x_k.

Finalement, les deux scores s_v et s_p seront considérés afin de proposer le résultat le plus juste. Ainsi, la formule sera la suivante :

$$s(x'_k|\boldsymbol{x}, k) = s_v(x'_k|\boldsymbol{x}, k) + \alpha \times s_p(x'_k|\boldsymbol{x}, k)$$

(Zhou et al., 2019).

« α » représente le poids du score de proposition

L'approche de (Zhou et al., 2019) a obtenu des résultats encourageants en anglais. Ils le comparent à WordNet à travers LS07 (jeu de données de SemEval 2007) et LS14 (connu sous le nom du jeu de données CoinCo provenant de [Kremer et al., 2014] cité par [Zhou et al., 2019]) qui sont des jeux de données servant pour étalons dans l'évaluation de la substitution lexicale. Des mesures sont proposées dans ces jeux de données afin d'évaluer la qualité de la substitution. Ils démontrent que leur approche de proposition de substitut est plus efficace pour la substitution lexicale grâce en partie à la contextualisation au niveau de la phrase comparée à l'approche mot par mot de WordNet.

V.5 CamemBERT (Martin et al., 2020) : le dérivé de BERT pour le français.

Maintenant que nous avons vu comment BERT fonctionne, il est intéressant de voir son dérivé pour la langue française : CamemBERT. (Martin et al., 2020). Nous pouvons noter que BERT existe dans une forme multilingue abrégée comme suit : mBert. Mais, un outil spécialisé dans une seule langue [104 langues différentes pour mBERT] sera toujours meilleur qu'un outil multilingue, dans ce cas de figure.

RoBERTa (Liu et al., 2019) est une forme améliorée de Bert qui est plus performante, utilise « dynamic masking » et plus de données d'entraînement également. Cette ressource se concentre de la première phase de préentraînement.

CamemBERT (Martin et al., 2020) peut être utilisé pour un étiquetage morphosyntaxique d'une phrase : il peut prédire l'arbre syntaxique pour déduire les relations syntaxiques entre les mots.

CamemBERT (*ibid.*) prend sa source dans le corpus monolingue (français) OSCAR qui est un ensemble de corpus monolingues extrait de Common Crawl.

CamemBERT (*ibid.*) est aussi un *transformateur bidirectionnel multicouche.* De même, il est très semblable à RoBERTa (Liu et al., 2019), ce qui les différencie est la façon dont les outils cachent les mots : CamemBERT (Martin et al., 2020) cache la totalité du mot, comparé à RoBERTa (Liu et al., 2019) qui cache partiellement les mots.

Tous les deux fonctionnent sur le même principe Masked Language Modeling (MLM : Modélisation Masquée du Langage ; ma traduction) dont découle le Whole Word Masking qui est spécifique à ce modèle pour le français (WWM : Masquage de Mots Entiers ; ma traduction). 15 % des tokens du total de la phrase sont sélectionnés pour être éventuellement remplacés. De ces 15 %, 80 % seront masqués par le token <MASK> ; 10 % intactes ; 10 % remplacés par un token aléatoire (Martin et al., 2020)

D'après (Martin et al., 2020), le rapport de l'évaluation de CamemBERT rapporte une précision 5,6 % plus fidèle que son équivalent multilingue mBert. Ce résultat reste cependant à contraster vu que les deux modèles ont été entraînés sur des jeux de données différents. Malgré cela, nous pouvons constater une amélioration de CamemBERT par rapport à BERT sur les 4 tâches suivantes :

1. Étiquetage morphosyntaxique des tokens.
2. Analyse des dépendances
3. Named Entity Recognition (NER : Reconnaissance des Entités Nommées ; ma traduction)
4. Natural Language Inference (NRI : Inférence en Langue Naturelle ; ma traduction)

Serait-il possible de combiner CamemBERT et FlauBERT dans le but d'améliorer les résultats ?

CamemBERT est le premier modèle de langue à s'intéresser seulement à une langue, le français. Son approche monolingue était novatrice au moment de sa publication par Le et al. (2020). Le modèle a permis l'apparition d'autres modèles monolingues pour le français comme FlauBERT ; qui surpasse CamemBERT sur le jeu de données CLS (Prettenhofer et Stein, 2010) qui est constitué d'avis déposés sur Amazon pour des objets des catégories suivantes : livres, DVD et musique. Le jeu de données est disponible en quatre langues, dont le français. De même (Le et al., 2020) ont démontré que la combinaison de FlauBERT et CamemBERT donnait des résultats encore supérieurs.

V.6 Flaubert : un modèle de langue pour le français

FlauBERT (Le et al., 2020) est un modèle de langue, tout comme CamemBERT, la comparaison est donc intéressante. Le modèle de langue FlauBERT arrive à de meilleurs résultats sur le jeu de données CLS (Prettenhofer et Stein, 2010) d'après la mesure FLUE par rapport CamemBERT. FLUE (French Language Understanding Evaluation | Évaluation de la Compréhension de la Langue Française ; ma traduction) évalue $\text{FlauBERT}_{\text{BASE/LARGE}}$ ainsi que certains de ses homologues comme CamemBERT et mBERT par exemple.

Nous pouvons constater que FlauBERT s'entraîne sur moins de données que CamemBERT (cf. Tableau 1) ; pourtant il obtient de meilleurs résultats. À quoi cela serait-il dû ? Même si les deux outils reposent sur la même langue (le français), ils ont deux manières différentes de fonctionner.

	$\text{BERT}_{\text{BASE}}$	$\text{RoBERTa}_{\text{BASE}}$	CamemBERT	$\text{FlauBERT}_{\text{BASE}}$/$\text{FlauBERT}_{\text{LARGE}}$
Language	English	English	French	French
Training data	13 GB	160 GB	138 GB†	71 GB‡
Pre-training objectives	NSP and MLM	MLM	MLM	MLM
Total parameters	110 M	125 M	110 M	138 M/ 373 M
Tokenizer	WordPiece 30K	BPE 50K	SentencePiece 32K	BPE 50K
Masking strategy	Static + Sub-word masking	Dynamic + Sub-word masking	Dynamic + Whole-word masking	Dynamic + Sub-word masking

†, ‡: 282 GB, 270 GB before filtering/cleaning.

Table 1: Comparison between FlauBERT and previous work.

Tableau 2 : Comparaison entre FlauBERT et de précédents travaux (Le et al., 2020)

La première est le masquage des mots. FlauBERT masque partiellement les mots, contrairement à CamemBERT, qui lui masque les mots cibles au complet, ce qui demande un nombre de données plus conséquent.

Par exemple, le mot « jouant » (participe présent de « jouer ») sera coupé en deux par FlauBERT ; donc : « jou », il ne restera qu'à deviner le suffixe « ant » alors que le modèle devra deviner l'entièreté du mot avec CamemBERT qui utilise le WWM.

La deuxième est que les deux modèles emploient des tokeniseurs différents, ce qui fait qu'ils traitent les données en entrée différemment. Ainsi, leur performance dépend en partie de leur tokeniseur.

Il est également nécessaire de faire la distinction entre $\text{FlauBERT}_{\text{BASE}}$ et $\text{FlauBERT}_{\text{LARGE}}$ (Le et al., 2020) qui ne reposent pas sur le même nombre de paramètres et n'obtiennent donc pas les mêmes résultats. La version reposant sur l'architecture *large* de BERT a tendance à obtenir de meilleurs résultats (cf. Tableau 2).

Le modèle FlauBERT peut également accomplir la tâche de désambiguïsation du sens des mots (WSD : Word Sense Disambiguation) qui consiste à attribuer un sens à des mots dont le modèle ne connaissait pas encore la signification.

$\text{Flaubert}_{\text{LARGE}}$ a obtenu de meilleurs résultats que CamemBERT dans l'analyse de bases de données sur des livres/DVD/Musiques (Le et al., 2020)

Model	Books	DVD	Music
MultiFiT†	91.25	89.55	93.40
mBERT†	86.15	86.90	86.65
CamemBERT	92.30	93.00	94.85
$\text{FlauBERT}_{\text{BASE}}$	93.10	92.45	94.10
$\text{FlauBERT}_{\text{LARGE}}$	**95.00**	**94.10**	**95.85**

† Results reported in (Eisenschlos et al., 2019).

Table 3: Accuracy on the CLS dataset for French.

Tableau 3 : Précision sur un ensemble de données CLS pour le français (Le et al., 2020)

95 % de précision pour les livres contre 92,30 % pour CamemBERT

94,10 % de précision pour les DVD contre 93 % pour CamemBERT

95,85 % de précision pour la musique contre 94,85 % pour CamemBERT

Autre chose importante à noter : nous pouvons clairement remarquer que les modèles monolingues sont beaucoup plus performants que les modèles multilingues [mBERT – MultiFiT] dans la classification de textes.

Pour conclure sur BERT, nous pouvons noter que c'est une architecture qui a été dérivée sous de nombreux modèles (RoBERTa, mBERT, CamemBERT, FlauBERT…) Cela a permis de l'adapter dans différentes langues, car même si certains modèles comme mBERT se veulent multilingues, il reste néanmoins notable que ces modèles multilingues obtiennent des résultats moins bons que leurs semblables monolingues (CamemBERT, FlauBERT). Le premier modèle monolingue consacré uniquement au français a été CamemBERT, il a ouvert la voie à FlauBERT et à beaucoup d'autres modèles monolingues pour d'autres langues. Les résultats obtenus par les modèles français sont très bons, comme en témoigne l'indicateur FLUE (Le et al., 2020). BERT se voit comme un modèle permettant l'accomplissement de nombreuses tâches de base comme les questions — réponses, dans lesquelles BERT prédit la réponse.

BERT est également utilisé dans le cadre de la simplification automatique, car son architecture le rend capable de paraphraser une phrase *input* et d'en sortir une autre paraphrasée en *output.* Elle peut également être utilisée dans le cadre de la substitution lexicale, comme nous l'avons vu avec (Zhou et al., 2019) qui a pour but de remplacer un mot cible tout en cachant le mot original. On pourrait donc imaginer l'appliquer à la simplification automatique en ciblant les mots dits « complexes » pour un lecteur novice dans le but de les remplacer par des synonymes, plus simples, tout en gardant le plus possible le sens original.

VI. Bilan

Écrire cet état de l'art m'a permis de faire le point sur les ressources actuellement disponibles et qui pourront me servir pour mon projet de créer un outil d'annotation sémantique pour le français. À travers les différentes mesures abordées, nous avons pu déceler laquelle sera la plus adaptée à l'évaluation de cet outil : SAMSA (Sulem, Abend, et Rappoport, 2018a). Cette mesure est différente des mesures dites « classiques » qui utilisent des paramètres comme le nombre de mots ajoutés ; supprimés ; etc., mais qui ne prennent pas en compte la structure sémantique ou syntaxique du texte ce qui n'est pas le cas de SAMSA. Par conséquent, nous n'avons pas choisi d'utiliser les mesures présentées dans la partie II. 2 (p. 8), puisqu'elles sont jugées bien trop classiques et trop spécialisées dans certains domaines.

De plus, ces ressources n'ont pas été privilégiées puisqu'il aurait été impossible de créer à la fois un outil d'annotation et d'évaluation en se reposant sur des dictionnaires comme LVF ou DicoValence dans le temps imparti. Ainsi, le schéma UCCA avec le parseur TUPA (et sa mesure SAMSA) sont devenus les ressources privilégiées pour mener à bien l'outil et les autres ressources sont de facto écartées de l'outil vu qu'il n'utilise pas de dictionnaire comme LVF, et les autres ressources présentées. Également, utiliser ce schéma nous a permis de gagner du temps lors de la création de l'outil puisque les bases de celui-ci avaient déjà été créées par d'autres chercheurs. De plus, reprendre un outil et un schéma déjà validés par la communauté scientifique permet de donner une certaine légitimité à notre outil d'annotation.

Le projet ASFALDA, présenté dans la partie II.2, était une alternative solide au schéma UCCA, rappelons-le, le projet se reposait sur un FrameNet francophone, ayant pour objectif d'égaler le FrameNet anglophone. Il aurait pu être une base intéressante pour un outil d'annotation ayant également pour but d'évaluer la simplification d'un texte, mais il n'a pas été retenu en raison de son manque de précision au niveau structurel comparé au schéma UCCA. Cette ressource est actuellement incomplète, ce qui limite son utilité.

Étudier BERT (Devlin et al., 2019) nous a permis de découvrir une architecture capable d'accueillir un éventuel outil d'annotation sémantique grâce à ses dérivés pour le français. Cependant, comme nous le verrons dans les prochaines parties, il va être difficile d'utiliser de tels modèles au vu du manque de ressources.

Notre objectif reste clair et atteignable et dans la partie suivante, nous allons aborder les moyens que nous allons mettre en place pour créer un outil d'annotation sémantique et d'évaluation pour le français.

VII. Méthodologie

Rappelons notre objectif premier : créer un outil qui annote sémantiquement des textes simplifiés afin d'évaluer la simplification opérée sur ces textes. Pour mener à bien cette tâche, nous avons choisi d'utiliser le schéma UCCA avec son outil d'annotation automatique TUPA. Originellement dédié à l'annotation de l'anglais, un modèle bi-LSTM pour annoter des textes français a été mis à disposition, ainsi qu'un modèle BERTm. Par conséquent, nous allons tout d'abord procéder à une analyse comparative des annotations provenant des deux modèles dans la partie IX, ce qui nous permettra de confirmer une certaine performance de la part du modèle BERTm lors de l'annotation de textes en français avec le schéma UCCA.

Concernant TUPA, peu de modifications ont été entreprises sur l'outil directement, si ce n'est de modifier le modèle bi-LSTM qu'il utilise afin d'annoter des textes en français. Quant à SAMSA, la mesure que nous avons décidé d'utiliser pour évaluer les résultats obtenus grâce à l'annotation de TUPA, quelques adaptations ont été effectuées afin que cette mesure évalue les textes français au mieux.

En effet, pour notre outil d'annotation et d'évaluation, nous avons repris une grande partie du fonctionnement originel de la librairie EASSE (Alva-Manchego et al., 2019a). De ce fait, cette librairie permet d'annoter avec TUPA et elle permet également d'évaluer avec SAMSA. Cependant, comme nous le verrons dans la partie X, nous avons dû procéder à de nombreuses adaptations pour que l'alignement effectué par EASSE (avec *stanfordnlp*) puisse aligner correctement les textes en français. L'une des principales adaptations étant de ne plus utiliser la librairie stanfordnlp étant obsolète, en la remplaçant par son homologue plus récent : *Stanza*. Cette modification nous permettra d'aligner correctement les textes en français passés en entrée tout en conservant le fonctionnement global de EASSE. Nous avons également permis l'utilisation du modèle BERTm avec EASSE en modifiant quelques méthodes de l'outil.

Nous avons également procédé de ce fait à un allégement de la librairie afin de garder uniquement la mesure SAMSA et TUPA pour consacrer l'outil à l'annotation avec le schéma UCCA, ce qui permet ultimement de rendre la librairie plus accessible, mais cela permet également de la spécialiser dans l'annotation et l'évaluation sémantique, ce qui était l'objectif initial. Toujours dans l'objectif de rendre Simpeval[8] accessible à tous, nous avons mis tout notre travail sur un dépôt GitHub[9] afin de permettre le téléchargement de la librairie à tous ceux qui le voudraient, mais également pour permettre un suivi de version avancé.

Après avoir effectué ces modifications, nous avons annoté chaque texte du corpus avec Simpeval, ce qui nous a donné un score SAMSA pour chaque texte ainsi qu'un ou plusieurs fichiers XML qui contiennent les annotations effectuées par TUPA. L'annotation des textes a

[8] Nom donné à la librairie EASSE adaptée au français et allégée.
[9] https://github.com/jessy3ric/git_memoire

été effectuée avec le modèle bi-LSTM FR de TUPA, ce qui a pu éventuellement impacter la qualité des résultats.

Pour finir, nous sommes revenus sur les difficultés rencontrées lors de la création de l'outil dans la partie X pour ainsi conclure le mémoire avec la partie XI qui nous a permis de prendre du recul sur le travail effectué dans cette deuxième partie de mémoire.

VIII. Choix des ressources

Il m'a semblé intéressant de revenir sur le choix des ressources afin d'expliquer pourquoi certaines ressources n'ont pas été choisies. Dans cette partie, nous allons surtout aborder les ressources présentes dans la partie II.2 et III. Ressources lexicales pour l'analyse sémantique, car toutes les ressources présentées dans cette première partie de mémoire n'ont pas été sélectionnées dans l'outil définitif qui utilise le schéma UCCA (Abend et Rappoport, 2013a).

Quant à l'outil d'annotation, il émanera directement du schéma choisi, car il a été développé par ceux qui ont créé le schéma UCCA. Ainsi, comme nous le verrons, certaines adaptations seront nécessaires pour qu'il soit utilisé pour le français. Nous allons aborder également la possibilité d'introduire des modèles émergents avec une performance remarquable comme CamemBERT ou FlauBERT. Nous verrons également les difficultés d'une telle adaptation avec un outil qui commence à ne plus être totalement à la page.

Quant aux mesures, nous allons toujours dans la même lignée : celle du schéma UCCA. Il aurait été effectivement moins innovant et surtout moins pratique de devoir d'abord annoter avec TUPA pour ensuite les évaluer avec SARI ou même BLEU, alors même que le corpus choisi avait déjà été évalué par ces mesures. Sachant que SAMSA est innovante dans sa manière d'évaluer la simplification opérée sur un texte, il est intéressant de proposer une évaluation avec cette mesure. De plus, cette mesure pourra nous apporter des conclusions précises avec peu d'adaptations, car elle a déjà été reprise dans certains outils, comme EASSE, qui ont simplifié l'exécution de la mesure, la rendant ainsi plus abordable. Également, notre outil d'annotation et d'évaluation avait pour seul but une portée sémantique et les mesures citées précédemment ne permettaient pas d'apporter une plus-value à notre outil. Ainsi, nous avons allégé l'outil en effaçant toutes les mentions de ces mesures dans les fichiers.

Nous avons également procédé à la modernisation de certaines librairies déjà importées par EASSE comme Stanza avec CoreNLP. Effectivement, la librairie « stanfordNlp » utilisée par EASSE ne permettait pas d'utiliser un modèle français pour l'annotation PoS[10] des textes en entrée. Ainsi, nous avons remplacé *stanfordNlp* par *Stanza* puisque cette dernière nous a permis d'utiliser un modèle monolingue français sans modifier trop de code. Il est important de préciser que TUPA utilise *spaCy* pour annoter les textes et EASSE *Stanza*. Nous n'avons pas modifié l'utilisation de *spaCy* dans TUPA puisque cela aurait rendu impossible l'utilisation de la librairie UCCA, puisqu'elle nécessite *spaCy* également.

L'objectif sera donc d'améliorer un outil déjà existant pour qu'il annote un texte simplifié et son original afin d'évaluer les résultats, dans ce même outil, avec la mesure SAMSA pour ainsi pouvoir tirer de nouvelles conclusions sur la simplification automatique opérée dans le corpus.

[10] PoS : Part of Speech tagging ou étiquetage morphosyntaxique en français.

L’évaluation avec cette mesure permettra d’évaluer la simplification structurelle de la phrase grâce au schéma UCCA.

IX. Étude des outils pour l'annotation avec le schéma UCCA

IX.1 Introduction

Nous allons maintenant aborder la partie pratique du mémoire dans laquelle les outils évoqués précédemment, ainsi que d'autres outils que nous allons introduire, vont être utilisés pour l'annotation et l'évaluation du corpus ALECTOR avec le schéma UCCA (page 15).

Le corpus ALECTOR est un corpus de textes spécialement conçu pour les enfants âgés de 7 à 9 ans, correspondant aux niveaux scolaires CE1 à CM1. Ce corpus de textes parallèles offre une variété de contenus narratifs et documentaires, soigneusement adaptés pour faciliter la lecture et la compréhension des jeunes lecteurs. Ce corpus a été simplifié automatiquement, ce qui veut dire qu'il existe deux versions pour chaque texte : une version « originale », sans traitement particulier et une version simplifiée, qui est donc une version retravaillée dans le but que le texte soit plus compréhensible pour un public plus large (enfants dyslexiques, faibles lecteurs, etc.). Il existe aussi la version du corpus ALECTOR simplifié manuellement, qui a servi de base de comparaison avec la sortie du système ALECTOR. Les deux versions seront annotées avec le schéma sémantique UCCA afin de tirer des conclusions sur la simplification opérée par l'outil de simplification intégré au projet ALECTOR. Il sera alors possible d'améliorer l'outil avec ces retours qui seront principalement d'ordre sémantique. La question principale sera celle du sens : la simplification d'un texte destiné aux enfants a-t-elle altéré le sens global ?

Nous espérons pouvoir répondre à cette question avec les résultats obtenus avec l'évaluation des annotations par la mesure SAMSA.

IX.2.1 Annotation du corpus : TUPA

TUPA (Transition-based UCCA Parser) est une librairie python fonctionnant avec le schéma UCCA qui permet une annotation automatique sémantique des textes. Cet outil, créé par Hershcovich, Abend et Rappoport en 2017, est doté d'un fonctionnement très performant grâce à l'utilisation de modèles Bi-LSTM et BERT multilingues pour obtenir des résultats d'annotations très précis. Dans cette partie, nous allons vous expliquer en détail le fonctionnement de TUPA et comment il peut être utilisé pour améliorer l'annotation d'un texte en utilisant le schéma UCCA.

Fonctionnement de TUPA :

TUPA fonctionne avec un modèle Bi-LSTM qui permet d'annoter automatiquement un texte qui est soit en anglais, soit en français soit en allemand. Ce modèle est performant, car il utilise des plongements lexicaux dynamiques, comme Bert. Les versions Sparse et Mlp sont moins performantes que le modèle Bi-LSTM, comme le montre le tableau 2 dans l'article introduisant TUPA (Hershcovich, Abend, et Rappoport, 2017).

Il est important de noter que des modèles préentraînés sont disponibles pour le français, que ce soit avec un modèle Bi-LSTM ou Bertm.

Output générée par TUPA :

Le résultat de TUPA est un fichier. XML qui représente le schéma DAG (Directed Acyclic Graph) de l'annotation. Il est possible de visualiser ce résultat avec la librairie python « visualize » qui est disponible sur le dépôt git de Daniel Hershcovich. Pour visualiser votre fichier. XML, il est possible d'exécuter la commande « python -m scripts. visualize <xml_files> ». Cette commande permet de visualiser le schéma DAG de l'annotation effectuée par TUPA.

Corpus d'entrainement de TUPA

Le corpus utilisé pour entraîner TUPA en français est « Twenty Thousand Leagues Under the Sea », c'est un corpus monolingue de 20K tokens.

Critiques

Cet outil est plutôt complexe à utiliser, que ce soit pour les problèmes de versions que nous pouvons rencontrer lors de la première exécution du script ou même le manque critique d'information pour faire fonctionner la version Bertm où que ce soit dans les articles publiés ou sur le dépôt GitHub dédié à l'outil.

De plus, la version de l'outil fonctionnant avec le modèle bi-LSTM entraîné sur un corpus de textes français fonctionne avec un modèle spaCy entraîné pour l'annotation POS (Part Of Speech) ainsi que pour les dépendances sur un corpus de textes anglophones. Ainsi, il faudrait changer le modèle utilisé, cependant cette opération peut engendrer des problèmes de versions entre les différents modèles et le code utilisé pour traiter la sortie de spaCy.

Conclusion

En conclusion, TUPA est un outil d'annotation automatique très performant qui fonctionne avec le schéma UCCA. Grâce à l'utilisation de modèles Bi-LSTM et BERT multilingue, TUPA permet d'obtenir des résultats d'annotations très précis pour votre texte. Le fichier. XML de sortie peut être visualisé avec la librairie python « visualize » pour faciliter la compréhension des annotations générées. TUPA a été entraîné avec un corpus français et fonctionne avec le schéma UCCA (Abend et Rappoport, 2013a). Cet outil fonctionne avec Python 3.7, nous

pouvons retrouver la documentation sur le dépôt git[11] de Daniel Hershcovich, le fondateur de l'outil et auteur de l'article sur TUPA.

Qu'en est-il de l'adaptation ? L'outil serait-il vraiment plus performant avec un modèle CamemBERT ?

IX.2.2 Annotation manuelle du corpus : compte rendu

a) Choix du corpus à annoter :

Le corpus choisi est le corpus ALECTOR (Gala et al., 2021). Ce corpus regroupe des textes destinés aux enfants avec leur simplification. Il contient 79 textes originaux avec leur simplification (lexicale, discursive, syntaxique). Il est possible de retrouver les textes et les classer en fonction de leur difficulté sur le site web dédié[12].

b) Exemples d'annotation :

Ex. 1 : Baptiste $_{A}$ prend $_{P}$ sa $_{E}$ douche $_{C}$.

Ex. 2 : Baptiste $_{A}$ est $_{S}$ incroyable.

Ex. 3 : [[Oulaya $_{C}$ et $_{N}$ Samuel $_{C}$] $_{A}$]$_{H}$ [[[ont $_{E}$ acheté $_{C}$] $_{P}$ un $_{E}$ canapé $_{C}$ ensemble]$_{D}$]$_{H.}$

Dans l'exemple 1, le syntagme « douche » est considéré comme le centre de la phrase (et de la Scène). Baptiste fait une action, il prend sa douche, alors le verbe « prendre » est annoté comme un processus (c.-à-d. la relation principale de la Scène qui évolue dans le temps). Alors que dans le deuxième exemple (cf. Ex. 2), c'est un état permanent décrit par le verbe « être » et est donc annoté S.

On peut dégager une tendance à annoter les prénoms/sujets de la phrase comme participants (A). Néanmoins, la catégorie peut également annoter des lieux ou même des entités abstraites. Également, un sujet peut être considéré comme un Centre (C) et être annoté comme participant par rapport à une autre scène (ex 3).

Les verbes transitifs auront tendance à être annotés avec l'étiquette P, car ils désignent un processus.

Les verbes d'état auront tendance à être annotés avec l'étiquette S qui décrit la relation principale d'une Scène qui n'évolue pas dans le temps.

La lettre D (Adverbiale), qui désigne une relation secondaire dans une Scène, sera utilisée pour les relations temporelles, comme indiqué dans le schéma UCCA. Nous pouvons le constater dans l'exemple suivant :

11 https://github.com/danielhers/tupa/

12 https://alectorsite.wordpress.com/corpus/

Exemple : [À la tombée de la nuit] $_{D}$ le hérisson part à la recherche de ses proies sous la haie.

Nous avons préféré découper les unités en 3 éléments : éléments liés à une Scène (regroupe P ; S ; **A** ; D) ; éléments non liés à une Scène (regroupe **C** ; E ; N ; R) ; Autre (U ; F) ; relations entre les Scènes (regroupe **H** ; L ; G ; explication des étiquettes à la page 15). Les catégories sont regroupées dans une seule unité, mais il est possible de changer la catégorie de l'unité à tout moment. La catégorie par défaut est en gras.

Les relations entre les Scènes ont été très peu utilisées hormis le H, car les paragraphes dans le texte sont plutôt courts, alors il est rare d'annoter avec L ou G. Le H est défini comme une Scène Parallèle : une Scène liée à d'autres Scènes par des liens réguliers (ex. : temporalité, logique, but). La plupart du temps, les relations H furent des relations de temporalité, mais également pour de la coréférence. Je justifie ce choix par le fait que les Scènes sont liées entre elles, car elles font référence à l'autre.

L'annotation manuelle a été plutôt longue, car il fallait parfois encadrer la phrase avec A, P, S ou D et également les unités à l'intérieur de la phrase. Pour illustrer ce propos, nous pouvons reprendre le début de l'exemple 3 :

[Oulaya $_{C}$ et $_{N}$ Samuel $_{C}$] $_{A}$

Les deux prénoms sont annotés comme les centres de la Scène A (Participant) et sont reliés par la conjonction de coordination « et » qui est considérée comme un connecteur (N), car la conjonction lie les deux centres.

Il était parfois nécessaire de le faire plusieurs fois dans des phrases contenant plusieurs Scènes, ce qui s'est révélé être chronophage.

<u>Les U</u>, qui ne sont pas intégrés dans le schéma par Abend et Rappoport (2013), semblent correspondre à la ponctuation de la phrase. Sachant que la syntaxe n'est pas la plus importante dans le sens global de la phrase, j'ai décidé d'annoter seulement la ponctuation ! ; « » ; — ; ? ; (), car elle peut transmettre une partie du sens de la phrase.

<u>La lettre F</u> sera utilisée quand le syntagme ne correspondra à aucune catégorie du schéma, c'est souvent le cas quand le syntagme ne modifie pas le sens de la phrase.

Exemple : [Il $_{A}$ [apprécie $_{C}$ aussi $_{F}$ les $_{E}$ œufs $_{C}$ et $_{N}$ les $_{E}$ fruits tombés $_{C}$] $_{S}$] $_{H}$

Ici, le syntagme « aussi » est un adverbe, c'est-à-dire qu'il peut être enlevé de la phrase sans changer le sens global de celle-ci. Ainsi, nous pouvons le considérer comme une Fonction selon le schéma UCCA, ce qui correspond à une unité qui n'introduit pas de relations ni de participants, est seulement requise par le schéma structurel dans lequel elle apparaît.

Ici, « à la tombée de la nuit » peut être considérée comme une relation secondaire dans la Scène puisqu'elle introduit seulement une temporalité.

<u>Les Centres (C)</u> sont des éléments nécessaires à la conceptualisation de l'unité mère. Ainsi, ils sont toujours connectés à une Scène (voir ex 3).

<u>Les Élaborateurs</u> sont des relations hors Scène qui sont connectées à un Centre

Exemple : [[Le $_E$ roi $_C$ déchu $_E$] $_A$] $_H$

Les connecteurs (N) sont des relations hors Scène qui s'appliquent à deux (ou +) Centres, mettant en lumière une caractéristique commune.

Exemple : Il $_A$ [apprécie $_C$ aussi $_F$ les $_E$ œufs $_C$ et $_N$ les $_E$ fruits tombés $_C$] $_S$

Les Relateurs correspondent à tous les autres types de relations hors Scène. Deux sortes : 1) Relateurs qui relie un Centre à une relation hyperonymique et 2) Relateurs qui relient deux Centres relatifs à des aspects différents de l'unité mère.

Exemple trouvé dans la publication de (Abend et Rappoport, 2013 b) :

« Golf$_A$ [became$_E$ a$_E$ passion$_C$] $_P$ [for$_R$ his$_E$ oldest$_E$ daughter$_C$] $_A$ »

Les scènes parallèles (H) correspondent aux étiquettes qui délimitent les scènes entre elles tout en montrant un lien entre celles-ci.

Exemple : [Baptiste $_A$ est $_S$ incroyable,] $_H$ [il $_A$ a sauvé $_P$ une petite fille] $_H$

Les deux scènes ont un lien fort entre elles, elles ne peuvent être séparées comme le montre l'étiquette H qui rend le lien sémantique plus explicite.

Un guide d'annotation plus complet est accessible depuis le dépôt GitHub d'UCCA, qui renvoie particulièrement à la vidéo pour apprendre à annoter avec le schéma UCCA pour l'anglais seulement[13]

Plus de la moitié du corpus a été annoté avec l'outil d'annotation Glozz qui n'était pas l'outil le plus adapté pour l'annotation sémantique d'un texte avec le schéma UCCA. Cependant, il existe un outil d'annotation pour annoter avec le schéma[14] UCCA, mais il n'est pas libre d'accès, nous avons pu obtenir un accès très restreint qui permettait seulement de visualiser les annotations déjà réalisées par les chercheurs — certainement en tant que démonstration — sans pouvoir les modifier ou annoter nos propres textes alors que la fonctionnalité était bien présente. Nous n'avions pas les droits pour accéder à cette fonctionnalité, ce qui fait que nous n'avons pas utiliser l'outil. Néanmoins, l'application INCEpTION[15] semble fonctionner sur les mêmes bases que l'application UCCA, il faudrait seulement paramétrer l'application avec toutes les « étiquettes » du schéma UCCA, ce qui sera fait ultérieurement afin d'annoter le restant du corpus.

c) Configuration de l'outil INCEpTION (Klie et al., 2018) pour l'annotation semi-automatique

INCEpTION est un outil d'annotation de texte puissant et flexible qui facilite l'annotation semi-automatique des données textuelles pour l'apprentissage automatique. Cet outil offre une

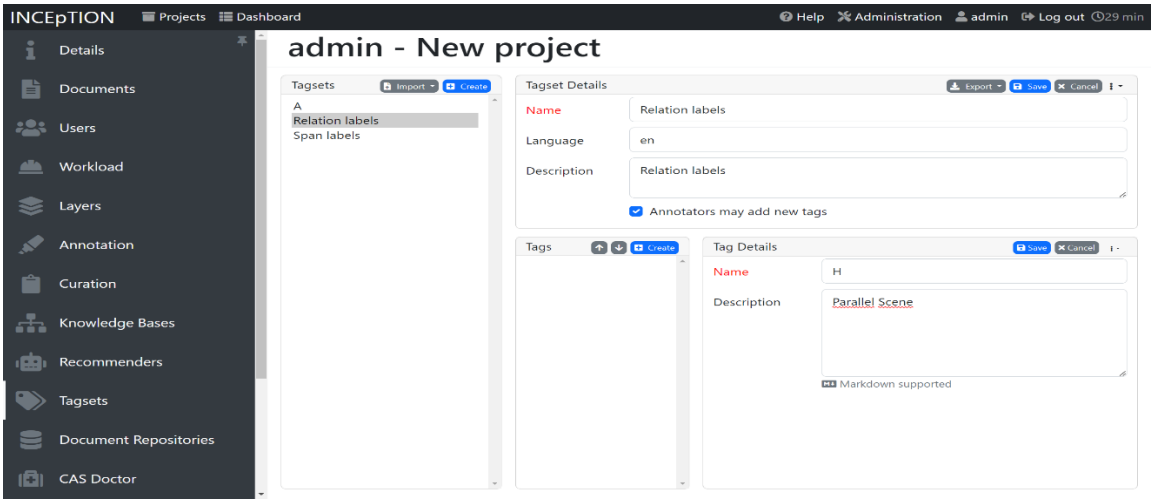
13 https://underline.io/lecture/8427-part-2---annotation-of-english
14 http://ucca-demo.cs.huji.ac.il/
15 https://inception-project.github.io/

interface conviviale permettant de charger et d'annoter facilement des données textuelles avec des entités nommées, des relations et d'autres attributs.

L'une des fonctionnalités clés d'INCEpTION réside dans son approche semi-automatique. Des modèles d'apprentissage automatique préentrainés sont utilisés pour proposer des annotations suggérées, accélérant ainsi le processus d'annotation. Les annotations suggérées peuvent ensuite être révisées et corrigées pour améliorer leur qualité. Une fois l'annotation terminée, les annotations peuvent être exportées dans différents formats couramment utilisés, tels que CoNLL, BRAT, UIMA et GATE, pour être intégrées dans des chaines de traitement du langage naturel et d'apprentissage automatique. INCEpTION permet de gagner du temps et d'optimiser le travail d'annotation. Cet outil performant simplifie le processus d'annotation de texte et maximise la productivité. Les tâches d'annotation ne sont plus un frein dans les projets d'apprentissage automatique.

Il est possible d'utiliser l'outil soit en l'installant localement[16] si JAVA est disponible sur le poste sur lequel il est installé (en exécutant la commande `java — jar inception-app-webapp-28.2-standalone.jar`) soit en ligne[17], sans rien installer.

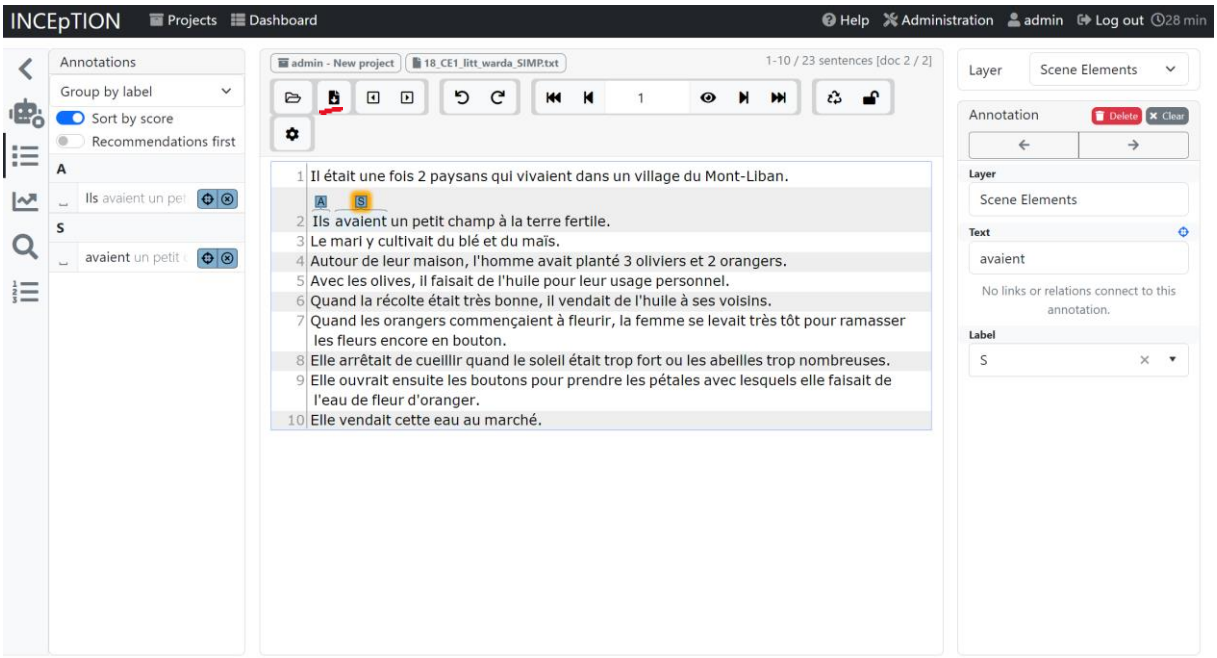
Après avoir installé l'outil et après avoir créé un nouveau projet, nous pouvons créer de nouveaux « tags » afin d'annoter le texte avec des étiquettes personnalisées (ici, venant du schéma UCCA) :



Exemple de création d'étiquettes avec l'outil INCEpTION

Après avoir créé de nouveaux tags pour correspondre au schéma (tags que nous associerons à une « layer » nommée avec le même nom que dans le schéma), nous pourrons ainsi commencer l'annotation semi-automatique (c'est-à-dire que le logiciel suggère seulement des annotations possibles) des différents textes du corpus.

16 https://inception-project.github.io/downloads/
17 https://morbo.ukp.informatik.tu-darmstadt.de/login.html#!d=31470&f=1 (user : demo; mdp: demo)



Exemple d'annotation avec une couche nommée « Scene Elements » qui regroupe les étiquettes P ; S ; A ; D

Une fois que l'annotation est effectuée pour l'ensemble des textes, il ne reste plus qu'à exporter chaque document avec le bouton « export document » souligné en rouge dans la capture d'écran.

Maintenant que nous avons vu comment l'annotation manuelle a été appliquée sur les textes du corpus, nous pouvons voir quelques exemples concrets d'annotations manuelles et automatiques afin de les comparer et éventuellement constater les différences entre mon annotation et celle des différents modèles Bi-LSTM.

IX.2.3 Annotation du corpus automatique : utilisation de TUPA

Dans cette partie, nous allons montrer les étapes nécessaires pour annoter automatiquement avec le modèle français de l'outil TUPA (Hershcovich, Abend, et Rappoport, 2018).

La première étape, la plus évidente, est celle qui consiste à installer TUPA, mais l'environnement qui est fourni évite tous les conflits de version et permet de passer directement à l'annotation. Il faut donc installer l'environnement et l'activer, comme décrit dans la partie X.2 : Création de l'outil (p. 70)

La deuxième étape est de se placer dans le dossier contenant à la fois le modèle français de TUPA et le corpus à annoter.

Ensuite, il vous suffit d'exécuter la commande suivante si vous utilisez TUPA directement :

`python — m tupa corpus_Alector/* — m models/ucca-bilstm-fr --lang fr`

IX.2.4 Annotation du corpus : comparaison des annotations manuelles et automatiques

Pour nous rendre compte des différences d'annotation entre les différents modèles proposés par TUPA, nous avons décidé de prendre une phrase simple en français et de la faire annoter par chacun des modèles présents jusqu'à lors : Bertm ; $TUPA_{Bi\text{-}LSTM}$ FR. Nous avons également jugé intéressant de comparer l'annotation effectuée par le modèle bi-LSTM anglais cette même phrase traduite en anglais.

La phrase que nous allons étudier est la suivante :

« La mort de John est un drame. » « John's death is a tragedy » (version anglaise)

C'est une phrase simple qui ne contient qu'un verbe et qui est centrée autour d'un événement centre « la mort de John ». Ainsi, nous avons quelques attentes quant à l'annotation de cette phrase. La mort est vraisemblablement le centre sémantique de la phrase. Cette scène pourra être annotée soit comme un participant à la scène en respectant la définition suivante « scene units typically in a core syntactic position » (Unités de scène traditionnellement en position syntaxique centrale, ma traduction) soit comme un processus. John, qui lui devra être considéré comme un participant, devra idéalement faire partie de ce centre ou être un centre par lui-même. « Un drame » est un participant dans la scène.

L'annotation acceptée pour « est » sera : S ; car selon la définition du schéma UCCA l'étiquette S correspond à « the main relation of a scene that does not evolve in time » (La relation principale d'une scène qui n'évolue pas dans le temps ; ma traduction), ce qui correspond entièrement au verbe « est » dans cette phrase.

Annotation et représentation manuelle :

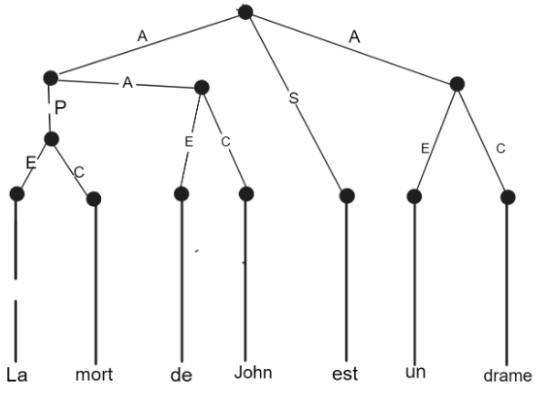


Schéma 1 : annotation manuelle

Remarques sur l'annotation :

- Une scène typée comme Participante regroupe « La mort » et « de John », qui sont ensuite découpées en deux autres scènes : un processus (la mort) et un participant (de

John). Les deux parties de la phrase ont été regroupées dans une même scène puisque nous pouvons tous les deux les considérer comme des participants dans la phrase.

TUPA$_{Bi\text{-}LSTM}$ EN :

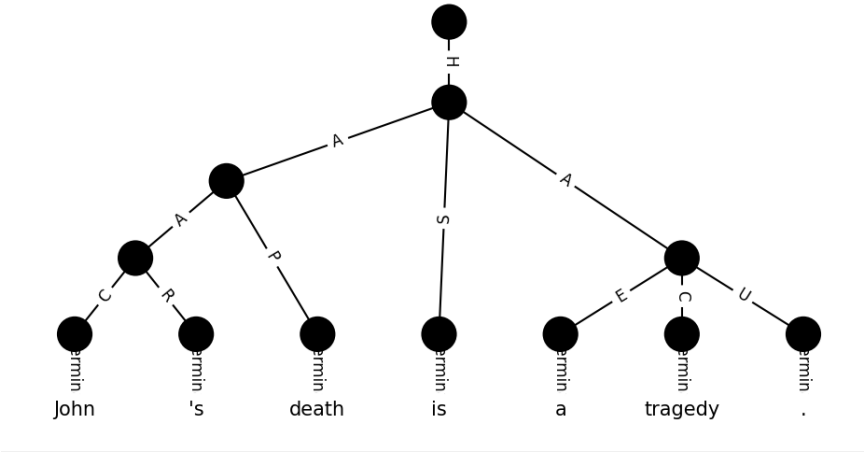


Schéma 2 : annotation automatique TUPA$_{Bi\text{-}LSTM}$ (version EN)

Remarques sur l'annotation :

L'annotation réalisée par le modèle Bi-LSTM de TUPA entraîné sur des textes en anglais montre une vraie performance dans l'annotation des textes en anglais et cela est sûrement dû à la taille du corpus d'entraînement qui est plus importante.

Pour cette phrase, nous pouvons remarquer de très bonnes annotations :

- John est considéré comme un participant, il est le centre de cette scène, ce qui est correct. Le marqueur du génitif « "s » dans « John's death » est considéré comme un relateur, c'est correct puisqu'il lie à la fois « John » et « death ». Il relie deux centres en soi.
- Le verbe « is » (verbe être à la troisième personne du singulier) est annoté comme un état, ce qui correspond totalement à ce que renvoie la phrase. John est mort, c'est un état définitif.
- « Tragedy » est annotée comme un participant, les autres annotations rejoignent celle-ci, dont notre annotation manuelle. De plus, l'article indéfini « a » est annoté comme un élaborateur (E), ce qui rejoint également notre annotation manuelle vu que l'article apporte du sens au centre sémantique de la phrase qui est « tragedy ». C'est une tragédie, mais ce n'est pas LA tragédie du siècle.

Pour résumer, l'annotation réalisée par ce modèle est donc très concluante puisqu'il n'y a aucune erreur.

TUPA$_{Bi\text{-}LSTM}$ FR :

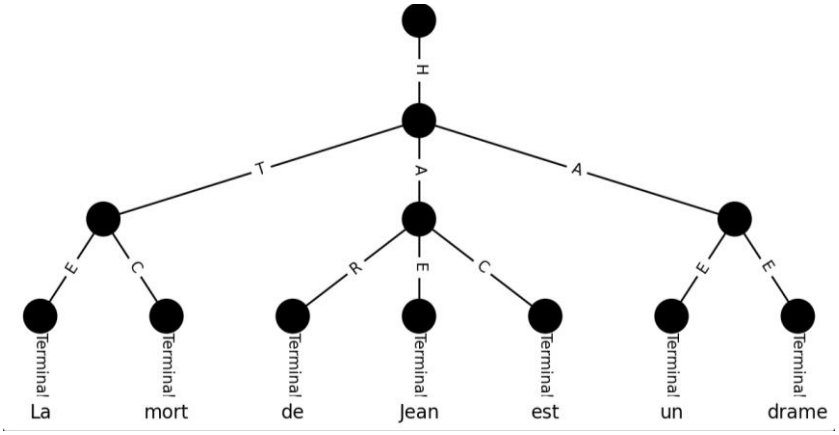


Schéma 2 : annotation automatique TUPA$_{Bi\text{-}LSTM}$ (version FR)

Remarques sur l'annotation :

Nous pouvons remarquer quelques erreurs sur l'annotation par le modèle Bi-LSTM. Nous pouvons tout d'abord apercevoir une annotation inconnue jusqu'à lors « T ». Cette annotation n'est référencée ni dans l'article de recherche introduisant TUPA ni dans le schéma UCCA. Cependant, elle est mentionnée dans le diaporama disponible sur le dépôt Git UCCA[18], qui est d'ailleurs accompagné par une vidéo explicative ;cette unité est comprise dans la catégorie des unités modificatrices. L'unité « T » « exprime le moment ou la fréquence d'un événement sans constituer une scène à part entière » (traduit depuis le Guide d'annotation[3], ma traduction). Ainsi, le fait que la scène « la mort » ait été annotée avec l'unité « T » peut être considéré comme une faute selon le guide.

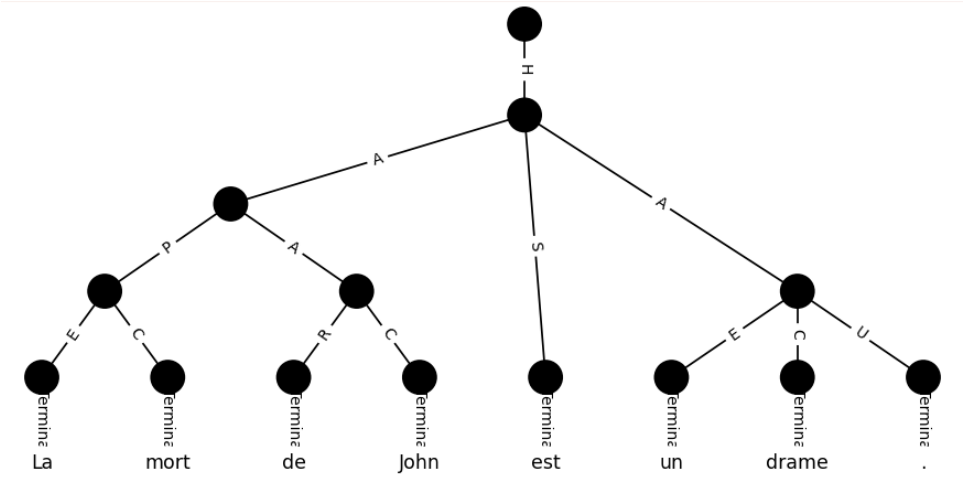
Le reste du schéma est à peu près similaire au nôtre, ce qui est encourageant pour la suite. Les différences à noter sont mineures :

- Commençons sur une note positive : la présence de l'unité « H » qui est utilisée pour la délimitation d'une scène. Cette unité n'est pas présente dans l'annotation manuelle effectuée ni dans l'annotation du modèle bi-LSTM EN alors qu'il est très recommandé d'utiliser cette étiquette pour démontrer un lien entre différentes scènes, quand bien même il n'y en aurait qu'une.
- L'article « de » est annoté comme un « R » donc qui, en théorie, lie deux centres. Ici, ce n'est pas le cas, c'est donc une erreur d'annotation.
- « Jean » est considéré comme l'élaborateur du Centre « est ». Les deux annotations sont possibles.

18 https://github.com/UniversalConceptualCognitiveAnnotation/tutorial/blob/master/02-Annotation.pdf

- L'article « un » et « drame » sont tous les deux annotés comme des Élaborateurs et ils devraient donc être liés à un Centre, ce qui n'est pas le cas. Nous pouvons donc les considérer comme des erreurs d'annotation.
- Le verbe « est » est considéré comme le centre d'une relation Participante, ce qui n'a pas vraiment de sens. Le centre sémantique de cette scène aurait dû être John et pourquoi pas « est » comme un élaborateur. C'est donc une erreur d'annotation

TUPA Bertm :



Remarques sur l'annotation :

L'annotation effectuée par le modèle Bertm est, sans équivoque, la meilleure des trois annotations automatiques proposées.

- Les deux scènes « la mort » et « de John » sont toutes les deux reliées par une scène « supérieure ». Ce qui veut dire que nous ne pouvons pas dissocier les deux dans la phrase, ce qui est totalement juste. Ce qui nous intéresse ce n'est pas la mort en tant que tel, mais la mort de John.
- Le verbe « est » est considéré comme une scène à part entière, comme un « état » (S), ce qui n'est pas complètement faux. Cependant, cela aurait été davantage sensé d'annoter « est un drame » comme un état, car le verbe être n'a pas vraiment de sens par lui-même, il ne peut donc être considéré totalement comme une scène. Nous ne pouvons cependant pas considérer cette catégorisation comme une erreur, surtout que notre annotation manuelle fait état de « un drame » comme une scène Participante (A) de même pour cette annotation automatique.
- La scène « la mort » est considérée comme un Processus (P), ce qui n'est pas le cas ici, car sa mort n'est pas décrite dans les détails alors, nous considérons cette annotation comme une erreur. Ici, il serait juste d'annoter la mort comme un état.

Pour confirmer le niveau de performances du modèle BERTm pour le français, il est nécessaire de comparer l'annotation de deux autres phrases, ce qui nous permettra de tirer des conclusions

sur l'annotation des textes en français opérée par nos deux modèles BERTm et bi-LSTM fr. Nous allons alors prendre, cette fois-ci, des phrases du corpus ALECTOR. La première proviendra du texte « 143_CM1_litt_orig_dragonneminuit », qui correspond à la première des trois parties du texte dragonne minuit.

La phrase que nous allons analyser, en comparant le résultat du modèle bi-LSTM FR et BERTm, est « Juché sur la cime d'un arbre mort ». Les deux visualisations des phrases sont disponibles sur GitHub[19] puisqu'elles sont trop volumineuses pour être insérées ici.

Nous pouvons d'emblée voir une réelle différence entre les deux annotations dans le sens où, celle produite par BERTm est beaucoup mieux structurée que celle produite par le modèle bi-LSTM. Dans l'annotation produite par BERTm, une scène relie entièrement la phrase et est elle-même liée à d'autres scènes au sein de cette même phrase, ce qui est très correct. L'annotation bi-LSTM n'obtient pas les mêmes résultats, les scènes de la phrase sont directement reliées à d'autres scènes qui n'ont pas beaucoup de lien avec elles.

Procédons à une analyse des deux annotations :

BERTm :

- « Juché » est annoté comme un Processus[20], ce qui est une annotation correcte puisque le verbe décrit une action qui est à même d'évoluer dans le temps.
- Ensuite, nous pouvons déceler une autre scène « sur la cime d'un arbre » qui est annotée comme une scène Participante, donc la cime et l'arbre seraient des participants.
    - Dans cette scène, nous pouvons retrouver « cime » comme le centre de celle-ci.
    - Avec pour élaborateur, « d'un arbre », « arbre » étant le centre de cette scène Elaborateur. Toutes ces annotations sont correctes, car la cime est une « extrémité pointue (d'un arbre, d'un rocher, d'une montagne » (dictionnaire LeRobert), les annotations sont donc en raccord avec cette définition.
- 'mort » le mot est considéré comme une scène Adverbiale (D)[21], donc comme une relation secondaire dans une scène (ici, « juché sur la cime d'un arbre mort ») ce qui est correct. Ce détail, le fait que l'arbre soit mort, apporte des détails dans la phrase, mais n'est pas crucial à la compréhension de celle-ci.

Bi-LSTM FR :

- La structure est tout de suite beaucoup moins claire. La phrase « Juché sur la cime d'un arbre mort » est entièrement reliée à une autre scène qui comprend deux autres phrases et le lien avec celles-ci n'a pas beaucoup de sens.
- « cime » le mot est considéré comme le centre de la scène élaborateur « à Brunoît », donc le lien entre les deux serait, selon l'annotation effectuée par le modèle bi-LSTM, un lien de localisation ? La cime se trouverait à Brunoît ? Ce n'est pas ce qui est convoyé dans le texte original. Ainsi, cette annotation est considérée comme fausse.

---

[19] Bi-LSTM FR : https://github.com/jessy3ric/git_memoire/blob/main/corpus/corpus_annoté_bilstm/visualisations/dragonne_minuit_bilstm.png
BERTm :https://github.com/jessy3ric/git_memoire/blob/main/corpus/corpus_annote_bertm/visualisations/dragonne_minuit_screenshot_1.png

[20] Relation principale d'une Scène qui évolue dans le temps (généralement un mouvement ou une action).

[21] Relation secondaire dans une Scène (relations temporelles incluses).

- Le reste de l'annotation n'a pas trop de sens non plus, « mort » est considéré comme le centre d'une scène qui est reliée à « À Brunoît, le roi déchu, à Névé, sa femme, » deux phrases plus loin. Cela n'a donc pas de sens puisque « mort » devrait être relié à « arbre », puisque c'est l'arbre qui est mort, pas le roi.

Pour l'annotation de cette phrase, nous pouvons constater une réelle faiblesse dans l'annotation opérée par le modèle bi-LSTM FR, les liens sémantiques effectués entre les mots et les phrases n'ont effectivement pas trop de sens. Quant à l'annotation effectuée par le modèle BERTm, nous la considérons comme l'annotation parfaite. Il n'y a aucune erreur recensée et le graphe généré est à l'image de l'annotation, beaucoup plus claire. Afin de confirmer cette différence d'annotation, nous allons analyser une phrase provenant d'un autre texte du corpus ALECTOR « 23_CE1_sci_orig_herisson » : « La nuit, le hérisson se balade sous la haie et passe aussi dans ton jardin ».

Procédons à nouveau à une analyse des deux annotations :

BERTm :

La phrase est découpée en deux scènes :

- « La nuit, le hérisson se balade sous la haie » :
  - « La nuit » est considéré comme une scène Participante et au vu de la définition assez abstraite de l'étiquette UCCA[22], nous ne pouvons pas infirmer cette annotation qui semble somme toute assez juste.
  - « le hérisson » est également considéré comme une scène Participante, avec « hérisson » comme centre. Nous pouvons effectivement désigner le hérisson comme un participant de la phrase, l'annotation est donc correcte.
  - « se balade » est considéré comme une scène Processus, qui décrit donc un état qui évolue dans le temps. Le hérisson se balade sous la haie la nuit, pas en journée, cela évolue alors dans le temps.
  - « sous la haie » est une scène Participante puisqu'elle décrit un lieu. L'annotation est correcte.
- « et passe aussi dans ton jardin. » :
  - « et » est relié à une scène très lointaine qui n'a pas forcément de sens ici. C'est une erreur d'annotation. L'adverbe devrait faire le lien entre les deux scènes qui constituent la phrase « La nuit, le hérisson se balade sous la haie et passe aussi dans ton jardin ».
  - « passe » est considéré comme un processus, une scène qui évolue dans le temps, ce qui est correct. Le hérisson est de passage, il ne reste pas indéfiniment sous la haie.
  - « aussi » est considéré comme une relation adverbiale, donc comme une relation secondaire, ce qui est vrai. C'est un adverbe et il peut être enlevé.

[22] Participant dans une Scène, définition assez générale qui regroupe : lieux ; entités abstraites et les Scènes servant d'argument.

    - « dans ton jardin » est considéré comme une scène Participante. Sachant qu'elle décrit un lieu, l'annotation est correcte, « jardin » est le centre sémantique de cette scène.
- Détail important : « hérisson » est relié à la deuxième scène « et passe aussi dans ton jardin », avec un lien « Participant », ce qui est vraiment une très bonne annotation. C'est le hérisson qui passe dans le jardin et ce lien met en lumière la relation sémantique entre les deux scènes.

Bi-LSTM FR :

De nouveau, nous pouvons constater un manque de clarté dans le graphe produit par l'annotation du modèle.

- Tout d'abord, la phrase n'est pas découpée en deux scènes, comme l'a fait BERTm, les scènes de la phrase sont reliées à d'autres scènes plus lointaines qui n'ont pas de lien. Par exemple : « la nuit » est annotée comme un L[23], ce qui n'a aucun sens ici vu qu'elle n'est liée à aucune autre scène parallèle (H). Ainsi, l'annotation est fausse.
- « le hérisson » est annoté comme un T, au lieu d'être un A (Participant). Il devrait être considéré comme un participant. L'annotation est erronée.
- « se balade » est annoté comme un Processus, ce qui est correct. Cette scène est liée à une autre scène qui comporte également « sous la haie ». L'annotation est correcte.
- « sous la haie et passe », ce segment est annoté comme une scène Participante, ce qui aurait été correct si le centre sémantique de la phrase avait été « la haie ». « passe » aurait dû faire partie d'une autre scène Processus.
- « aussi dans ton jardin » est annoté comme une scène Participante, ce qui est correct puisque jardin est un lieu et les lieux peuvent être annotés comme des participants.

L'analyse de cette phrase a encore une fois mis en avant la performance de BERTm, même s'il y avait moins d'erreurs dans l'annotation de la part du modèle bi-LSTMFR. L'annotation effectuée par le premier reste tout de même plus claire, plus précise et ne comporte presque aucune erreur, ce qui fait écho avec l'écart de performances constaté dans la partie V.3. Nous pouvons donc conclure que l'annotation effectuée par BERTm est meilleure que celle effectuée par le modèle bi-LSTM.

Après avoir étudié les différences d'annotation entre les deux modèles, nous pouvons tenter de tirer des conclusions sur l'annotation et l'évaluation apportée par notre outil, qui utilise le modèle Bi-LSTM entraîné sur des textes en français.

[23] Une relation entre deux, ou plus, Hs (ex. : « quand », « si », « dans le but de »).

Annotation et évaluation du corpus en entier :

L'évaluation réalisée par SAMSA sur les textes entièrement annotés par TUPA du corpus ALECTOR a pu mettre en lumière des différences considérables dans la simplification automatique opérée par l'outil qui a performé cette simplification.

En effet, le score maximal a été attribué par SAMSA pour le texte « 144_CM1_litt_SIMP_crinblanc » avec un score de 47 399[24], le deuxième meilleur score étant détenu par « 142_CM1_litt_orig_oeilduloup » avec 24 187. Cette différence de score doit pouvoir se traduire dans la simplification obtenue et effectivement, nous pouvons observer des différences notables :

Dans la version simplifiée du texte 142, plusieurs nuances et éléments des émotions du loup sont omis ou adoucis. Par exemple, dans la version originale, le loup est agacé « M'agace, celui-là… », il pense que le garçon l'intrigue « Ce garçon-là, non. Il reste debout, immobile, silencieux. Seuls ses yeux bougent. », et évoque sa propre patience (« Il se lassera avant moi […] Je suis plus patient que lui. Je suis le loup. ».

Ces nuances sont atténuées dans la version simplifiée, où le loup pense seulement « Il m'énerve, celui-là… » et « Je suis plus patient que lui. Je suis le loup. » De plus, le lien entre la louve morte et le loup se trouve simplifié en « Sa cage vide, car la louve est morte la semaine dernière. », dans la version simplifiée.

La version simplifiée réussit à conserver l'essentiel de l'histoire et des émotions, mais elle perd certaines des subtilités et de la profondeur du langage original. Cependant, elle parvient à rendre le texte plus accessible pour un public plus large.

La simplification opérée sur le texte 144 est meilleure que celle opérée sur le texte 142 puisqu'elle parvient à conserver plus fidèlement l'essence de l'histoire originale tout en la rendant plus accessible. Les nuances et les émotions de Folco sont maintenues, comme son lien profond avec le marais, son sens d'appartenance et son sentiment de liberté. Le texte simplifié parvient à conserver des détails importants, tels que l'utilisation de la perche pour déplacer le bateau dans les eaux peu profondes et la description des chevaux sauvages. Bien que certaines descriptions soient raccourcies, elles préservent encore l'atmosphère et l'environnement du marais de Camargue.

Le texte 144 est donc, selon la mesure SAMSA, la meilleure simplification du corpus que l'outil a pu annoter. Son score très élevé nous permet d'établir un repère par rapport aux autres textes. Les textes comme le 142 qui ont un score dans la tranche 19-24 ont une simplification satisfaisante, mais avec quelques pertes sémantiques importantes et parfois des pertes structurelles.

Certains autres textes comme « 159_CM1_sci_perles » ont obtenu un score très bas : 7 516 qui peut se justifier à la fois par des différences structurelles importantes avec la version originale.

24 Scores à retrouver ici : https://github.com/jessy3ric/git_memoire/blob/main/annexes/samsa_results.txt

Sur le plan structurel, la version simplifiée conserve en grande partie l'organisation du texte original, en réduisant principalement la complexité des phrases et en éliminant certains détails et exemples moins essentiels. Cependant, certaines phrases complexes de la version originale sont décomposées en phrases plus simples dans la version simplifiée, ce qui peut rendre le texte moins fluide.

Sémantiquement, la version simplifiée conserve globalement l'information de base, mais nous pouvons constater des pertes quand il s'agit d'informations spécifiques : la version originale précise que les huîtres sont « des mollusques bivalves », tandis que la version simplifiée ne retient que « animaux avec une coquille s'ouvrant en 2 parties ». Cette réduction élimine la mention de la catégorie taxonomique « mollusques bivalves », qui est importante pour décrire précisément l'animal. De plus, certains termes plus spécifiques sont remplacés par des termes plus généraux, ce qui peut également réduire la précision des informations transmises. Également, nous pouvons constater un changement de vocabulaire et de terminologie :

La version originale mentionne que les huîtres sont « réparties sur de nombreuses mers du globe », tandis que la version simplifiée parle de « réparties sur de nombreuses mers ». La version simplifiée supprime le détail « du globe », ce qui peut réduire la compréhension de l'étendue géographique.

Ainsi, nous pouvons constater des disparités importantes entre des textes simplifiés par le même outil de simplification. Certains textes, comme le texte 159, perdent une certaine richesse lexicale qui nuit à la clarté ou même la précision du texte. D'autres textes, comme le 144, ont moins subi d'altérations par l'outil de simplification. Cette évaluation nous a permis de démontrer que SAMSA permet d'évaluer les différences à la fois sémantiques et structurelles grâce au schéma UCCA, ce qui rend la mesure très intéressante.

La comparaison des annotations venant des différents modèles a démontré que le modèle bi - LSTM FR de TUPA est le meilleur candidat pour être intégré dans notre outil d'annotation et d'évaluation automatique. Néanmoins, même si le modèle Bertm a donné des résultats meilleurs que ceux obtenus avec la version bi-LSTM, nous ne pourrons pas annoter l'entièreté des textes du corpus avec ce modèle du fait du coût intrinsèque à son utilisation.

Dans la partie suivante, nous allons aborder les étapes pour le développement de cet outil qui a à la fois annoté le corpus de textes avec le schéma UCCA grâce à l'outil d'annotation automatique TUPA et son modèle français. TUPA, avec l'aide de SAMSA, a également été chargé d'évaluer les résultats obtenus avec l'annotation, ce qui nous a permis de tirer des conclusions de la simplification opérée sur les textes du corpus Alector.

X. Développement de l'outil « Simpeval » pour l'annotation automatique avec le schéma UCCA :

L'outil développé pour annoter les textes avec le schéma UCCA s'intitule Simpeval qui est le raccourci pour « Simplification évaluation » ou évaluation de la simplification des textes, ici en français. L'outil est disponible sur PyPi[25] ainsi que sur le git dédié à l'outil et aux résultats.

Cet outil fonctionne principalement avec le parser TUPA qui annote donc avec le schéma UCCA (voir détails p. 15). Pour rappel, le schéma permet d'annoter les relations entre les différentes phrases d'un texte. Chaque phrase est considérée comme une scène et dans chaque scène nous pouvons retrouver un ou plusieurs événements. UCCA permet donc de montrer les différences sémantiques entre les phrases, ce qui est plutôt innovant concernant l'annotation de textes français simplifiés. Ainsi, utiliser UCCA dans un outil d'annotation sémantique permet d'analyser automatiquement les différences sémantiques entre la version originale d'un texte et sa version simplifiée. À travers une étude des relations et des différentes scènes dans les deux textes, nous pouvons tirer des conclusions sur sa simplification et cette étude est possible grâce à la mesure semi-automatique SAMSA.

SAMSA, qui est une mesure dédiée à l'évaluation des textes annotés par TUPA, permet d'évaluer ces annotations en fonction de critères sémantiques et structurels. Par exemple, si une phrase simplifiée comporte plusieurs événements, elle recevra un score bas. Elle utilise donc la puissance du schéma UCCA pour tirer des conclusions sur la simplification sémantique et structurelle d'un texte.

Combiner ces outils [UCCA, TUPA, SAMSA] permet de constituer un outil automatique puissant qui repose en très grande partie sur le fonctionnement de la librairie EASSE qui permet l'évaluation de la simplification d'un texte avec différentes mesures. Comme nous le verrons dans la prochaine sous-partie, bien qu'il soit irréprochable dans sa conception, cet outil n'est pas adapté à l'évaluation de la simplification dans une autre langue que l'anglais. Notre tâche, dans les prochaines sous-parties, sera donc de l'adapter au français et de rendre également plus simple l'adaptation avec d'autres langues.

X.1 EASSE (Easier Automatic Sentence Simplification Evaluation) (Alva-Manchego et al., 2019a) :

EASSE[26] (Alva-Manchego et al., 2019b) est un outil qui permet d'évaluer automatiquement un texte avec différentes mesures : SARI (Xu et al., 2016), BLEU (Papineni et al., 2002), FKGL (mesure la lisibilité d'un texte ; FleschKincaid Grade Level) et SAMSA (Sulem, Abend, et Rappoport, 2018a). Il permet de simplifier le processus d'évaluation en calculant lui-même

25 https://pypi.org/project/simpeval/
26 https://github.com/feralvam/easse

toutes les mesures et en effectuant toutes les tâches nécessaires pour évaluer un texte donné, par exemple : l'alignement d'un texte. D'après Alva-Manchego et al. (2019 b), cette simplification du processus d'évaluation passe par le regroupement des différentes mesures dans un seul outil plutôt que de télécharger chaque dépôt Git et de les utiliser individuellement. Cette simplification passe aussi par le regroupement des mesures dans un même langage. Alva-Manchego et al. (2019 b) précisent que certaines mesures ont été programmées dans deux langages différents, il prend l'exemple de SARI qui a été construit en Java et en Python.

Ainsi, la librairie est un atout non négligeable pour mon outil d'annotation. Une grande partie de mon outil reprendra le travail effectué par les développeurs de EASSE, qui ont eu la patience de refactoriser le code de SAMSA, qui est à première vue plutôt difficile à comprendre. Comme le disent Alva-Manchego et al. (2019 b), dans leur article, SAMSA ne facilite pas la tâche aux chercheurs étant donné son manque de documentation ainsi que son manque de chemins relatifs : les développeurs en charge de SAMSA, avaient une fâcheuse tendance à utiliser des chemins absolus, une pratique vivement déconseillée en programmation, car chaque personne qui clonera le dépôt devra aussi changer les chemins.

Pour comprendre comment fonctionne la librairie, nous avons dû effectuer quelques tests. Premièrement, nous avons évalué un corpus test contenant le texte *CM1_sci_orig_nico* (version originale) et sa version simplifiée *CM1_sci_nico_SIMP* à retrouver dans le dossier *corpus_annoté_bilstm*[27], ce sont des textes provenant du corpus ALECTOR.

Voici un exemple de commande à exécuter pour annoter un texte en anglais avec EASSE :

`easse evaluate-t custom --orig_sents_path « %~dp0\corpus_Alector\CE1_orig_nico.txt »-m « bleu, sari, samsa » --sys_sents_path « %~dp0\corpus_Alector\CE1_nico_SIMP.txt » --refs_sents_paths « %~dp0\corpus_Alector\CE1_orig_nico.txt » > easse_report_samsa.txt`

Cette commande m'a permis, sur mon propre environnement anaconda avec TUPA installé ainsi que UCCA, de faire fonctionner l'annotation ainsi que l'évaluation proposée par EASSE. On remarquera un manque de clarté dans l'exécution des commandes, il aurait été appréciable d'avoir un guide encore plus complet, notamment pour les développeurs juniors.

Voici le score obtenu pour le texte CM1_sci_nico (à retrouver dans le fichier easse_report_samsa.txt) :

---

27 https://github.com/jessy3ric/git_memoire/tree/main/corpus/corpus_annot%C3%A9_bilstm

Pourquoi EASSE ?
Lors de mes recherches pour comprendre le fonctionnement de la mesure SAMSA et la manière de l'appliquer après avoir annoté le texte avec le schéma UCCA, j'ai trouvé dans la partie « Issues » du dépôt GitHub de SAMSA[27] un développeur qui demandait des informations supplémentaires pour appliquer SAMSA à son propre outil. Ce développeur, c'était M. Alva-Manchego, le développeur à l'origine de EASSE. Ainsi, il me semble nécessaire d'introduire cet outil qui me servira de base pour mon propre outil d'évaluation de la simplification automatique des textes en français.

{« bleu » : 57 599, « sari » : 23 555, « samsa » : 19 792}

Sur la base des scores SARI, BLEU et SAMSA obtenus pour ce texte, nous pouvons suggérer que le système de simplification de phrases semble avoir une efficacité modérée. Le score SARI de 23,555 suggère que les phrases simplifiées sont relativement précises, exhaustives et fluides par rapport aux phrases d'origine et aux phrases de référence. Le score BLEU de 57,599 indique que la sortie a un niveau de similitude élevé avec les phrases de référence, car plus son score est grand, plus l'entrée et la sortie sont similaires. Enfin, le score SAMSA indique que les phrases simplifiées préservent le sens des phrases d'origine dans une large mesure. Cependant, nous pouvons observer qu'il reste le score le moins élevé entre les trois mesures.

Dans l'ensemble, ces scores suggèrent que le système de simplification de phrases fonctionne bien et produit une sortie qui est efficace pour préserver le sens et la fluidité tout en réduisant la complexité. Cependant, ce ne sont que des conclusions hâtives, car nous n'avons évalué qu'un seul texte pour le moment.

Par conséquent, une analyse et une évaluation supplémentaires de la sortie et des performances du système sont nécessaires pour obtenir une évaluation plus complète de son efficacité.

Dans le but de créer un outil 100 % dédié au schéma UCCA, il est peut-être plus intéressant de reprendre seulement la partie consacrée à l'annotation avec TUPA et l'évaluation avec SAMSA dans EASSE. Quelques modifications seront effectuées, comme l'intégration du modèle TUPA entraîné sur un corpus français ou l'utilisation d'un modèle pour le français pour CoreNLP qui sert à l'alignement dans l'outil, mais le fonctionnement global de EASSE sera gardé (ex : exécution depuis le terminal, annotation POS avec CoreNLP, etc.)

X.2 : Création de l'outil

X.2.1 Programmation d'un script pour SAMSA (Sulem, Abend et Rappoport, 2018a) :

SAMSA est une mesure permettant d'évaluer l'efficacité de la simplification automatique d'un outil. Cette mesure, comme nous l'avons rappelé dans la partie III.1.1 (p.19) est une mesure qui s'intéresse particulièrement à la sémantique structurelle, ce qui diffère de toutes les autres mesures introduites précédemment (ex. : BLEU, SARI, etc.)

SAMSA prend en entrée les graphes acycliques produits par TUPA pour ainsi étudier les différences sémantiques du texte original et du texte simplifié. Grâce aux graphes acycliques, SAMSA (selon (Sulem, Abend et Rappoport, 2018a)) vérifie que chaque phrase contient un seul événement, si tel est le cas, la phrase sera considérée comme une bonne simplification. Toujours selon (Sulem, Abend et Rappoport, 2018a), la mesure vérifie également si la relation principale, et ses participants, de chaque événement est conservée dans la version simplifiée. Si c'est le cas, alors la mesure récompense une telle simplification. Tout cela est possible grâce aux graphes acycliques donnés en entrée par TUPA à la mesure SAMSA. Une fois que les graphes UCCA sont annotés, SAMSA peut calculer la similarité sémantique entre les graphes

de différentes phrases. Cela permet de mesurer la similitude des significations des phrases, même si leur structure grammaticale diffère. De plus, SAMSA a pour but d'associer chaque scène à la phrase correspondante, cela est également réalisé grâce aux graphes acycliques et aux textes donnés en entrée qui sont alignés avec un aligneur de mots que nous étudierons dans la sous-partie suivante.

La mesure fonctionne avec Python, ce qui rend la tâche un peu plus simple que si elle était codée en Java et Python comme SARI. Cependant, comme l'a critiqué Alva-Manchego et al. (2019 b), SAMSA n'est pas facilement exécutable, car les scripts sont présents dans plusieurs fichiers avec des chemins absolus donc il est nécessaire de les regrouper et d'adapter un minimum le code, comme l'ont fait (Alva-Manchego et al. (2019 b)) avec EASSE.

Pour utiliser SAMSA, il est aussi nécessaire d'avoir un aligneur de mots. Dans l'outil EASSE, ils utilisent un modèle entraîné pour l'anglais CORENLP. Le lien pour télécharger les modèles CoreNLP sera changé afin de télécharger également un modèle pour annoter le français. Etant donné EASSE a été publié en 2019, il existe des versions plus récentes pour aligner les mots, dont des versions utilisant la librairie *transformers*. Il sera intéressant d'essayer d'implémenter la version utilisant cette librairie pour voir si elle est compatible avec l'outil. Cette implémentation sera certainement réalisée grâce à la librairie Stanza.

Voici la fonction à modifier pour avoir le bon modèle coreNLP :

```
53
54  def download_stanford_corenlp():
55      url = 'http://nlp.stanford.edu/software/stanford-corenlp-full-2018-10-05.zip'
56      temp_filepath = get_temp_filepath(create=True)
57      download(url, temp_filepath)
58      STANFORD_CORENLP_DIR.mkdir(parents=True, exist_ok=True)
59      unzip(temp_filepath, STANFORD_CORENLP_DIR.parent)
60
```

Fonction pour télécharger le modèle CoreNLP (easse/easse/utils/resources.py)

Cette fonction sera modifiée (voir IX.2.4 ; vers p.68) avec le modèle [15] utilisant la librairie transformers. Ils permettent de meilleurs résultats. Cependant, il se peut qu'il y ait des problèmes de compatibilité avec le code. Dans ce cas, une version plus ancienne[28] sera utilisée.

Le fonctionnement de SAMSA sera presque entièrement repris de EASSE, le fichier intitulé « samsa.py » sera repris ainsi que tous les imports auxquels il fait appel. Ce sera une base conséquente pour notre outil, seulement quelques adaptations seront nécessaires.

28 https://search.maven.org/remotecontent?filepath=edu/stanford/nlp/stanford-corenlp/4.4.0/stanford-corenlp-4.4.0-models-french.jar

Dans ces imports, nous pouvons retrouver les trois fonctions suivantes :

```
27  @lru_cache(maxsize=1)
28  def get_parser():
29      if not UCCA_PARSER_PATH.parent.exists():
30          download_ucca_model()
31      update_ucca_path()
32      with mock_sys_argv(['']):
33          # Need to mock sysargs otherwise the parser will use try to use them and throw an exception
34          return Parser(str(UCCA_PARSER_PATH))
35
36
37  def ucca_parse_texts(texts: List[str]):
38      passages = []
39      for text in texts:
40          passages += list(ucca.convert.from_text(text.split(), tokenized=True))
41      parser = get_parser()
42      parsed_passages = [passage for (passage, *_) in parser.parse(passages, display=False)]
43      return parsed_passages
44
45
46  def get_scenes_ucca(ucca_passage: Passage):
47      return [x for x in ucca_passage.layer('1').all if x.tag == "FN" and x.is_scene()]
48
```

Extrait du fichier : easse/easse/utils/ucca_utils.py

Ces fonctions sont essentielles au bon fonctionnement de SAMSA et ont pour but de :

- « get_parser » : télécharger un modèle préentraîné pour TUPA s'il n'en existe pas un. La fonction retourne un objet de la classe Parser, qui prend en argument le chemin vers le modèle préentraîné.
- « ucca_parse_texts » : prend une liste de textes en entrée et retourne une liste de passages analysés par TUPA en appelant la fonction » get_parser ». La fonction convertit chaque texte en une liste de mots, puis analyse les passages convertis à l'aide de TUPA pour renvoyer une liste de passages analysés.
- « get_scenes_ucca » : cette fonction prend un objet UCCA ucca_passage et retourne une liste de toutes les scènes de cet objet. Pour chaque élément dans la couche « 1″ de l'objet UCCA, la fonction vérifie si l'étiquette est « FN » et si l'élément est une scène. Les éléments qui satisfont à ces deux conditions sont ajoutés à la liste retournée.

Fonction modifiée utils/resources.py l.67 :

```
67      config_json['vocab'] = str(UCCA_DIR / 'vocab/en_core_web_lg.csv')
```

Modification pour la version française du ficher contenant le vocabulaire :

```
67      config_json['vocab'] = str(UCCA_DIR / 'vocab/fr_core_news_md.csv')
68      with open(json_path, 'w') as f:
```

L'ensemble du code peut être mieux appréhendé grâce à la suite de tests disponibles sur le dépôt de l'outil EASSE, chaque fonction est testée scrupuleusement ce qui est très utile quand nous ne pouvons pas utiliser le debugger intégré à VS Code pour comprendre le fonctionnement pas à pas de la mesure. Le code garde tout de même un peu de mystère vu sa complexité à certains endroits, surtout dans la partie de comptabilisation des scènes qui a pour but de tirer des conclusions sur la simplicité du texte annoté.

Après avoir vu quelques changements mineurs, nous allons aborder en profondeur, dans la sous-partie suivante, les changements effectués sur la librairie EASSE afin de l'adapter pour l'annotation du français avec TUPA, et l'alignement aussi, mais également pour alléger la librairie qui n'aura plus qu'une seule mesure pour évaluer les textes : SAMSA.

X.2.2 Modification de EASSE pour créer un outil dédié à SAMSA & TUPA

Précédemment, nous avons déjà appliqué quelques changements à la librairie EASSE afin qu'elle soit le plus adaptée possible à l'annotation de textes en français. Nous avons changé le modèle spaCy ainsi que le modèle corenlp, qui tous deux sont utilisés pour l'annotation des textes.

Maintenant, il est important de faire le tri dans la librairie afin de ne garder que ce qui sera utilisé. Rappelons-le, EASSE est une librairie qui a pour but d'accueillir le plus de mesures d'évaluation de la simplification des textes possible. Ainsi, nous pouvons retrouver des mesures comme BLEU, SARI, etc., qui ne nous seront pas utiles pour notre outil qui sera dédié à l'utilisation du schéma UCCA ; nous pouvons donc faire le tri afin d'alléger la librairie.

Quelques modifications ont été appliquées à l'utilisation de TUPA pour le français. EASSE utilise le modèle préentraîné pour l'anglais, ce qui fait que les évaluations effectuées par la librairie sont de ce fait faussées. Par conséquent, nous avons changé le modèle utilisé. Dans le dossier « resources/tools », nous pouvons retrouver un dossier contenant « ucca-bilstm-1.3.10 » qui contient le modèle préentraîné pour l'anglais ainsi que son vocabulaire dans le dossier adjacent « vocab » avec « en_core_web_lg.csv » qui est un fichier cSv contient des mots extraits de la presse anglophone.

Il était donc important de changer à la fois le modèle et le vocabulaire utilisé par le modèle. Pour changer le modèle, il fallait tout d'abord modifier le fichier contenant les variables constantes puisqu'une des constantes contient le nom du fichier et est ensuite réutilisée dans le reste des scripts concernant UCCA et SAMSA.

```
10  UCCA_DIR = TOOLS_DIR / "ucca-bilstm-1.3.10-fr"
11  UCCA_PARSER_PATH = UCCA_DIR / "models/ucca-bilstm-fr"
```

Ces deux variables contenaient avant le nom du dossier pour le modèle entraîné sur des textes anglais. Ainsi, leurs valeurs ont été modifiées.

De plus, il était nécessaire de modifier l'URL présente dans la fonction « download_ucca_model » pour entraîner le téléchargement du modèle entraîné sur des textes français.

```
def dowload_ucca_model(use_bert=False):
    url = "https://github.com/huji-nlp/tupa/releases/download/v1.3.10/ucca-bilstm-1.3.10-fr.tar.gz"
    if use_bert:
        url = "https://github.com/huji-nlp/tupa/releases/download/v1.4.0/bert_multilingual_layers_4_layers_pooling_weighted_align_sum.tar.gz"
    temp_filepath = get_temp_filepath(create=True)
    download(url, temp_filepath)
    UCCA_DIR.mkdir(
        parents=True, exist_ok=True
    ) if not use_bert else UCCA_BERT_DIR.mkdir(parents=True, exist_ok=True)
    untar(temp_filepath, UCCA_DIR) if not use_bert else untar(
        temp_filepath, UCCA_BERT_DIR
    )
    update_ucca_path(use_bert=use_bert)
```

Une fois ces adaptations faites, nous pouvons déjà constater une amélioration dans le jugement de SAMSA. Auparavant, la mesure jugeait que la simplification opérée sur le texte CE1_Nico_simp.txt n'était pas optimale vu le score attribué : 0,875. Désormais, le score est plus représentatif et est presque aligné avec les autres mesures qui ont évalué ce texte : 19 792.

Dans cette modification, nous pouvons apercevoir un nouvel argument présent dans la fonction « use_bert », qui permet le téléchargement du modèle Bertm et ainsi l'annotation sera effectuée avec ce modèle.

```
    @click.option(
        "--use-bert",
        "-uB",
        is_flag=True,
        default=False,
        help="Use bert to annotate with TUPA.",
    )
    def _evaluate_system_output(*args, **kwargs):
        kwargs["metrics"] = kwargs.pop("metrics").split(",")
        kwargs["use_bert"] = kwargs.pop("use_bert")
        metrics_scores = evaluate_system_output(*args, **kwargs)

        def recursive_round(obj):
```

Pour utiliser Bert, il suffit de rajouter un argument dans la ligne de commande « -uB » ou « --use-bert ». Cela a été rendu possible grâce à l'ajout de l'option ci-dessus dans le fichier cli.py, présent dans le dossier root de la librairie. Cela va déclencher le téléchargement du modèle Bertm s'il n'est pas disponible sur la machine de l'utilisateur. Sinon, l'annotation avec TUPA et le modèle est censée commencer.

Cependant, n'ayant pas de GPU à disposition, nous n'avons pas pu tester cette méthode puisqu'une erreur surgissait à l'exécution sur une machine qui est seulement dotée de CPU. Cette méthode reste donc à tester et certainement à déboguer.

Après avoir effectué toutes les modifications concernant TUPA, que ce soit pour l'adapter à l'annotation de textes français ou permettre l'éventuelle utilisation de Bert pour l'annotation, il

était nécessaire d'adapter l'implémentation de CoreNLP[29] (qui était configuré pour l'anglais) au sein de l'aligneur de mots, en raison des différences linguistiques en termes d'ordre des mots, de morphologie et de vocabulaire entre le français et l'anglais, ce qui impacte la précision de l'alignement des mots dans le contexte des textes français. Les nuances linguistiques propres au français, telles que les articles, l'accord de genre et la conjugaison verbale, nécessitent des ajustements pour garantir un alignement adéquat et des résultats pertinents.

Pour utiliser la mesure SAMSA, il faut aligner les mots du texte d'origine avec les mots du texte simplifié afin de mesurer correctement les changements apportés par l'outil de simplification automatique. C'est le rôle du dossier « aligner » dans EASSE qui est repris dans Simpeval, l'entièreté des modifications est effectuée dans le fichier corenlp_utils[30] qui contient les scripts permettant à la fois de télécharger les modèles CoreNLP (POS, tagger, depparse, etc.) et d'initialiser le serveur CoreNLP pour annoter les phrases.

Également, une fonction pour le téléchargement des modèles CoreNLP a été modifiée :

```
54  def download_stanford_corenlp():
55      url = 'http://nlp.stanford.edu/software/stanford-corenlp-full-2018-10-05.zip'
56      temp_filepath = get_temp_filepath(create=True)
57      download(url, temp_filepath)
58      STANFORD_CORENLP_DIR.mkdir(parents=True, exist_ok=True)
59      unzip(temp_filepath, STANFORD_CORENLP_DIR.parent)
60
```

fonction présente dans le fichier resources.py dans EASSE (dossier : utils)

Décryptons ensemble cette fonction, l'URL est déclarée dans la variable portant le même nom puis un lien temporaire vers ce fichier est créé. Ensuite, la fonction « download » est appelée avec l'URL en argument ainsi que le lien temporaire. Un dossier vide est alors créé avec pour nom la constante « STANFORD_CORENLP_DIR ». Pour finir, la fonction unzip dézippe le fichier dans le dossier tout juste créé.

En somme, la fonction n'est pas mauvaise, mais elle peut être optimisée. De plus, il n'est pas possible de trouver une URL vers un dossier « stanford corenlp full » pour le français (seulement des modèles seuls, sans les fichiers apportés dans la version full). Ainsi, il fallait trouver une alternative : la librairie Stanza.

```
61  def download_stanford_corenlp():
62      """Function using stanza to dowload corenlp models"""
63      STANFORD_CORENLP_DIR.mkdir(parents=True, exist_ok=True)
64      corenlp_dir = str(STANFORD_CORENLP_DIR)
65      os.environ["CORENLP_HOME"] = corenlp_dir
66      stanza.install_corenlp(dir=corenlp_dir)
67      stanza.download_corenlp_models(model="french", version="4.5.4", dir=corenlp_dir)
68
```

29 CoreNLP est une bibliothèque Java pour le traitement du langage naturel (NLP) qui fournit une suite d'outils pour analyser le texte. https://stanfordnlp.github.io/CoreNLP/

30 Fichier dans EASSE : https://github.com/feralvam/easse/blob/master/easse/aligner/corenlp_utils.py
Fichier dans Simpeval : https://github.com/jessy3ric/git_memoire/blob/main/annotation_tool/simpeval/simpeval/aligner/corenlp_utils.py

Donc, le fonctionnement a complètement changé. On commence par créer le dossier puis on le définit comme l'endroit où doit être placé CoreNLP. Alors, il ne reste plus qu'à faire appel à la fonction de la librairie Stanza « install_corenlp » et « download_corenlp_models » pour avoir le serveur CoreNLP ainsi que tous les modèles pour le français. Modifier cette fonction pour intégrer Stanza permet de changer le moins de code possible, en ne changeant pas de librairie par exemple, et permet alors d'éviter le maximum d'erreurs.

Pour finir, il fallait donc modifier la méthode présente dans le fichier corenlp_utils qui se charge d'annoter les textes. Tout d'abord, la version originale de EASSE :

```
122         if with_constituency_parse:
123             corenlp_annotators.append("parse")
124         annotators_properties = {
125             "tokenize.whitespace": not tokenize,
126             "ssplit.eolonly": not sentence_split,
127             "depparse.model": "edu/stanford/nlp/models/parser/nndep/english_SD.gz",
128             "outputFormat": "json",
129         }
130         if not STANFORD_CORENLP_DIR.exists():
131             download_stanford_corenlp()
132         os.environ["CORENLP_HOME"] = str(STANFORD_CORENLP_DIR)
133 
134         parse_results = []
135 
136         with CoreNLPClient(
137             annotators=corenlp_annotators,
138             properties=annotators_properties,
139             threads=40,
140         ) as client:
141             for text in tqdm(texts, disable=(not verbose)):
142                 if isinstance(text, List):
143                     text = " ".join(text)
144                 raw_parse_result = client.annotate(text)
145                 parse_result = format_parser_output(raw_parse_result["sentences"])
```

Ensuite, la version modifiée pour Simpeval :

```
111     def syntactic_parse_texts(
126         if with_constituency_parse:
127             corenlp_annotators.append("parse")
128
129         annotators_properties = {
130             "tokenize.whitespace": not tokenize,
131             "ssplit.eolonly": not sentence_split,
132         }
133
134         if not STANFORD_CORENLP_DIR.exists():
135             download_stanford_corenlp()
136
137         os.environ["CORENLP_HOME"] = str(STANFORD_CORENLP_DIR)
138
139         parse_results = []
140
141         with CoreNLPClient(
142             annotators=corenlp_annotators,
143             properties="french",
144             timeout=15000,
145             memory="6G",
146             endpoint="http://localhost:9000",
147             output_format="json",
148             be_quiet=not verbose,
149             threads=40,
150         ) as client:
151             for text in tqdm(texts, disable=(not verbose)):
152                 if isinstance(text, List):
153                     text = " ".join(text)
154                 raw_parse_result = client.annotate(text, properties=annotators_properties)
155                 parse_result = format_parser_output(raw_parse_result["sentences"])
```

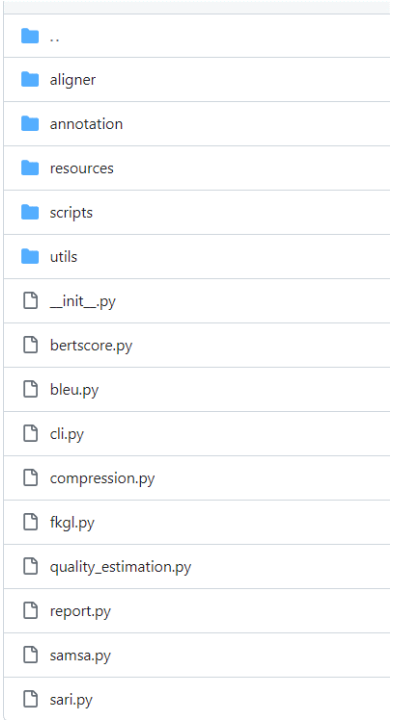
Pour que l'aligneur de mots de EASSE fonctionne encore parfaitement, il fallait que CoreNLP annote les POS, les lemmes, les dépendances ainsi que les NER tout en tokénisant à nouveau le texte en entrée. Tous ces paramètres sont disponibles dans le dictionnaire passé dans le constructeur de la classe CoreNLPClient sous le nom de « corenlp_annotators ».

Cependant, nous pouvons constater des différences au niveau du dictionnaire « annotators_properties » qui est plus simple du fait de la définition de certains arguments directement dans le constructeur de la classe CoreNLPClient. Nous pouvons constater que le dictionnaire n'est plus passé dans le constructeur, mais dans la méthode « annotate », ce qui en soi produit le même résultat, mais il fallait impérativement mettre « french » dans properties pour que le serveur soit lancé en français. Donc, ces adaptations ont permis d'utiliser un modèle français pour l'annotation, en gardant les propriétés de l'annotation pour que la sortie de la méthode « .annotate » soit identique à celle précédemment produite avec la version anglaise.

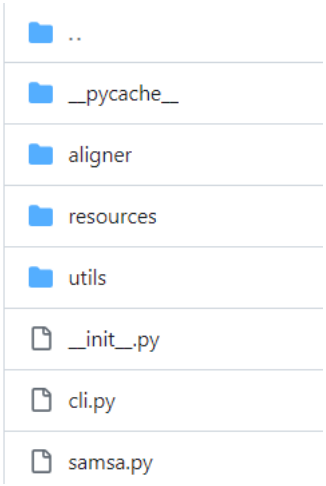
Maintenant que toutes les adaptations pour faire en sorte que EASSE traite les textes en français au mieux ont été effectuées, un allègement de la librairie était nécessaire.

L’allègement devait se compléter par des tests constants pour qu’aucune fonctionnalité nécessaire à SAMSA ne soit impactée. Ainsi, après chaque retrait de fichier, il fallait vérifier si l’outil fonctionnait encore. Tous les fichiers et dossiers qui étaient nécessaires à d’autres mesures que SAMSA ont été retirés. Dans ce principe, nous avons retiré les fichiers pour l’exécution de la mesure BLEU, FKGL, SARI, bertscore, etc. Seul le fichier « cli.py », qui gère l’appel dans la ligne de commande de la librairie, a été gardé dans le dossier simpeval/simpeval, qui contient les fichiers pour exécuter la librairie. Également, le dossier « scripts » a été retiré du fait que Stanza et les derniers modèles CoreNLP gèrent entièrement les NER. Tous les fichiers de tests dans le dossier « root » de la librairie, sauf celui concernant SAMSA, ont été retirés. De plus, le dossier « annotation » qui se trouvait dans le même dossier que le dossier « resources » a été retiré du fait que l’annotation est prise en charge par TUPA, ce qui n’était pas le cas pour les autres mesures. Dans cette même optique, le fichier utils/text.py a été retiré.

Pour résumer les allégements, voici deux images qui montrent un avant après :



Dossier principal de la librairie Easse, chemin : easse/easse



Dossier principal de « Simpeval », chemin : simpeval/simpeval.

Nous avons également enlevé la fonctionnalité qui permettait de produire des rapports, tout d'abord parce que celle-ci était intéressante seulement pour la comparaison qu'elle permettait avec les autres mesures, mais également en raison de son manque de clarté : il était difficile de produire un rapport fonctionnel (en tout cas, pour la mesure SAMSA.) Pour finir, le dossier data, présent dans le dossier « resources » a été retiré vu qu'il contenait des fichiers comme des « test_sets » qui n'étaient pas utiles pour Simpeval.

En guise de résumé, voici une liste à puces des changements effectués à l'outil EASSE pour qu'il devienne Simpeval :

- Changements précédents :
    - Changement du vocabulaire pour qu'il soit adapté à l'annotation du français (« fr_core_news_mb »). Il est nécessaire de garder spaCy pour le bon fonctionnement de la librairie UCCA et donc de TUPA. Nous aurions pu adapter TUPA pour qu'il fonctionne avec Stanza, mais ce choix n'a pas été fait pour éviter de casser certaines fonctionnalités de l'outil. En global, la librairie EASSE a subi plus de changements que TUPA, puisque les seuls changements appliqués à TUPA concernent le modèle spaCy et son modèle bi-LSTM. De plus, il aurait été difficile de modifier à la fois EASSE et TUPA puisqu'il aurait été nécessaire de faire plusieurs dépôts Git, un pour chaque outil. Au vu du peu de changements à appliquer à TUPA, cela n'était pas nécessaire. Changer Stanza pour spaCy lors de l'annotation avec TUPA n'aurait pas fait gagner tant de performances à l'outil. Ainsi, nous avons préféré garder spaCy pour TUPA et nous continuons à utiliser Stanza pour l'évaluation avec SAMSA.
- Travaux réalisés :
    - Réorganisation de la librairie pour ne conserver que les mesures pertinentes pour l'outil dédié à l'utilisation du schéma UCCA.
    - Suppression des mesures telles que BLEU, SARI, etc.
    - Modifications pour TUPA (outil d'annotation de textes) :
    - Adaptations pour l'utilisation de TUPA en français.
    - Remplacement du modèle préentraîné pour l'anglais par un modèle français.

    - Changement du fichier de constantes pour le modèle utilisé.
    - Adaptation de l'URL de téléchargement du modèle pour le français.
    - Nouvel argument « use_bert » pour annoter avec le modèle Bertm.
    - Ajout de constantes pour l'utilisation de Bert.
    - Ajout de l'option « --use-bert » dans le fichier cli.py.

- Changements dans l'implémentation de CoreNLP17 :
    - Changements appliqués sur l'aligneur de mots pour mesurer les changements apportés par l'outil de simplification (SAMSA).
    - Adaptation du fichier corenlp_utils pour télécharger et initialiser CoreNLP avec les modèles français.
    - Utilisation de la librairie Stanza pour les modèles CoreNLP en français.

- Allègement de la librairie :
    - Retrait des fichiers et dossiers non nécessaires à l'outil SAMSA.
    - Suppression des fichiers pour les mesures BLEU, FKGL, SARI, bertscore, etc.
    - Conservation du fichier « cli.py » pour exécuter la librairie.
    - Retrait du dossier « scripts » géré par Stanza et les modèles CoreNLP.
    - Retrait des fichiers de tests sauf ceux liés à SAMSA.
    - Suppression du dossier « annotation » puisque l'annotation est gérée par TUPA.
    - Retrait du fichier utils/text.py.
    - Suppression de la fonctionnalité de production de rapports.

En conclusion, les changements apportés à la librairie EASSE visent à adapter celle-ci à l'annotation de textes en français en utilisant TUPA et des modèles français pour l'annotation avec CoreNLP dans le but d'obtenir le résultat le plus représentatif avec la mesure SAMSA. Pour finir, la librairie a été allégée en supprimant les fonctionnalités et les mesures non nécessaires pour l'outil dédié à l'utilisation du schéma UCCA en français.

Afin que Simpeval soit téléchargeable très facilement par le plus grand nombre, nous avons également pris la décision de le transformer en package pour l'uploader sur PyPi et pour pouvoir l'installer en une commande : « pip install simpeval ». Il ne suffisait plus qu'à créer une release sur GitHub (grâce à un tag de version) et vérifier que la librairie fonctionnait au téléchargement avec pip[31] ou sur GitHub[32].

Après avoir effectué les modifications nécessaires pour rendre l'outil plus adapté au français sans changer sa structure dans la partie suivante, nous allons voir comment changer le modèle dans la partie qui succèdera pour potentiellement entraîner un nouveau modèle avec CamemBERT afin d'améliorer les performances globales de l'outil.

31 La librairie est disponible sur PyPi : https://pypi.org/project/simpeval/
32 Release pour l'outil Simpeval : https://github.com/jessy3ric/git_memoire/releases/tag/v0.1.1

X.2.3 Adaptation de TUPA au modèle CamemBERT

Avant, de documenter la possible installation de CamemBERT dans TUPA, nous devons préciser que cette adaptation ne sera pas utilisée du fait qu'entraîner un modèle BERT demande trop de ressources et de données pour avoir des résultats concluants. Cependant, nous avons ajouté la possibilité d'entraîner le modèle sans le faire de notre côté.

À la suite d'un échange avec Daniel Herschovich, le développeur en charge de TUPA, il a été convenu que la librairie » pytorch_pretrained_bert' utilisée dans l'outil était obsolescente dans le sens où elle ne pouvait pas accueillir les nouveaux modèles de Bert pour le français. Ainsi, il fallait « seulement » remplacer le code de la ligne 62 à 83 dans le fichier tupa/classifiers/nn/**neural_network.py** :

```
62          if self.config.args.use_bert:
63              import torch
64              from pytorch_pretrained_bert import BertTokenizer, BertModel
65              import logging
66
67              self.torch = torch
68              if self.config.args.bert_multilingual is not None:
69                  assert "multilingual" in self.config.args.bert_model
70              logging.basicConfig(level=logging.INFO)
71              is_uncased_model = "uncased" in self.config.args.bert_model
72              self.tokenizer = BertTokenizer.from_pretrained(self.config.args.bert_model, do_lower_case=is_uncased_model)
73              self.bert_model = BertModel.from_pretrained(self.config.args.bert_model)
74              self.bert_model.eval()
75              if self.config.args.bert_gpu:
76                  self.bert_model.to('cuda')
77              self.bert_layers_count = 24 if "large" in self.config.args.bert_model else 12
78              self.bert_embedding_len = 1024 if "large" in self.config.args.bert_model else 768
79
80              self.last_weights = ""
81          else:
82              self.torch = self.tokenizer = self.bert_model = self.bert_layers_count = self.bert_embedding_len = \
83                  self.last_weights = None
84
```

tupa/classifiers/nn/neural_network.py

Le résultat aurait été le suivant :

```
61
62          if self.config.args.use_bert:
63              from transformers import CamembertTokenizer, CamembertModel
64              import torch
65              import logging
66
67              self.torch = torch
68              if self.config.args.bert_multilingual is not None:
69                  assert "multilingual" in self.config.args.bert_model
70              logging.basicConfig(level=logging.INFO)
71              is_uncased_model = "uncased" in self.config.args.bert_model
72              self.tokenizer = CamembertTokenizer.from_pretrained("camembert-base", do_lower_case=is_uncased_model)
73              self.bert_model = CamembertModel.from_pretrained("camembert-base")
74              self.bert_model.eval()
75              self.bert_model.to('cuda')
76              self.bert_layers_count = 12
77              self.bert_embedding_len = 768
78
79              self.last_weights = ""
80          else:
81              self.torch = self.tokenizer = self.bert_model = self.bert_layers_count = self.bert_embedding_len = \
82                  self.last_weights = None
83
```

tupa/classifiers/nn/neural_network.py

Comme nous pouvons le voir, nous avons choisi d'importer la librairie *transformers* avec le modèle CamemBERT et son tokéniseur, car c'est une librairie beaucoup plus récente et puissante que *pytorch_pretrained_bert.* Ces imports permettent d'utiliser CamemBERT lors de l'entraînement de TUPA.

Il était aussi nécessaire d'adapter la suite du code parce que BERT et CamemBERT n'utilisent pas les mêmes tokens :

```
192      def get_bert_embed(self, passage, lang, train=False):
193          orig_tokens = passage
194          bert_tokens = []
195          # Token map will be an int -> int mapping between the `orig_tokens` index and
196          # the `bert_tokens` index.
197          orig_to_tok_map = []
198
199          # Example:
200          # orig_tokens = ["John", "Johanson", "'s",  "house"]
201          # bert_tokens == ["[CLS]", "john", "johan", "##son", "'", "s", "house", "[SEP]"]
202          # orig_to_tok_map == [(1), (2,3), (4,5), (6)]
203
204          bert_tokens.append("[CLS]")
205          for orig_token in orig_tokens:
206              start_token = len(bert_tokens)
207              bert_token = self.tokenizer.tokenize(orig_token)
208              bert_tokens.extend(bert_token)
209              end_token = start_token + len(bert_token)
210              orig_to_tok_map.append(slice(start_token, end_token))
211          bert_tokens.append("[SEP]")
212
```

Ce code contient des tokens spécifiques à BERT [CLS], qui est un token utilisé pour représenter le début d'une séquence, de même pour le token [SEP] qui est utilisé pour séparer deux séquences différentes. On peut retrouver des équivalents pour le modèle CamemBERT comme pour le modèle FlauBERT : <s> (début séquence) et </s> (séparateur).

Le résultat aurait été le suivant :

```
191     def get_bert_embed(self, passage, lang, train=False):
192         orig_tokens = passage
193         camembert_tokens = []
194         # Token map will be an int -> int mapping between the `orig_tokens` index and
195         # the `camembert_tokens` index.
196         orig_to_tok_map = []
197 
198         # Example:
199         # orig_tokens = ["John", "Johanson", "'s",  "house"]
200         # camembert_tokens == ["<s>", "john", "johan", "son", "</s>"]
201         # orig_to_tok_map == [(1), (2,3), (4)]
202 
203         camembert_tokens.append("<s>")
204         for orig_token in orig_tokens:
205             start_token = len(camembert_tokens)
206             camembert_token = self.camembert_tokenizer.tokenize(orig_token)
207             camembert_tokens.extend(camembert_token)
208             end_token = start_token + len(camembert_token)
209             orig_to_tok_map.append(slice(start_token, end_token))
210         camembert_tokens.append("</s>")
211 
212         indexed_tokens = self.camembert_tokenizer.convert_tokens_to_ids(camembert_tokens)
213         tokens_tensor = self.torch.tensor([indexed_tokens])
214         if self.config.args.camembert_gpu:
215             tokens_tensor = tokens_tensor.to('cuda')
216 
217         with self.torch.no_grad():
218             encoded_layers = self.camembert_model(tokens_tensor)[0]
219         assert len(encoded_layers) == self.camembert_layers_count, "Invalid CamemBERT layer count %s" % len(encoded_layers)
220 
```

tupa/classifiers/nn/neural_network.py

Comme indiqué précédemment, le changement principal concerne le type de token utilisé pour indiquer le début et la fin d'une phrase. Il est également important d'indiquer que contrairement à BERT, CamemBERT utilise « SentencePiece » pour effectuer la tokenisation des sous-mots, ce qui signifie qu'il ne cache pas les limites des mots. En d'autres termes, les jetons dans CamemBERT correspondent à des sous-mots ou à des caractères réels, plutôt qu'à des mots entiers ou à des morceaux de mots comme dans BERT. Cela permet à CamemBERT de capturer des informations plus fines sur la langue et d'améliorer potentiellement ses performances dans certaines tâches. Ainsi, les tokens produits par les deux tokéniseurs seront tout de même différents. Cependant, l'adaptation de cette partie du code permet de couvrir les différences entre les tokens.

Ici, on utilise CamemBERT, mais aucun autre changement n'est demandé pour l'utilisation de FlauBERT. En résumé, il suffisait d'adapter le code que nous avons étudié (en remplaçant l'utilisation de BERT par CamemBERT dans le fichier *neural_network.py*) pour que TUPA puisse être entraîné avec un modèle Bert pour le français.

Toutes les adaptations pour accueillir le modèle CamemBERT ayant été faites, TUPA était alors prêt à entraîner un modèle Camembert. De prime abord, nous croyions que cette partie me

permettait d'utiliser directement le modèle CamemBERT sans entraînement ultérieur, mais il est évident que ce n'est pas le cas. J'aurais donc dû entraîner le modèle soit avec mon propre PC, qui ne possède pas les capacités techniques, soit avec Colab.

Ainsi, malgré notre volonté, nous n'avons pas pu utiliser ces modèles lors de l'annotation des textes du corpus ALECTOR. Nous avons annoté les textes avec la version BI-LSTM et tout le travail effectué pour adapter CamemBERT à TUPA, n'a pas pu être utilisé, toujours çà cause du manque de ressources. De plus, nous verrons prochainement toutes les difficultés que cette plateforme et les librairies à installer ont engendrées.

X.3 : Difficultés rencontrées et prise de recul par rapport à « Simpeval » :

X.3.1 : Difficultés rencontrées lors de la création de l'outil

TUPA dans sa version Bi-LSTM, que ce soit avec un modèle préentraîné pour le français ou l'anglais, est plutôt simple à utiliser et plutôt rapide. Une commande à exécuter pour recevoir un fichier XML qu'il est possible de visualiser avec un script compris dans la librairie TUPA (cf. illustrations p.59)

Quant à sa version BERT multilingue, elle est impossible à faire tourner sur un PC qui ne possède pas de GPU (c'est-à-dire un PC qui ne possède pas une carte graphique NVIDIA compatible CUDA) ; n'ayant pas ce matériel à disposition, nous nous sommes tourné vers Google Colab. Cette plateforme permet d'obtenir un GPU rapide gratuitement, c'était donc la solution pour notre problème. À chaque solution, son lot de problèmes : impossible d'installer TUPA et DyNET et il fallait aussi apporter des correctifs au code installé dans les prérequis (cf. la librairie Spotlight). Pour TUPA, il est possible de l'installer manuellement en clonant le repository GIT et en le plaçant dans un Drive Google pour l'importer directement dans le fichier Colab.

Quant à DyNET, nous avons essayé de la faire fonctionner pendant des heures et des heures, mais rien n'y a fait : impossible d'installer DyNET sur Colab, alors même qu'il fonctionne parfaitement sur mon PC personnel, qui lui ne possède pas de GPU. La version dont je disposais, et qui était indiquée dans les prérequis, ne fonctionnait pas sur Colab : il manquait plusieurs classes, ce qui provoquait des erreurs au moment de l'exécution de TUPA. Il était impossible de mener à bien la commande `pip install dynet`, même en précisant la version, du fait de cette erreur : « ERROR: Could not build wheels for dynet, which is required to install pyproject.toml-based projects. » Dans ces conditions, il m'a été impossible d'utiliser TUPA utilisant Bertm pendant une grande partie de la création de l'outil qui accueille TUPA et SAMSA, ce qui est regrettable.

Le problème inhérent à Colab et TUPA était un simple problème de version qui m'a fait perdre un nombre d'heures incalculable. Etant donné qu'il est très difficile de changer la version de Python sur laquelle repose Colab, je ne pouvais pas exécuter ni entraîner mon modèle sur la plateforme. La version 3.7 de Python est un prérequis pour la librairie TUPA puisque c'est

seulement dans cette version que la librairie DyNET fonctionne. Ainsi, je pense qu'il sera difficile d'entraîner un modèle TUPA, avec CamemBERT, pour l'annotation de textes en français avec le schéma UCCA. Dès lors, le but du projet sera de créer un outil pour annoter automatiquement les textes en français avec un modèle préentraîné sur un corpus de textes français assez limité. Nous allons cependant pouvoir annoter une petite partie du corpus avec la version Bertm de TUPA et ce grâce à Amalia Todirascu, la professeure qui encadre ce mémoire. Il sera donc possible de comparer les annotations entre les versions bi-LSTM et la version Bertm de TUPA, des conclusions pourront probablement être tirées de cette analyse comparée. Cette analyse nous montrera le modèle qui est le plus adapté, même si nous n'avons pas la chance d'annoter tout le corpus avec le modèle Bertm, du fait de sa dépendance aux GPU.

X.3.2 Prise de recul par rapport à « Simpeval »

L'outil Simpeval est, pour rappel, un outil entièrement conçu pour l'évaluation de la simplification automatique des textes en français. Il a emprunté la plupart de son architecture à EASSE (Alva-Manchego et al., 2019b), qui est également un outil dédié à cet effet, mais qui ne répondait pas entièrement à nos critères. Les modifications apportées à EASSE ont principalement ciblé la langue : cet outil était presque entièrement dédié à l'annotation et l'évaluation des textes en anglais. De plus, tout ce qui ne concernait pas le schéma UCCA et l'annotation avec celui-ci n'était pas nécessaire dans l'outil final. Alors, tout ce qui concernait les mesures comme BLEU, SARI, fkgl n'est plus présent dans l'outil, qui en effet, contient seulement les fichiers nécessaires à l'exécution de SAMSA & TUPA, ce qui était l'objectif initial.

Ainsi, l'objectif principal a été atteint : créer un outil dédié à l'annotation et à l'évaluation du français. Cependant, nous pouvons émettre quelques réserves concernant cet outil. En effet, il aurait été optimal de pouvoir intégrer BERT avec la version multilingue de celui-ci, Bertm. Nous n'avons pas pu l'utiliser du fait du manque de ressources et cela représente certainement une lacune majeure de l'outil. Dans une période aussi importante pour les modèles comme GPT, Bert, etc., c'est un vrai regret de ne pas avoir pu mettre l'annotation avec Bert comme annotation par défaut pour Simpeval. Nous l'avons vu précédemment, l'annotation avec ce type de modèle permet d'avoir des gains significatifs dans l'annotation et donc la précision de l'évaluation effectuée par SAMSA.

Il aurait également été plus optimal de revoir comment Simpeval est appelée en ligne de commande puisque le fonctionnement a été presque entièrement repris de EASSE ; seule une option a été ajoutée : celle d'utiliser Bertm pour annoter. Il aurait été préférable de simplifier cet appel en ligne de commande puisque certains arguments que la librairie utilise ne sont pas assez explicites dans leur fonctionnement.

Pour finir, il est possible que la librairie Simpeval ne fonctionne pas du premier coup lors de la première installation. Parfois, il manque des librairies alors même qu'elles sont spécifiées dans les prérequis. Également, il nous est arrivé de rencontrer des erreurs lors de l'installation avec « pip install simpeval », la librairie ne voulait plus s'installer correctement. Ces erreurs ont normalement été corrigées dans les derniers commits apportés au dépôt Git.

XI. Conclusion et perspectives

Nous sommes parti de l'intitulé de notre mémoire « Ressources et outils pour l'évaluation des systèmes de simplification automatique des textes » pour développer un outil permettant d'évaluer automatiquement la simplification automatique des textes en français. Tout d'abord, nous avons commencé par un état de l'art qui nous a permis de découvrir toutes les ressources déjà existantes pour l'annotation des textes en français ainsi que de nombreux outils et modèles tels que Bert, CamemBERT permettant une analyse et une annotation automatique des textes. Premièrement, nous pensions développer un outil en partant de zéro avec des dictionnaires comme LVF. Cependant, il aurait été très difficile de reproduire un schéma scientifiquement acceptable et de créer un outil qui annote précisément en fonction de celui-ci. De plus, lors des recherches pour développer un outil permettant d'évaluer les textes simplifiés pour le français, nous avons trouvé le schéma UCCA. Le schéma UCCA a été le candidat parfait : il peut s'appliquer à de nombreuses langues et il possède son outil d'annotation (dédié à l'anglais) ainsi qu'une mesure multilingue. Il était donc évident qu'il fallait choisir ce schéma et l'outil qui en découlait. De plus, plusieurs modèles préentraînés étaient disponibles avec l'outil.

Nous fondant sur ce constat, nous avons essayé de développer un outil reprenant les modèles disponibles ainsi que d'autres candidats potentiels comme CamemBERT ou FlauBERT. Cependant, nous n'avons pas pu les faire fonctionner avec TUPA du fait du manque de ressources (manque de GPU). Comme détaillé dans la partie IX (p.68), nous avons donc choisi le modèle Bi-LSTM préentraîné sur des textes en français. Ce modèle, bien que beaucoup moins performant que Bertm ou même CamemBERT, peut produire des annotations relativement correctes pour des textes en français. Ainsi, nous avons annoté la dizaine de textes disponibles sur GitHub provenant du corpus ALECTOR. Notre but était d'évaluer chaque texte afin de tirer des conclusions sur la simplification opérée sur le corpus donné. Cette tâche a été effectuée dans la partie VIII.2.4 (p.52), nous avons comparé les textes aux extrêmes, c'est-à-dire le texte qui a le meilleur score avec le texte qui a le moins bon score de simplification ainsi que certains textes situés entre les deux. Cette évaluation a pu mettre en lumière certaines différences importantes de simplification entre les textes, certaines simplifications étant très réussies tandis que d'autres simplifications présentaient certaines lacunes sémantiques ou structurelles.

La raison principale qui nous a poussés à choisir le schéma UCCA et la mesure SAMSA au lieu de choisir une mesure comme SARI ou BLEU est sa manière d'annoter les textes. En effet, nous pouvons rappeler que le schéma UCCA annote sémantiquement les textes et permet à la fois une annotation automatique et une évaluation automatique. Cette caractéristique sémantique et structurelle apportée à la fois par UCCA et SAMSA permet de les différencier des mesures classiques qui se contentent d'évaluer les textes syntaxiquement pour la plupart. Ainsi, nous étions motivé pour transposer TUPA, qui annote avec UCCA, pour le français afin d'avoir un outil entièrement dédié à l'annotation sémantique et structurelle du français dans le but d'apporter une nouvelle manière d'évaluer les simplifications automatiques pour le français.

En effet, cette transposition et cette évaluation n'ont pas encore été opérées pour le français, ce qui distingue Simpeval des autres outils. L'outil Simpeval qui trouve sa base dans l'outil EASSE est un outil facile d'utilisation grâce au fichier « demo.bat » présent dans le dossier annexes du dépôt Git. En effet, le fonctionnement de base de EASSE a été repris, ce fonctionnement est loin d'être le plus évident, mais dans le fichier .bat fourni, il suffit de remplacer le nom des textes à annoter et évaluer pour que l'outil fonctionne. Une option pour utiliser Bertm a été rajoutée sans que celle-ci puisse être intégralement testée. Outre quelques problèmes d'utilisations qui peuvent apparaître sans qu'ils aient pu être résolus, la librairie Simpeval fonctionne dans l'ensemble et permet d'évaluer correctement les textes passés en entrée. L'outil pourrait éventuellement être conteneurisé grâce à Docker[33] ce qui le rendrait moins conflictuel puisqu'il suffirait de lancer le conteneur pour évaluer les textes, ce qui éviterait des conflits entre les versions ou même des fichiers manquants parfois. C'est une amélioration qui peut être apportée à l'outil sans pour autant être nécessaire à son fonctionnement.

Ainsi, il nous semble donc possible d'estimer que l'objectif premier de ce mémoire a été rempli puisque l'outil est fonctionnel et que celui-ci, malgré son manque de performances, pourra être utilisé pour évaluer la simplification automatique opérée sur des textes en français.

33 Docker : https://www.docker.com

XII. Bibliographie

Abend O., Rappoport A. « Universal Conceptual Cognitive Annotation (UCCA) ». In : *Proc. 51st Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap.* [En ligne]. *ACL 2013*. Sofia, Bulgaria : Association for Computational Linguistics, 2013a. p. 228-238. Disponible sur : < https://aclanthology.org/P13-1023 > (consulté le 30 décembre 2021)

Abend O., Rappoport A. « Universal Conceptual Cognitive Annotation (UCCA) ». In : *Proc. 51st Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap.* [En ligne]. *ACL 2013*. Sofia, Bulgaria : Association for Computational Linguistics, 2013b. p. 228-238. Disponible sur : < https://aclanthology.org/P13-1023 > (consulté le 30 décembre 2021)

Akbik A., Bergmann T., Blythe D., Rasul K., Schweter S., Vollgraf R. « FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP ». In : *Proc. 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Demonstr.* [En ligne]. Minneapolis, Minnesota : Association for Computational Linguistics, 2019. p. 54-59. Disponible sur : < https://doi.org/10.18653/v1/N19-4010 > (consulté le 2 novembre 2021)

Alva-Manchego F., Martin L., Scarton C., Specia L. « EASSE: Easier Automatic Sentence Simplification Evaluation ». *ArXiv190804567 Cs* [En ligne]. 13 septembre 2019a. Disponible sur : < http://arxiv.org/abs/1908.04567 > (consulté le 2 janvier 2022)

Alva-Manchego F., Martin L., Scarton C., Specia L. « EASSE: Easier Automatic Sentence Simplification Evaluation ». *ArXiv190804567 Cs* [En ligne]. 13 septembre 2019b. Disponible sur : < http://arxiv.org/abs/1908.04567 > (consulté le 2 janvier 2022)

Baker C. F., Fillmore C. J., Lowe J. B. « The Berkeley FrameNet Project ». In : *COLING 1998 Vol. 1 17th Int. Conf. Comput. Linguist.* [En ligne]. *COLING 1998*. [s.l.] : [s.n.], 1998. Disponible sur : < https://aclanthology.org/C98-1013 > (consulté le 5 mai 2022)

Birch A., Abend O., Bojar O., Haddow B. « HUME: Human UCCA-Based Evaluation of Machine Translation ». In : *Proc. 2016 Conf. Empir. Methods Nat. Lang. Process.* [En ligne]. *EMNLP 2016*. Austin, Texas : Association for Computational Linguistics, 2016. p. 1264-1274. Disponible sur : < https://doi.org/10.18653/v1/D16-1134 > (consulté le 24 mars 2022)

Brouwers L., Bernhard D., Ligozat A.-L., François T. « Simplification syntaxique de phrases pour le français (Syntactic Simplification for French Sentences) [in French] ». In : *Proc. Jt. Conf. JEP-TALN-RECITAL 2012 Vol. 2 TALN* [En ligne]. *JEP/TALN/RECITAL 2012*. Grenoble, France : ATALA/AFCP, 2012. p. 211-224. Disponible sur : < https://aclanthology.org/F12-2016 > (consulté le 29 mars 2022)

Candito M., Amsili P., Barque L., Benamara F., De Chalendar G., Djemaa M., Haas P., Huyghe R., Mathieu Y. Y., Muller P., Sagot B., Vieu L. « Developing a French FrameNet: Methodology and First results ». *Asfalda Proj.* 2014. p. 8.

Devlin J., Chang M.-W., Lee K., Toutanova K. « BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding ». In : *Proc. 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Vol. 1 Long Short Pap.* [En ligne]. *NAACL-HLT 2019*. Minneapolis, Minnesota : Association for Computational Linguistics, 2019. p.

4171-4186. Disponible sur : < https://doi.org/10.18653/v1/N19-1423 > (consulté le 2 mars 2022)

Dixon R. M. W. *Basic Linguistic Theory Volume 1: Methodology*. [s.l.] : OUP Oxford, 2009. 398 p.ISBN : 978-0-19-157144-2.

Djemaa M., Candito M., Muller P., Vieu L. « Corpus annotation within the French FrameNet: a domain-by-domain methodology ». 2016. p. 8.

Dubois J., Dubois-Charlier F. « INFORMATIONS_LVF.pdf ». *LVF*. 1997. p. 7.

Dubois J., Dubois-Charlier F. « On the electronicl lexical resources of J. Dubois and F. Dubois-Charlier (Présentation du Dictionnaire Electronique des Mots (DEM) et de Locutions Verbales (LOCVERB) de Jean Dubois et Françoise Dubois-Charlier) [in French] ». Marseille, France : Association pour le Traitement Automatique des Langues (FondamenTAL), 2014. p. 69-73. Disponible sur : < https://aclanthology.org/W14-6401 > (consulté le 14 février 2022)

Dubois-Charlier. « Les verbes français (LVF) en format XML | RALI ». In : *RALI* [En ligne]. [s.l.] : [s.n.], [s.d.]. Disponible sur : < http://rali.iro.umontreal.ca/rali/fr/LVF-XML#consultation > (consulté le 9 mai 2022)

François T., Billami M. B., Gala N., Bernhard D. « Bleu, contusion, ecchymose : tri automatique de synonymes en fonction de leur difficulté de lecture et compréhension ». In : *JEP-TALN-RECITAL 2016* [En ligne]. Paris, France : [s.n.], 2016. p. 15-28. Disponible sur : < https://hal.archives-ouvertes.fr/hal-01346538 > (consulté le 2 mai 2022)

Francois T., Gala N., Watrin P., Fairon C. « FLELex: a graded lexical resource for French foreign learners ». 2014. p. 8.

Gala N., Bernhard D., Todirascu A., JAVOUREY-DREVET L. « RECOMMANDATIONS pour des transformations de textes en français afin d'améliorer leur lisibilité et leur compréhension ». *ALECTOR* [En ligne]. novembre 2020. Disponible sur : < https://alectorsite.files.wordpress.com/2020/11/guidelines-linguistiques_alector_final.pdf > (consulté le 2 mai 2022)

Gala N., Javourey-Drevet L. « Mots « faciles » et mots « difficiles » dans ReSyf : un outil pour la didactique du lexique mobilisant polysémie, synonymie et complexité ». *Lidil Rev. Linguist. Didact. Lang.* [En ligne]. 3 novembre 2020. n°62,. Disponible sur : < https://doi.org/10.4000/lidil.8373 > (consulté le 2 mai 2022)

Gala N., Todirascu A., Francois T., Javourey-Drevet L. *Rapport final du projet ANR ALECTOR (Aide à la LECTure pour améliORer l'accès aux documents pour enfants dyslexiques)* [En ligne]. [s.l.] : Agence Nationale de la Recherche, 2021. Disponible sur : < https://hal.archives-ouvertes.fr/hal-03361468 > (consulté le 23 octobre 2022)

Glavaš G., Štajner S. « Simplifying Lexical Simplification: Do We Need Simplified Corpora? ». In : *Proc. 53rd Annu. Meet. Assoc. Comput. Linguist. 7th Int. Jt. Conf. Nat. Lang. Process. Vol. 2 Short Pap.* [En ligne]. *ACL-IJCNLP 2015*. Beijing, China : Association for Computational Linguistics, 2015. p. 63-68. Disponible sur : < https://doi.org/10.3115/v1/P15-2011 > (consulté le 5 mai 2022)

Guillaume B., Fort K., Perrier G., Bédaride P. « Mapping the Lexique des Verbes du Français (Lexicon of French Verbs) to a NLP Lexicon using Examples ». *Eur. Lang. Resour. Assoc. ELRA*. mai 2014. p. 2806-2810.

Hershcovich D., Abend O., Rappoport A. « A Transition-Based Directed Acyclic Graph Parser for UCCA ». In : *Proc. 55th Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap.* [En ligne]. *ACL 2017*. Vancouver, Canada : Association for Computational Linguistics, 2017. p. 1127-1138. Disponible sur : < https://doi.org/10.18653/v1/P17-1104 > (consulté le 5 mai 2022)

Hershcovich D., Abend O., Rappoport A. « Multitask Parsing Across Semantic Representations ». In : *Proc. 56th Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap.* [En ligne]. *ACL 2018*. Melbourne, Australia : Association for Computational Linguistics, 2018. p. 373-385. Disponible sur : < https://doi.org/10.18653/v1/P18-1035 > (consulté le 16 septembre 2022)

Hershcovich D., Arviv O. « TUPA at MRP 2019: A Multi-Task Baseline System ». In : *Proc. Shar. Task Cross-Framew. Mean. Represent. Parsing 2019 Conf. Nat. Lang. Learn.* [En ligne]. *CoNLL 2019*. Hong Kong : Association for Computational Linguistics, 2019. p. 28-39. Disponible sur : < https://doi.org/10.18653/v1/K19-2002 > (consulté le 17 septembre 2023)

Hochreiter S., Jürgen S. « Long Short-Term Memory ». *Neural Comput. 9817351780 1997* [En ligne]. 1997. Disponible sur : < https://www.bioinf.jku.at/publications/older/2604.pdf >

Huang Z., Xu W., Yu K. *Bidirectional LSTM-CRF Models for Sequence Tagging* [En ligne]. 9 août 2015. Disponible sur : < http://arxiv.org/abs/1508.01991 > (consulté le 17 septembre 2023)

Kingsbury P., Palmer M. « From TreeBank to PropBank ». *Proc. 3rd Int. Conf. Lang. Resour. Eval. LREC-2002*. 2002. p. 5.

Klie J.-C., Bugert M., Boullosa B., De Castilho R. E., Gurevych I. « The INCEpTION Platform: Machine-Assisted and Knowledge-Oriented Interactive Annotation ». 2018. p. 5.

Kupść A., Haas P., Marin R., Balvet A. « Towards TreeLex++: Syntactico-Semantic Lexical Resource for French ». In : *Lang. Technol. Conf.* [En ligne]. Poznan, Poland : [s.n.], 2019. p. 6. Disponible sur : < https://hal.archives-ouvertes.fr/hal-02120183 > (consulté le 2 janvier 2022)

Le H., Vial L., Frej J., Segonne V., Coavoux M., Lecouteux B., Allauzen A., Crabbé B., Besacier L., Schwab D. « FlauBERT: Unsupervised Language Model Pre-training for French ». *ArXiv191205372 Cs* [En ligne]. 12 mars 2020. Disponible sur : < http://arxiv.org/abs/1912.05372 > (consulté le 4 mars 2022)

Lété B., Sprenger-Charolles L., Colé P. « MANULEX: A grade-level lexical database from French elementary school readers ». *Behav. Res. Methods Instrum. Comput.* [En ligne]. février 2004. Vol. 36, n°1, p. 156-166. Disponible sur : < https://doi.org/10.3758/BF03195560 >

Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L., Stoyanov V. « RoBERTa: A Robustly Optimized BERT Pretraining Approach ». *ArXiv190711692 Cs* [En ligne]. 26 juillet 2019. Disponible sur : < http://arxiv.org/abs/1907.11692 > (consulté le 23 mars 2022)

Loureiro D., Jorge A. « Language Modelling Makes Sense: Propagating Representations through WordNet for Full-Coverage Word Sense Disambiguation ». In : *Proc. 57th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. *ACL 2019*. Florence, Italy : Association for

Computational Linguistics, 2019. p. 5682-5691. Disponible sur : < https://doi.org/10.18653/v1/P19-1569 > (consulté le 20 mars 2022)

Martin L., Muller B., Suárez P. J. O., Dupont Y., Romary L., De la Clergerie É. V., Seddah D., Sagot B. « CamemBERT: a Tasty French Language Model ». *Proc. 58th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. 2020. p. 7203-7219. Disponible sur : < https://doi.org/10.18653/v1/2020.acl-main.645 >

McCarthy D., Navigli R. « The English lexical substitution task ». *Lang. Resour. Eval.* [En ligne]. juin 2009. Vol. 43, n°2, p. 139-159. Disponible sur : < https://doi.org/10.1007/s10579-009-9084-1 >

Mertens P. « Le dictionnaire de valence DICOVALENCE : manuel d'utilisation ». 25 juin 2010. Disponible sur : < https://d1wqtxts1xzle7.cloudfront.net/8217254/manuel_100625-with-cover-page-v2.pdf?Expires=1644839731&Signature=HB~VkPpWA8IsMOY5t2w0Be4rn1Pwn8RjrispefKyhve9YJkCLuT~cPjL8T06pxAFG3TRT9xUFSee6QeiXauFsakmi1hU2WgptgmiDVY3rnPrE855BIpN73jvtEb1z0XsGwNaHkTvSx31zbdfJHLmMjefrDNx0PQrp076EtVEQB6PBCwC27pT31Ozzhd3n-X4R5iMeCRgPi3GC9RUrWOXKYXid7pOTshRyYZiFcSumCqebnhtxXtutL1pDTByQPsITkMVNLUUGdCpIsUivmlr9-l8i4-qSrYGd1Ia7gydPpwSWpF5IMXv3PnRi-gfibrloTLZ3QI6VOJ~k9Hv2UR9TA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA > (consulté le 14 février 2022)

Olah C. « Understanding LSTM Networks -- colah's blog ». [s.l.] : [s.n.], [s.d.]. Disponible sur : < https://colah.github.io/posts/2015-08-Understanding-LSTMs/ > (consulté le 17 septembre 2023)

Papineni K., Roukos S., Ward T., Zhu W.-J. « Bleu: a Method for Automatic Evaluation of Machine Translation ». In : *Proc. 40th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. *ACL 2002*. Philadelphia, Pennsylvania, USA : Association for Computational Linguistics, 2002. p. 311-318. Disponible sur : < https://doi.org/10.3115/1073083.1073135 > (consulté le 30 décembre 2021)

Prettenhofer P., Stein B. « Cross-Language Text Classification Using Structural Correspondence Learning ». In : *Proc. 48th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. *ACL 2010*. Uppsala, Sweden : Association for Computational Linguistics, 2010. p. 1118-1127. Disponible sur : < https://aclanthology.org/P10-1114 > (consulté le 5 mai 2022)

Quiniou S., Daille B. « Towards a Diagnosis of Textual Difficulties for Children with Dyslexia ». In : *11th Int. Conf. Lang. Resour. Eval. LREC* [En ligne]. Miyazaki, Japan : [s.n.], 2018. Disponible sur : < https://hal.archives-ouvertes.fr/hal-01737726 > (consulté le 9 mai 2022)

Saggion H. « Automatic Text Simplification ». *Synth. Lect. Hum. Lang. Technol.* [En ligne]. 25 avril 2017. Vol. 10, n°1, p. 1-137. Disponible sur : < https://doi.org/10.2200/S00700ED1V01Y201602HLT032 >

Schnedecker C. *Nom propre et chaînes de référence* [En ligne]. [s.l.] : Librairie KLINCKSIECK, 1997. 231 p.(Recherches linguistiques). Disponible sur : < https://hal.archives-ouvertes.fr/hal-00808797 > (consulté le 9 mai 2022)

Shardlow M. « A Survey of Automated Text Simplification ». *Int. J. Adv. Comput. Sci. Appl.* [En ligne]. 2014. Vol. 4, n°1,. Disponible sur : < https://doi.org/10.14569/SpecialIssue.2014.040109 > (consulté le 18 février 2022)

Štajner S., Béchara H., Saggion H. « A Deeper Exploration of the Standard PB-SMT Approach to Text Simplification and its Evaluation ». In : *Proc. 53rd Annu. Meet. Assoc. Comput. Linguist. 7th Int. Jt. Conf. Nat. Lang. Process. Vol. 2 Short Pap.* [En ligne]. *ACL-IJCNLP 2015*. Beijing, China : Association for Computational Linguistics, 2015. p. 823-828. Disponible sur : < https://doi.org/10.3115/v1/P15-2135 > (consulté le 5 mai 2022)

Sulem E., Abend O., Rappoport A. « Semantic Structural Evaluation for Text Simplification ». In : *Proc. 2018 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Vol. 1 Long Pap.* [En ligne]. *NAACL-HLT 2018*. New Orleans, Louisiana : Association for Computational Linguistics, 2018a. p. 685-696. Disponible sur : < https://doi.org/10.18653/v1/N18-1063 > (consulté le 13 mars 2022)

Sulem E., Abend O., Rappoport A. « BLEU is Not Suitable for the Evaluation of Text Simplification ». *ArXiv181005995 Cs* [En ligne]. 14 octobre 2018b. Disponible sur : < http://arxiv.org/abs/1810.05995 > (consulté le 2 janvier 2022)

Sulem E., Abend O., Rappoport A. « Conceptual Annotations Preserve Structure Across Translations: A French-English Case Study ». In : *Proc. 1st Workshop Semant.-Driven Stat. Mach. Transl. S2MT 2015* [En ligne]. Beijing, China : Association for Computational Linguistics, 2015. p. 11-22. Disponible sur : < https://doi.org/10.18653/v1/W15-3502 > (consulté le 22 décembre 2021)

Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. « Attention is All you Need ». In : *Adv. Neural Inf. Process. Syst.* [En ligne]. [s.l.] : Curran Associates, Inc., 2017. Disponible sur : < https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html > (consulté le 7 mars 2022)

Wang A., Singh A., Michael J., Hill F., Levy O., Bowman S. R. « GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding ». *ArXiv180407461 Cs* [En ligne]. 22 février 2019. Disponible sur : < http://arxiv.org/abs/1804.07461 > (consulté le 20 mars 2022)

Wilkens R., Todirascu A. « Un corpus d'évaluation pour un système de simplification discursive (An Evaluation Corpus for Automatic Discourse Simplification) ». In : *Actes 6e Conférence Conjointe Journ. DÉtudes Sur Parole JEP 33e Édition Trait. Autom. Lang. Nat. TALN 27e Édition Rencontre Étud. Cherch. En Inform. Pour Trait. Autom. Lang. RÉCITAL 22e Édition Vol. 2 Trait. Autom. Lang. Nat.* [En ligne]. *JEP/TALN/RECITAL 2020*. Nancy, France : ATALA et AFCP, 2020. p. 361-369. Disponible sur : < https://aclanthology.org/2020.jeptalnrecital-taln.36 > (consulté le 4 mai 2022)

Xu W., Napoles C., Pavlick E., Chen Q., Callison-Burch C. « Optimizing Statistical Machine Translation for Text Simplification ». *Trans. Assoc. Comput. Linguist.* [En ligne]. 1 juillet 2016. Vol. 4, p. 401-415. Disponible sur : < https://doi.org/10.1162/tacl_a_00107 >

Zhou W., Ge T., Xu K., Wei F., Zhou M. « BERT-based Lexical Substitution ». In : *Proc. 57th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. *ACL 2019*. Florence, Italy : Association for Computational Linguistics, 2019. p. 3368-3373. Disponible sur : < https://doi.org/10.18653/v1/P19-1328 > (consulté le 19 février 2022)

Abend O., Rappoport A. « Universal Conceptual Cognitive Annotation (UCCA) ». In : *Proc. 51st Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap.* [En ligne]. *ACL 2013*. Sofia, Bulgaria : Association for Computational Linguistics, 2013b. p. 228-238. Disponible sur : < https://aclanthology.org/P13-1023 > (consulté le 30 décembre 2021)

Akbik A., Bergmann T., Blythe D., Rasul K., Schweter S., Vollgraf R. « FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP ». In : *Proc. 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Demonstr.* [En ligne]. Minneapolis, Minnesota : Association for Computational Linguistics, 2019. p. 54-59. Disponible sur : < https://doi.org/10.18653/v1/N19-4010 > (consulté le 2 novembre 2021)

Alva-Manchego F., Martin L., Scarton C., Specia L. « EASSE: Easier Automatic Sentence Simplification Evaluation ». *ArXiv190804567 Cs* [En ligne]. 13 septembre 2019a. Disponible sur : < http://arxiv.org/abs/1908.04567 > (consulté le 2 janvier 2022)

Alva-Manchego F., Martin L., Scarton C., Specia L. « EASSE: Easier Automatic Sentence Simplification Evaluation ». *ArXiv190804567 Cs* [En ligne]. 13 septembre 2019b. Disponible sur : < http://arxiv.org/abs/1908.04567 > (consulté le 2 janvier 2022)

Baker C. F., Fillmore C. J., Lowe J. B. « The Berkeley FrameNet Project ». In : *COLING 1998 Vol. 1 17th Int. Conf. Comput. Linguist.* [En ligne]. *COLING 1998*. [s.l.] : [s.n.], 1998. Disponible sur : < https://aclanthology.org/C98-1013 > (consulté le 5 mai 2022)

Birch A., Abend O., Bojar O., Haddow B. « HUME: Human UCCA-Based Evaluation of Machine Translation ». In : *Proc. 2016 Conf. Empir. Methods Nat. Lang. Process.* [En ligne]. *EMNLP 2016*. Austin, Texas : Association for Computational Linguistics, 2016. p. 1264-1274. Disponible sur : < https://doi.org/10.18653/v1/D16-1134 > (consulté le 24 mars 2022)

Brouwers L., Bernhard D., Ligozat A.-L., François T. « Simplification syntaxique de phrases pour le français (Syntactic Simplification for French Sentences) [in French] ». In : *Proc. Jt. Conf. JEP-TALN-RECITAL 2012 Vol. 2 TALN* [En ligne]. *JEP/TALN/RECITAL 2012*. Grenoble, France : ATALA/AFCP, 2012. p. 211-224. Disponible sur : < https://aclanthology.org/F12-2016 > (consulté le 29 mars 2022)

CamemBERT. « CamemBERT ». In : *CamemBERT* [En ligne]. [s.l.] : [s.n.], [s.d.]. Disponible sur : < https://camembert-model.fr/ > (consulté le 13 mai 2022)

Candito M., Amsili P., Barque L., Benamara F., De Chalendar G., Djemaa M., Haas P., Huyghe R., Mathieu Y. Y., Muller P., Sagot B., Vieu L. « Developing a French FrameNet: Methodology and First results ». *Asfalda Proj.* 2014. p. 8.

Devlin J., Chang M.-W., Lee K., Toutanova K. « BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding ». In : *Proc. 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Vol. 1 Long Short Pap.* [En ligne]. *NAACL-HLT 2019*. Minneapolis, Minnesota : Association for Computational Linguistics, 2019. p. 4171-4186. Disponible sur : < https://doi.org/10.18653/v1/N19-1423 > (consulté le 2 mars 2022)

Dixon R. M. W. *Basic Linguistic Theory Volume 1: Methodology*. [s.l.] : OUP Oxford, 2009. 398 p.ISBN : 978-0-19-157144-2.

Djemaa M., Candito M., Muller P., Vieu L. « Corpus annotation within the French FrameNet: a domain-by-domain methodology ». 2016. p. 8.

Dubois J., Dubois-Charlier F. « INFORMATIONS_LVF.pdf ». *LVF*. 1997. p. 7.

Dubois J., Dubois-Charlier F. « On the electronicl lexical resources of J. Dubois and F. Dubois-Charlier (Présentation du Dictionnaire Electronique des Mots (DEM) et de Locutions Verbales (LOCVERB) de Jean Dubois et Françoise Dubois-Charlier) [in French] ». Marseille, France : Association pour le Traitement Automatique des Langues (FondamenTAL), 2014. p. 69-73. Disponible sur : < https://aclanthology.org/W14-6401 > (consulté le 14 février 2022)

Dubois-Charlier. « Les verbes français (LVF) en format XML | RALI ». In : *RALI* [En ligne]. [s.l.] : [s.n.], [s.d.]. Disponible sur : < http://rali.iro.umontreal.ca/rali/fr/LVF-XML#consultation > (consulté le 9 mai 2022)

François T., Billami M. B., Gala N., Bernhard D. « Bleu, contusion, ecchymose : tri automatique de synonymes en fonction de leur difficulté de lecture et compréhension ». In : *JEP-TALN-RECITAL 2016* [En ligne]. Paris, France : [s.n.], 2016. p. 15-28. Disponible sur : < https://hal.archives-ouvertes.fr/hal-01346538 > (consulté le 2 mai 2022)

Francois T., Gala N., Watrin P., Fairon C. « FLELex: a graded lexical resource for French foreign learners ». 2014. p. 8.

Gala N., Bernhard D., Todirascu A., JAVOUREY-DREVET L., Bernhard D., Wilkens R., Meyer J.-P. « RECOMMANDATIONS pour des transformations de textes en français afin d'améliorer leur lisibilité et leur compréhension ». *ALECTOR* [En ligne]. novembre 2020. Disponible sur : < https://alectorsite.files.wordpress.com/2020/11/guidelines-linguistiques_alector_final.pdf > (consulté le 2 mai 2022)

Gala N., Javourey-Drevet L. « Mots "faciles" et mots "difficiles" dans ReSyf : un outil pour la didactique du lexique mobilisant polysémie, synonymie et complexité ». *Lidil Rev. Linguist. Didact. Lang.* [En ligne]. 3 novembre 2020. n°62,. Disponible sur : < https://doi.org/10.4000/lidil.8373 > (consulté le 2 mai 2022)

Gala N., Todirascu A., Francois T., Javourey-Drevet L. *Rapport final du projet ANR ALECTOR (Aide à la LECTure pour améliORer l'accès aux documents pour enfants dyslexiques)* [En ligne]. [s.l.] : Agence Nationale de la Recherche, 2021. Disponible sur : < https://hal.archives-ouvertes.fr/hal-03361468 > (consulté le 23 octobre 2022)

Glavaš G., Štajner S. « Simplifying Lexical Simplification: Do We Need Simplified Corpora? ». In : *Proc. 53rd Annu. Meet. Assoc. Comput. Linguist. 7th Int. Jt. Conf. Nat. Lang. Process. Vol. 2 Short Pap.* [En ligne]. *ACL-IJCNLP 2015*. Beijing, China : Association for

Computational Linguistics, 2015. p. 63-68. Disponible sur : < https://doi.org/10.3115/v1/P15-2011 > (consulté le 5 mai 2022)

Guillaume B., Fort K., Perrier G., Bédaride P. « Mapping the Lexique des Verbes du Français (Lexicon of French Verbs) to a NLP Lexicon using Examples ». *Eur. Lang. Resour. Assoc. ELRA*. mai 2014. p. 2806-2810.

Hershcovich D., Abend O., Rappoport A. « A Transition-Based Directed Acyclic Graph Parser for UCCA ». In : *Proc. 55th Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap.* [En ligne]. *ACL 2017*. Vancouver, Canada : Association for Computational Linguistics, 2017. p. 1127-1138. Disponible sur : < https://doi.org/10.18653/v1/P17-1104 > (consulté le 5 mai 2022)

Hershcovich D., Abend O., Rappoport A. « Multitask Parsing Across Semantic Representations ». In : *Proc. 56th Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap.* [En ligne]. *ACL 2018*. Melbourne, Australia : Association for Computational Linguistics, 2018. p. 373-385. Disponible sur : < https://doi.org/10.18653/v1/P18-1035 > (consulté le 16 septembre 2022)

Kingsbury P., Palmer M. « From TreeBank to PropBank ». *Proc. 3 rd Int. Conf. Lang. Resour. Eval. LREC-2002*. 2002. p. 5.

Klie J.-C., Bugert M., Boullosa B., De Castilho R. E., Gurevych I. « The INCEpTION Platform: Machine-Assisted and Knowledge-Oriented Interactive Annotation ». 2018. p. 5.

Kupść A., Haas P., Marin R., Balvet A. « Towards TreeLex++: Syntactico-Semantic Lexical Resource for French ». In : *Lang. Technol. Conf.* [En ligne]. Poznan, Poland : [s.n.], 2019. p. 6. Disponible sur : < https://hal.archives-ouvertes.fr/hal-02120183 > (consulté le 2 janvier 2022)

Le H., Vial L., Frej J., Segonne V., Coavoux M., Lecouteux B., Allauzen A., Crabbé B., Besacier L., Schwab D. « FlauBERT: Unsupervised Language Model Pre-training for French ». *ArXiv191205372 Cs* [En ligne]. 12 mars 2020. Disponible sur : < http://arxiv.org/abs/1912.05372 > (consulté le 4 mars 2022)

Lété B., Sprenger-Charolles L., Colé P. « MANULEX: A grade-level lexical database from French elementary school readers ». *Behav. Res. Methods Instrum. Comput.* [En ligne]. février 2004. Vol. 36, n°1, p. 156-166. Disponible sur : < https://doi.org/10.3758/BF03195560 >

Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L., Stoyanov V. « RoBERTa: A Robustly Optimized BERT Pretraining Approach ». *ArXiv190711692 Cs* [En ligne]. 26 juillet 2019. Disponible sur : < http://arxiv.org/abs/1907.11692 > (consulté le 23 mars 2022)

Loureiro D., Jorge A. « Language Modelling Makes Sense: Propagating Representations through WordNet for Full-Coverage Word Sense Disambiguation ». In : *Proc. 57th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. *ACL 2019*. Florence, Italy : Association for Computational Linguistics, 2019. p. 5682-5691. Disponible sur : < https://doi.org/10.18653/v1/P19-1569 > (consulté le 20 mars 2022)

Martin L., Muller B., Suárez P. J. O., Dupont Y., Romary L., De la Clergerie É. V., Seddah D., Sagot B. « CamemBERT: a Tasty French Language Model ». *Proc. 58th Annu. Meet.*

Assoc. Comput. Linguist. [En ligne]. 2020. p. 7203-7219. Disponible sur : < https://doi.org/10.18653/v1/2020.acl-main.645 >

McCarthy D., Navigli R. « The English lexical substitution task ». *Lang. Resour. Eval.* [En ligne]. juin 2009. Vol. 43, n°2, p. 139-159. Disponible sur : < https://doi.org/10.1007/s10579-009-9084-1 >

McCormick C. « BERT Word Embeddings Tutorial · Chris McCormick ». [s.l.] : [s.n.], [s.d.]. Disponible sur : < http://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/#introduction$ > (consulté le 9 mars 2022)

Mertens P. « Le dictionnaire de valence DICOVALENCE : manuel d'utilisation ». 25 juin 2010. Disponible sur : < https://d1wqtxts1xzle7.cloudfront.net/8217254/manuel_100625-with-cover-page-v2.pdf?Expires=1644839731&Signature=HB~VkPpWA8IsMOY5t2w0Be4rn1Pwn8Rjrispef Kyhve9YJkCLuT~cPjL8T06pxAFG3TRT9xUFSee6QeiXauFsakmi1hU2WgptgmiDVY3rnPr E855BIpN73jvtEb1z0XsGwNaHkTvSx31zbdfJHLmMjefrDNx0PQrp076EtVEQB6PBCwC2 7pT31Ozzhd3n-X4R5iMeCRgPi3GC9RUrWOXKYXid7pOTshRyYZiFcSumCqebnhtxXtutL1pDTByQPsIT kMVNLUUGdCpIsUivmlr9-l8i4-qSrYGd1Ia7gydPpwSWpF5IMXv3PnRi-gfibrloTLZ3QI6VOJ~k9Hv2UR9TA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA > (consulté le 14 février 2022)

Papineni K., Roukos S., Ward T., Zhu W.-J. « Bleu: a Method for Automatic Evaluation of Machine Translation ». In : *Proc. 40th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. *ACL 2002*. Philadelphia, Pennsylvania, USA : Association for Computational Linguistics, 2002. p. 311-318. Disponible sur : < https://doi.org/10.3115/1073083.1073135 > (consulté le 30 décembre 2021)

Prettenhofer P., Stein B. « Cross-Language Text Classification Using Structural Correspondence Learning ». In : *Proc. 48th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. *ACL 2010*. Uppsala, Sweden : Association for Computational Linguistics, 2010. p. 1118-1127. Disponible sur : < https://aclanthology.org/P10-1114 > (consulté le 5 mai 2022)

Qi P., Zhang Yuhao, Zhang Yuhui, Bolton J., Manning C. D. *Stanza: A Python Natural Language Processing Toolkit for Many Human Languages* [En ligne]. 23 avril 2020. Disponible sur : < https://doi.org/10.48550/arXiv.2003.07082 > (consulté le 14 août 2023)

Quiniou S., Daille B. « Towards a Diagnosis of Textual Difficulties for Children with Dyslexia ». In : *11th Int. Conf. Lang. Resour. Eval. LREC* [En ligne]. Miyazaki, Japan : [s.n.], 2018. Disponible sur : < https://hal.archives-ouvertes.fr/hal-01737726 > (consulté le 9 mai 2022)

Saggion H. « Automatic Text Simplification ». *Synth. Lect. Hum. Lang. Technol.* [En ligne]. 25 avril 2017. Vol. 10, n°1, p. 1-137. Disponible sur : < https://doi.org/10.2200/S00700ED1V01Y201602HLT032 >

Schnedecker C. *Nom propre et chaînes de référence* [En ligne]. [s.l.] : Librairie KLINCKSIECK, 1997. 231 p.(Recherches linguistiques). Disponible sur : < https://hal.archives-ouvertes.fr/hal-00808797 > (consulté le 9 mai 2022)

Shardlow M. « A Survey of Automated Text Simplification ». *Int. J. Adv. Comput. Sci. Appl.* [En ligne]. 2014. Vol. 4, n°1,. Disponible sur : < https://doi.org/10.14569/SpecialIssue.2014.040109 > (consulté le 18 février 2022)

Štajner S., Béchara H., Saggion H. « A Deeper Exploration of the Standard PB-SMT Approach to Text Simplification and its Evaluation ». In : *Proc. 53rd Annu. Meet. Assoc. Comput. Linguist. 7th Int. Jt. Conf. Nat. Lang. Process. Vol. 2 Short Pap.* [En ligne]. *ACL-IJCNLP 2015*. Beijing, China : Association for Computational Linguistics, 2015. p. 823-828. Disponible sur : < https://doi.org/10.3115/v1/P15-2135 > (consulté le 5 mai 2022)

Sulem E., Abend O., Rappoport A. « Semantic Structural Evaluation for Text Simplification ». In : *Proc. 2018 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Vol. 1 Long Pap.* [En ligne]. *NAACL-HLT 2018*. New Orleans, Louisiana : Association for Computational Linguistics, 2018a. p. 685-696. Disponible sur : < https://doi.org/10.18653/v1/N18-1063 > (consulté le 13 mars 2022)

Sulem E., Abend O., Rappoport A. « BLEU is Not Suitable for the Evaluation of Text Simplification ». *ArXiv181005995 Cs* [En ligne]. 14 octobre 2018b. Disponible sur : < http://arxiv.org/abs/1810.05995 > (consulté le 2 janvier 2022)

Sulem E., Abend O., Rappoport A. « Conceptual Annotations Preserve Structure Across Translations: A French-English Case Study ». In : *Proc. 1st Workshop Semant.-Driven Stat. Mach. Transl. S2MT 2015* [En ligne]. Beijing, China : Association for Computational Linguistics, 2015. p. 11-22. Disponible sur : < https://doi.org/10.18653/v1/W15-3502 > (consulté le 22 décembre 2021)

Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. « Attention is All you Need ». In : *Adv. Neural Inf. Process. Syst.* [En ligne]. [s.l.] : Curran Associates, Inc., 2017. Disponible sur : < https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html > (consulté le 7 mars 2022)

Wang A., Singh A., Michael J., Hill F., Levy O., Bowman S. R. « GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding ». *ArXiv180407461 Cs* [En ligne]. 22 février 2019. Disponible sur : < http://arxiv.org/abs/1804.07461 > (consulté le 20 mars 2022)

Wilkens R., Todirascu A. « Un corpus d'évaluation pour un système de simplification discursive (An Evaluation Corpus for Automatic Discourse Simplification) ». In : *Actes 6e Conférence Conjointe Journ. DÉtudes Sur Parole JEP 33e Édition Trait. Autom. Lang. Nat. TALN 27e Édition Rencontre Étud. Cherch. En Inform. Pour Trait. Autom. Lang. RÉCITAL 22e Édition Vol. 2 Trait. Autom. Lang. Nat.* [En ligne]. *JEP/TALN/RECITAL 2020*. Nancy, France : ATALA et AFCP, 2020. p. 361-369. Disponible sur : < https://aclanthology.org/2020.jeptalnrecital-taln.36 > (consulté le 4 mai 2022)

Xu W., Napoles C., Pavlick E., Chen Q., Callison-Burch C. « Optimizing Statistical Machine Translation for Text Simplification ». *Trans. Assoc. Comput. Linguist.* [En ligne]. 1 juillet 2016. Vol. 4, p. 401-415. Disponible sur : < https://doi.org/10.1162/tacl_a_00107 >

Zhou W., Ge T., Xu K., Wei F., Zhou M. « BERT-based Lexical Substitution ». In : *Proc. 57th Annu. Meet. Assoc. Comput. Linguist.* [En ligne]. *ACL 2019*. Florence, Italy :

Association for Computational Linguistics, 2019. p. 3368-3373. Disponible sur : < https://doi.org/10.18653/v1/P19-1328 > (consulté le 19 février 2022)