

Faculté	des Langues
	Université de Strasbourg

Master Technologies des langues

2021-2023

Annotation morphosyntaxique du corse

Connor MACLEAN

Mémoire de recherche

Sous la direction de :

Laurent Kevers, Stella Retali-Medori, Delphine Bernhard

Tables des matières

1. Définition et objectif du projet	1
2. Description de la langue corse	3
2.1 La situation de la langue corse	3
2.2 L'usage du corse aujourd'hui	4
2.3 Caractéristiques morphosyntaxiques du corse pour le TAL	5
2.4 Conclusion	7
3. État de l'art sur les outils en TAL	8
3.1 Modèles par apprentissage supervisé "classiques"	9
3.1.1 TreeTagger	9
3.1.2 Talismane	9
3.1.4 Stanford POS Tagger	10
3.2 Systèmes à base de réseaux de neurones	11
3.2.1 Stanza	11
3.2.2 spaCy	12
3.2.3 Flair	13
3.3 Avantages et inconvénients pour le corse	15
3.4 Licences de chaque outil	16
4. État de l'art sur l'étiquetage morphosyntaxique des langues peu dotées	18
4.1 Étiquetage morphosyntaxique des langues peu dotées, exemple de l'alsacien	18
4.2 Étiquetage morphosyntaxique des langues peu dotées, travail mené sur l'occitan, le picard et l'alsacien	19
4.3 Outiller plusieurs dialectes à partir d'un seul dialecte, le cas de l'occitan	21
4.4 TAL pour la langue corse	23
4.5 Applications possibles dans le cadre du projet de corse	25
5. Méthodologie	26
5.1 Prétraitement	26
5.2 Choix du corpus	27
5.3 Corpus annoté	27
5.4 Entraînement de modèle	29
5.5 Processus	29
6. Corpus	30
6.1 Canopé de Corse	30
6.3 Avantages d'un corpus nettoyé	31
6.5 Problèmes rencontrés pendant le traitement	32
6.6 Statistiques sur le corpus final	35
7. Tokénisation	37
7.1 Un bref aperçu du processus et des outils disponibles	37
7.3 Difficultés pour le corse	38
7.5 Notre tokeniseur	38
8. Création de word embeddings/plongements lexicaux	41
8.1 Introduction	41

8.2 Word2Vec	41
8.3 FastText / Floret	43
8.4 Comparaison de Word2Vec et FastText	46
9. Utilisation et entraînement de spaCy pour le corse	47
9.1 Processus	47
9.3 Utilisation des plongements lexicaux pour l'italien de spaCy	48
9.4 Entraînement du modèle corse - nos plongements FastText	50
9.5 Entraînement du modèle corse - plongements corses disponibles sur le site de FastText	51
9.6 Entraînement du modèle corse - nos plongements Floret	51
9.7 Utilisation des plongements français de spaCy	53
9.8 Discussion sur les résultats	54
10. Flair	56
10.1 Introduction	56
10.2 Plongements italiens Flair	58
10.3 Plongements multilingues Flair - version compacte	60
10.4 Plongements multilingues Flair	62
10.5 Plongements multilingues Flair, 100 itérations	64
10.6 Plongements multilingues Flair, version compacte, 100 itérations	66
10.7 Discussion sur les résultats	68
11. Comparaison de Flair et spaCy	69
11.1 Introduction	69
11.2 Facilité d'utilisation	69
11.3 Annotations de spaCy et Flair	70
11.4 Suite du projet	74
Références bibliographiques	75
1. Annexes	80
1.1 SpaCy	80
1.2 Utilisation des outils en TAL	86
1.2.1 TreeTagger	86
1.2.2 SpaCy	86
1.2.3 Talismane	88
1.2.4 Stanford POS Tagger	89
1.2.5 Stanza	90
1.2.6 Flair	91
1.3 Tokeniseur	92
1.4 Processus de conversion de PDF en TXT	93
1.5 Flair	98

1. Définition et objectif du projet

Ce projet a comme but de développer un outil d'annotation morphosyntaxique à partir d'un corpus en langue corse. Nous avons accès à un corpus d'entraînement de cent phrases annotées par Alice Millour (2022). Nous visons à faire prédire ces étiquettes de manière automatique afin d'élaborer ou de pouvoir élaborer un corpus annoté d'une plus grande taille.

Il s'agira de séparer le corpus en tokens et d'appliquer les étiquettes des UD (Universal Dependencies : <https://universaldependencies.org/u/pos/>)¹. Avant de procéder à la création du corpus, il faut en premier lieu répondre à de nombreuses questions concernant la création du corpus et son analyse. Le corse, une langue issue de la famille italo-romane, est une langue parlée en Corse et au nord de la Sardaigne. Au vu de sa situation géographique et de son appartenance à l'ensemble français, la langue française est désormais majoritairement parlée de l'île. Le corse compte aussi beaucoup de richesse et de variation selon la région dans laquelle il est parlé. Tout d'abord, il existe quatre voire cinq aires dialectales de la langue corse qui sont tout à fait compréhensibles par tout locuteur natif de la langue, mais cela présente un problème, surtout au niveau de l'orthographe –qui n'est en outre pas normalisée–, en ce qui concerne le corpus. Le corse fait partie de la catégorie de langues dites 'peu dotées' en ressources numériques et il ne compte pas une vaste quantité de publications, d'œuvres littéraires ou de sites-web. L'orthographe, qui vise à refléter la variation entre les aires dialectales, est différente et non standardisée, mais sa structure est semblable. Par exemple, le mot *fourmi*² dans la BDLC (*Banque de données Langue Corse*) (Kevers et al. 2019) se voit dans la plupart des cas en tant que *furmicula*, alors que le mot peut aussi s'écrire comme *fomicula* dans certains endroits. À l'aide de plusieurs ressources, notamment de la BDLC, il serait utile de choisir une aire à utiliser afin d'avoir des annotations cohérentes. Cependant, ce choix pourrait aussi avoir un impact sur les données disponibles à analyser car la taille du corpus serait sûrement réduite.

Le tout premier défi est de choisir quelles unités et quelles catégories grammaticales exactement il faut annoter. Le corpus comprendra-t-il des expressions idiomatiques ou sera-t-il un corpus purement académique ? Le corpus va être développé et évoluera avec le

¹ Universal Dependencies a créé un jeu d'étiquettes qui est parmi les plus utilisés dans l'étiquetage morphosyntaxique et compte de nombreuses ressources pour plusieurs langues. De leur site : "Universal Dependencies (UD) is a framework for consistent annotation of grammar (parts of speech, morphological features, and syntactic dependencies) across different human languages. UD is an open community effort with over 300 contributors producing nearly 200 treebanks in over 100 languages."

Notre traduction :

"Universal Dependencies (UD) est un *framework* pour l'annotation cohérente de la grammaire (parties du discours, caractéristiques morphologiques et dépendances syntaxiques) dans différentes langues humaines. UD est un effort communautaire avec plus de 300 contributeurs produisant près de 200 *treebanks* dans plus de 100 langues."

² https://bdlc.univ-corse.fr/bdlc/corse.php?page=lexiquefr&valeur_id=1633&idque=1633

projet pour ensuite créer un maximum de ressources pour la langue. Cette approche permettra de voir au fur et à mesure quels aspects sont les mieux traités.

Le défi de l'annotation morphosyntaxique de la langue sera d'attribuer à chaque forme un lemme ainsi que des informations grammaticales et flexionnelles. Pourtant, dans le cadre de ce mémoire, nous laissons de côté le problème de la lemmatisation pour nous limiter et nous concentrer sur l'attribution des catégories grammaticales. Cette approche, de se concentrer uniquement sur l'attribution des catégories grammaticales, est la seule qui est étudiée dans ce projet. Il existe déjà des outils d'annotation automatique avec des modèles créés pour l'italien, le français et l'espagnol qui pourraient être utiles pour bien définir les étiquettes et la syntaxe de la langue corse. Mais, vu que le corse n'a malheureusement pas de modèles déjà créés pour traiter des corpus, il va falloir développer une méthodologie et un plan de travail basés sur des projets déjà menés sur les langues romanes ainsi que sur des langues peu dotées en ressources.

Pour les étiquettes, le choix est pourtant plus simple. A notre connaissance, il existe le plus de supports pour les étiquettes des Universal Dependencies, qui seront donc favorisées dans le cadre de ce projet.

Dans le cadre de ce mémoire, nous avons regroupé plusieurs stratégies et la procédure de traitement a été définie et standardisée pour toutes nos approches. Nous avons commencé par choisir un corpus à convertir en plongements lexicaux. Ce corpus est le même pour tous les modèles utilisés. Ensuite, nous avons choisi un corpus d'entraînement et un corpus d'évaluation. Ces deux corpus sont disjoints, permettant une évaluation sur des données qui ne figurent pas dans le test. Finalement, nous avons évalué les modèles afin de voir quel modèle ou quels modèles sont les plus performants dans la tâche de l'annotation morphosyntaxique.

2. Description de la langue corse

Le corse est une langue faisant partie de la famille italo-romane, parlée sur l'île de Corse et dans le nord de la Sardaigne (Gallura). L'île se situe à 170 km au sud de Nice (France), 40 km de l'île d'Elbe et 70 km de la côte toscane. L'île de Corse est séparée en deux départements français dont la Haute-Corse et la Corse-du-Sud, qui sont au Nord et au Sud de l'île, respectivement.

2.1 La situation de la langue corse

Comme dans la majorité des régions de France, il s'avère que plusieurs langues se parlent sur l'île. La langue la plus parlée est le français, suivi de l'arabe maghrébin et ensuite le corse (Leclerc, s. d.). Il faut aussi préciser qu'il existe plusieurs langues et dialectes historiquement présents tels que le grec qui se parle à Cargèse et le bonifacien de Bonifacio.

Le corse est une langue qui trouve plusieurs similarités avec l'italien. Cela est le résultat de son évolution et du contact entre les langues. Notamment, le corse a beaucoup de points communs avec le dialecte toscan, qui est devenu l'italien "standard". La différenciation de l'italien, selon (langues-cultures-de-france, s. d.), est expliquée comme suit :

“Le corse a conquis son autonomie vis-à-vis de l'italien lorsque le français s'est imposé. Privé du continuum linguistique avec l'italien, il lui a fallu s'extraire de sa condition dialectale pour continuer à exister. Cela est d'autant plus vrai qu'avec les progrès de l'école publique, de l'intégration à la France, une France soucieuse d'éradiquer l'italien, la langue des élites, le français désormais, a eu ensuite tendance à s'installer dans les usages traditionnels du vernaculaire. Aussi peut-on affirmer sans craindre le paradoxe que la langue corse, d'une certaine façon, est la fille inattendue et illégitime de l'imposition du français dans l'île.”³

Cette analyse suppose que le corse, en raison de la distanciation à l'italien et de l'imposition du français a dû s'autonomiser pour continuer son développement et sa propagation. La langue aujourd'hui est malheureusement dans une phase de faible transmission familiale, ce qui la menace d'extinction à terme. L'un des buts de la création des ressources pour des langues peu dotées en ressource est de préserver et essayer de revitaliser une langue qui est en danger.

La langue aujourd'hui, selon l'Inchiesta sociolinguistica nant'à a lingua corsa, (2013) est considérée comme très importante pour l'identité et pour l'évolution de l'île. “86% disent que celui-ci est important dont surtout 48% pour qui il est très important. C'est chez les plus anciens que cet intérêt est le plus intense. D'ailleurs pour une écrasante majorité (90%) il faudrait à l'avenir en Corse parler le corse et le français.”

³ La Corse est passée sous domination française en 1769 suite à une deuxième bataille menée par les Français, après leur défaite contre Pasquale Paoli en 1768. (Leclerc, s. d.)

Aujourd'hui, la langue comprend énormément de variation dialectale, ce qui, à la fois, contribue à la richesse de la langue corse et complique le traitement de la langue à l'aide des outils en TAL. Mais, cette variation est relative et il sera tout à fait possible de la faire analyser par des outils de TAL.

La langue corse fait partie des langues considérées peu dotées en ressources. Cela veut dire que la langue ne possède pas une vaste collection de ressources numériques, comme c'est le cas pour l'anglais et le français par exemple. Les enjeux des langues peu dotées sont bien expliqués par Bernhard et al. (2018) :

“All languages with little resources have in common that their computerisation has a low financial profitability which does not compensate for considerable development costs. However, endowing these languages with electronic resources and tools is a major concern for their dissemination, protection and teaching (including for new speakers).”

En français (notre traduction) :

“Toutes les langues peu dotées en ressources ont en commun que leur numérisation a une faible rentabilité financière qui ne compense pas des coûts de développement considérables. Or, doter ces langues de ressources et d'outils électroniques est une préoccupation majeure pour leur diffusion, leur protection et leur enseignement (y compris pour les nouveaux locuteurs).”

2.2 L'usage du corse aujourd'hui

La situation de la langue corse, une langue parlée en France (Corse) et en Italie (Sardaigne) par rapport aux autres langues régionales de l'Italie est définie comme le suivant : “À ce jour, les autorités italiennes n'ont pas reconnu les langues régionales appartenant aux sous-groupes italo roman et gallo-roman, bien que le ladin et le frioulan soient reconnus. L'italien méridional et le sicilien, ainsi que le corse dans une moindre mesure, bénéficient néanmoins d'une position relativement stable en tant que langues vernaculaires de leurs communautés.” (*Atlas des langues en danger dans le monde - UNESCO Bibliothèque Numérique*, s. d.)

En France, la langue fait bien partie des langues dites régionales, qui sont définies sur le site du ministère de la culture comme suit : ⁴ “Les langues régionales se définissent, dans l'Hexagone, comme des langues parlées sur une partie du territoire national depuis plus longtemps que le français langue commune.”

Le nombre de locuteurs de la langue peut varier selon plusieurs sources dont (Leclerc, s. d.) où le nombre de locuteurs est autour de 150 000. Mais, l'article prend en compte que les

4

<https://www.culture.gouv.fr/Thematiques/Langue-francaise-et-langues-de-France/Nos-missions/Promouvoir-les-langues-de-France/Langues-regionales>

données ne sont pas toujours fiables et que le niveau de compréhension de la langue peut varier selon les locuteurs.

2.3 Caractéristiques morphosyntaxiques du corse pour le TAL

Le corse, issu de la famille italo-romane, est une langue qui trouve beaucoup de similitudes avec l'italien, et est donc, en théorie, au moins partiellement exploitable par des outils développés pour la langue italienne.

Par contre, il reste plusieurs formes qui peuvent apparaître selon le locuteur ou auteur d'un texte. Prenons l'exemple des articles décrits sur le site de la *Grammaire corse : accueil*, s. d. :

		Genre / Gèneru	
		masculin / maschile	féminin / feminile
nombre / nùmeru	singulier / in unu	lu, u, l'	la, a, l'
	pluriel / in parechji	li, i, l'	le, e, l'

Figure 1 - les articles, *Grammaire corse : accueil*, s. d.

Les formes sur cette figure sont des formes qui montrent des différences régionales. Par exemple, lu, la, li et le sont les formes normales du Cap Corse et de certaines propositions articulées.

Dans un tableau plus bas sur la page, on voit tous les articles accompagnés d'un nom et de leur traduction française. Ce qui est intéressant est que pour deux noms propres féminins, on peut voir les articles *A* et *La* utilisés dans le même contexte :


Corsu	Français	
U mare era calmu	La mer était calme	
L'omu andò inde Carlu	L'homme alla chez Charles	
Mi piace u vinu	J'aime le vin	
La Balanina	La Balanine (route)	
A Casinca	La Casinca	
A Porta	La Porta (village)	

Figure 2 - paires de phrases corses/français, *Grammaire corse : accueil*, s. d.

Donc, on peut supposer que les articles sont sémantiquement semblables et peuvent être employés de façon interchangeable.

Une autre difficulté dans le traitement du corse par un locuteur du français est la différence de genre entre les mots corses et français. Dalbera-Stefanaggi, (1978) donne quelques exemples de mots ayant un genre différent en corse:

- a zitella (féminin) : la fille
- u mare (masculin) : la mer
- a fica (féminin) : le figuier
- u ficu (masculin) : la figue

Dans le cadre du projet sur la langue corse, il existe aussi plusieurs variations qui pourraient être “neutralisées” par une approche qui choisit une forme canonique.

Par exemple :

mot français	mot corse	lemmes similaires	forme canonique
coccinelle	bella viola, bolelluccia, bulabulella, bulella, caccarinetta, caccavella, calacaloce, capitu mantellu, cardalina, catalina, catalinetta, ciattariola, cinciriola, ciriola, cuchjarella d'oru, fasgianella, figliola...	bella viola, bolelluccia, bulabulella, bulella, caccarinetta, caccavella, calacaloce, capitu mantellu, cardalina, catalina, catalinetta, ciattariola, cinciriola, ciriola, cuchjarella d'oru, fasgianella, figliola...	à voir

Tableau 1 - variation linguistique, créé à partir des données de la BDLC (Kevers et al. 2019)

Une dernière caractéristique du corse qui pose des questions à notre traitement est l’inclusion des apostrophes dans des mots corses. Il existe plusieurs cas où un mot contient une apostrophe et qu’il doit être séparé en deux mots distincts. Mais, par contre, il existe des mots contenant des apostrophes qui ne doivent pas être séparés. Quelques exemples de ce phénomène sont :

- nant’à = nant’+à
- liberalla = liberà+ella
- s'è = un seul token

Cette question est abordée en plus de détail dans les sections sur la création d’un tokeniseur.

2.4 Conclusion

Cette section montre la nécessité de développer des outils pour le traitement du corse ou d'utiliser des ressources existantes pour une autre langue. Des expérimentations et des tests ont été menés sur les outils développés pour le français et l'italien, notamment des bibliothèques TAL. Cette approche a été choisie afin de s'appuyer sur les ressources existantes pour les langues les plus proches. Des modèles italiens ont été utilisés pour la plupart des expérimentations, mais nous avons aussi vu des résultats intéressants avec un modèle français. Ces expériences sont abordées en plus de détail dans les sections qui suivent.

3. État de l’art sur les outils en TAL

Les *word embeddings* (également appelés plongements lexicaux) sont une technologie destinée à traiter le langage naturel et à améliorer la représentation des mots et des phrases. Les plongements lexicaux sont des représentations vectorielles qui peuvent avoir plusieurs formes différentes selon le système de création de ces vecteurs. Grâce à ces représentations, les algorithmes des outils existants en TAL peuvent traiter le langage humain à un niveau plus profond et plus précis. Ils sont devenus une partie essentielle dans le traitement du langage naturel et en particulier pour l’apprentissage profond.

Alors qu’il existe plusieurs outils nous permettant de réaliser l’annotation morphosyntaxique, ces outils sont plus ou moins adaptés à notre projet spécifique. Dans cette section nous allons voir la différence entre plusieurs outils en termes d’utilité dans notre travail, approche et utilisation. Les deux catégories principales explorées sont les modèles par apprentissage classique et les systèmes à base de réseaux de neurones. Les deux systèmes utilisent l’apprentissage supervisé.

Pour résumer, les modèles d’apprentissage supervisé “classiques” sont généralement basés sur des méthodes statistiques et nécessitent un ensemble d’entraînement annoté. Ils sont entraînés sur des données préalablement étiquetées pour prédire les annotations morphosyntaxiques sur de nouvelles données.

Les systèmes à base de réseaux de neurones utilisent des architectures de réseau de neurones profonds pour l’annotation morphosyntaxique. Ils sont généralement plus flexibles, mais ces systèmes seront entraînés sur les mêmes données que les modèles d’apprentissage supervisé “classiques”. Cela permet d’avoir des résultats montrant l’efficacité des deux approches sur la même tâche.

3.1 Modèles par apprentissage supervisé “classiques”

3.1.1 TreeTagger

TreeTagger⁵ (Schmid, 1994) est un outil d’annotation automatique développé à l’Université de Stuttgart. L’outil, créé en 1994, peut être utilisé et a été conçu avec plus de modèles afin d’analyser plusieurs langues. Des nouveaux modèles sont toujours en train d’être ajoutés à l’outil. Il a comme but de séparer des énoncés d’abord en tokens puis ensuite attribuer la catégorie grammaticale ainsi que le lemme. L’outil est très efficace et bien reconnu, malgré les erreurs parfois présentes en sortie de l’outil. Par contre, cet outil étiquette les catégories grammaticales mais n’effectue pas d’analyse au niveau de la syntaxe.

L’algorithme de TreeTagger est basé sur des modèles statistiques qui sont préalablement entraînés sur de grands corpus annotés. Ces modèles apprennent à associer des mots, lemmes, étiquettes grammaticales à partir des contextes dans lesquels ils apparaissent. Lors de l’annotation d’un nouveau texte, TreeTagger utilise ces modèles statistiques pour attribuer des étiquettes morphosyntaxiques aux mots du texte, en se basant sur leur contexte.

Cet outil, qui existe depuis presque trente ans, est aussi très fiable. Dans l’analyse d’un petit corpus en français de 500 mots, j’ai pu constater une exactitude de plus de 95 %. Pourtant, l’exactitude serait différente selon la taille et la complexité du corpus analysé, mais globalement les résultats sont plus que satisfaisants.

Parmi les avantages de TreeTagger il y a la quantité de recherche déjà faite à l’aide de TreeTagger et les ressources développées qu’il possède ainsi que sa polyvalence et la possibilité de l’intégrer dans plusieurs logiciels. Par exemple, TreeTagger est utilisable dans de recherches TXM, un outil d’analyse de corpus textométrique (Heiden et al., 2010). TreeTagger est aussi exécutable à l’aide de scripts Python et une librairie Python, `treetaggerwrapper` (Pointal, 2019) a été également développé pour afficher la sortie des analyses TreeTagger. L’utilisation de TreeTagger est décrite dans cette annexe : [1.2.1 TreeTagger](#).

3.1.2 Talismane

⁵ <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/> Le jeu d’étiquettes utilisé pour le traitement du français se trouve ici : <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/french-tagset.html>

Celui de l’italien se trouve ici : <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/italian-tagset.txt>

Note : Il existe deux jeux d’étiquettes différents pour l’italien ce qui montre qu’il n’y a pas qu’une façon standard d’analyser une langue.

Talismane est un analyseur syntaxique développé par Urieli (2013) dans le cadre de sa thèse. Pour l'instant, l'outil n'est disponible que pour l'analyse de trois langues : le français, l'anglais et l'occitan. L'utilisation de Talismane en tant qu'analyseur syntaxique pour l'occitan est discuté plus en détail dans (Vergez-Couret & Urieli, 2014).

L'outil analyse un texte en le passant par quatre étapes :

1. le découpage en phrases
2. la segmentation en mots
3. l'étiquetage (attribution d'une catégorie morphosyntaxique)
4. le parsing (repérage et étiquetage des dépendances syntaxiques entre les mots)

Cet outil utilise une approche statistique et traite séparément chacune de ses quatre étapes, décrite comme une analyse en cascade. L'outil base ses calculs sur plusieurs critères dont les étiquettes présentes dans un lexique et les catégories des mots qui entourent un mot. Cette utilisation de collocations permet de faire la différence de deux mots qui ont la même forme mais qui ont deux sens et/ou deux catégories différentes. L'exemple trouvé sur le site de Talismane ⁶ montre exactement ce phénomène où il fait la différence entre un verbe et un nom qui ont la même forme :

1	Les	le	DET	det	p	2	det	—	—
2	poules	poule	NC	nc	fp	5	subj	—	—
3	du	de	P+D	P+D	ms	2	dep	—	—
4	couvent	couvent	NC	nc	ms	3	obj	—	—
5	couvent	couver	V	v	PS3p	0	root	—	—
6	.	.	PONCT	PONCT	null	5	punct	—	—

Figure 3 - analyse par Talismane, (Urieli, 2013)

Ce qui fait de Talismane un outil intéressant pour l'analyse en TAL est sa façon d'appliquer ses règles d'étiquetage et ses possibilités de modification des règles. Selon (Urieli, 2013), il y a des règles définissables qui sont appliquées au moment de l'analyse. Quelques applications des règles pourraient être pour des spécificités des corpus à analyser, comme la définition de certains mots qui ne seraient pas normalement reconnus.

L'utilisation de Talismane est décrite dans cette annexe: [1.2.3 Talismane](#).

3.1.4 Stanford POS Tagger

Stanford POS Tagger est un outil d'étiquetage automatique développé en 2000. Créé par Kristina Toutanova à l'origine, plusieurs personnes ont contribué au projet et ont amélioré la performance et ajouté plus de langues (Toutanova et al., 2003). Pour l'instant, il existe des

⁶ <http://redac.univ-tlse2.fr/applications/talismane.html>

modèles pour l'anglais, le chinois, l'arabe, le français, l'allemand et l'espagnol. Par contre, l'étiqueteur est modifiable et il est possible de l'entraîner pour une autre langue si on dispose d'un corpus d'entraînement.

Le fonctionnement du Stanford POS Tagger est basé sur un algorithme de classification supervisée. Il est entraîné à partir de grands corpus de textes annotés manuellement, où chaque mot est associé à une étiquette de partie du discours.

Le modèle utilisé, selon (Toutanova et al., 2003) et appliqué aux données du Penn Treebank WSJ (*Wall Street Journal*, un journal américain), a atteint une exactitude de 97,24%. Ils expliquent ensuite le fonctionnement de plusieurs modèles statistiques qui sont actuellement utilisés dans l'étiquetage morphosyntaxique. Par exemple, ils trouvent que les modèles actuels (en 2003) sont surtout des modèles unidirectionnels, c'est-à-dire qu'ils recherchent le contexte du mot en se basant sur l'étiquette précédente. Il existe donc trois types de modèles : le modèle droite à gauche, le modèle gauche à droite et le modèle bidirectionnel, détaillé dans le graphique de (Toutanova et al., 2003) ci-dessous :

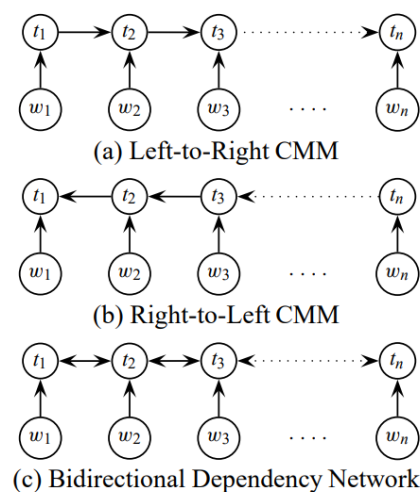


Figure 4 - architecture des modèles Stanford POS Tagger, (Toutanova et al., 2003)

L'utilisation de Stanford POS Tagger est décrite dans cette annexe : [1.2.4 Stanford POS Tagger](#).

3.2 Systèmes à base de réseaux de neurones

3.2.1 Stanza

Stanza <https://stanfordnlp.github.io/stanza/> est une librairie développée par le groupe Stanford NLP, qui est le groupe qui a également créé le Stanford POS Tagger. Stanza comprend plusieurs fonctionnalités qui sont, selon (Qi et al., 2020) :

- Un minimum d'efforts pour la configuration ;

- Une chaîne de traitement complète à base de réseaux neuronaux pour une analyse de texte robuste, y compris la tokenisation, l'expansion de token multi-mots, la lemmatisation, l'étiquetage des caractéristiques morphologiques et de la partie du discours (POS), l'analyse syntaxique des dépendances et la reconnaissance des entités nommées
- Des modèles neuronaux pré-entraînés supportant 66 langues
 - Note : Il existe quatre modèles entraînés pour le français et cinq pour l'italien
- Une interface Python stable et officiellement maintenue pour CoreNLP.

L'approche de Stanza repose sur l'utilisation de réseaux de neurones, pour effectuer l'analyse morphosyntaxique des textes. Les modèles de Stanza sont entraînés sur de grands corpus de textes annotés manuellement.

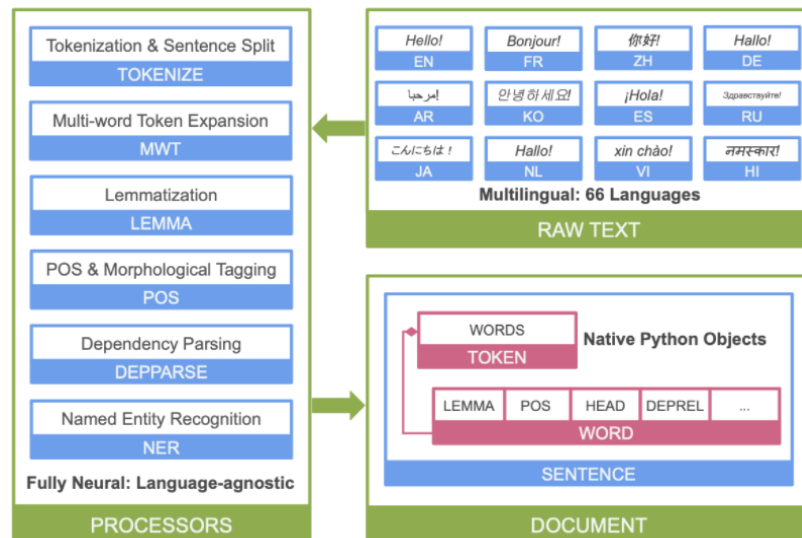


Figure 5 - processus de conversion de Stanza, de <https://stanfordnlp.github.io/stanza/>

Dans la figure 5, nous voyons qu'un texte est traité par plusieurs processus y compris la tokenisation, l'annotation morphosyntaxique et la reconnaissance d'entités nommées avant d'être converti en objet Python.

L'utilisation de Stanza est décrite dans cette annexe: [1.2.5 Stanza](#).

3.2.2 spaCy

La librairie spaCy, <https://spacy.io/>, un projet open source créé par Honnibal & Montani, (2017), est extrêmement adaptable et un projet récent qui est toujours mis à jour par ses développeurs. Cet outil est très bien documenté, utilisé par des chercheurs et contient une vaste quantité d'options pour pouvoir adapter l'outil à une nouvelle langue ou dialecte. Il y a donc une communauté derrière, ce qui le rend à la fois plus facile l'utilisation et rend plus pertinent et moderne l'outil. À ce stade, il y a du support pour plus de 64 langues, dont des

pipelines créés pour 19 d'entre elles. Notamment, il existe plusieurs ressources spaCy pour le traitement de l'italien et du français.

SpaCy utilise principalement des réseaux de neurones pour effectuer ses analyses. SpaCy utilise des techniques d'apprentissage supervisé, où les modèles sont alimentés avec des exemples annotés manuellement.

La liste complète des fonctionnalités (adapté de l'anglais) comme présentée sur leur site est :

- Apprentissage multi-tâches avec des *transformers* prétraités comme BERT
- Vecteurs de mots prétraités
- Tokenisation à motivation linguistique
- Composants pour la reconnaissance des entités nommées, l'étiquetage morphosyntaxique, l'analyse syntaxique des dépendances, la segmentation des phrases, la classification des textes, la lemmatisation, l'analyse morphologique, la liaison des entités, etc.
- Facilement extensible avec des composants et des attributs personnalisés.
- Prise en charge de modèles personnalisés dans PyTorch, TensorFlow, et d'autres plateformes.
- Visualisateurs intégrés pour la syntaxe et le *NER* (*Named Entity Recognition*, Reconnaissance des entités nommées)
- Gestion aisée du packaging, du déploiement et du flux de travail des modèles
- Précision robuste, rigoureusement évaluée

Pour l'instant, ce qui serait le plus important serait la tokenisation, la lemmatisation et les analyses syntaxique et morphologique. De façon générale, l'étiquetage se fait à l'aide des étiquettes des Universal Dependencies (de Marneffe et al., 2021).

Cet outil ne dispose pas de modèle pour le corse, mais peut être entraîné à partir de corpus annotés.

L'utilisation de spaCy est décrite dans cette annexe: [1.2.2 SpaCy](#).

3.2.3 Flair

Flair⁷ (Akbik et al., 2018) est une librairie Python open source construite en PyTorch par l'Université de Humboldt à Berlin. Chez Flair (Akbik et al., 2019), on trouve plusieurs similitudes avec spaCy. Par exemple, comme spaCy, Flair est un étiqueteur syntaxique et possède un modèle de reconnaissance d'entités nommées. Par contre, Flair n'a pas autant de modèles que spaCy, qui a seulement 21 modèles disponibles contrairement aux 64 chez spaCy. Mais, cela ne pose pas problème pour ce projet car il existe des modèles créés pour le français et l'italien.

⁷ <https://github.com/flairNLP/flair>

Flair utilise des réseaux de neurones récurrents et des modèles de langage contextualisés, tels que les *Transformers*, pour capturer les relations entre les mots et leur contexte dans un texte.

Pour l'étiquetage morphosyntaxique, Flair se sert des corpus des Universal Dependencies pour 16 langues des 21, dont l'italien. Pour les 5 autres langues, dont l'anglais et le français, l'entraînement se fait à l'aide des corpus Flair (Akbik et al., 2018). Le modèle d'entraînement utilisé pour l'étiquetage morphosyntaxique est montré et expliqué par Schweter (2020) :

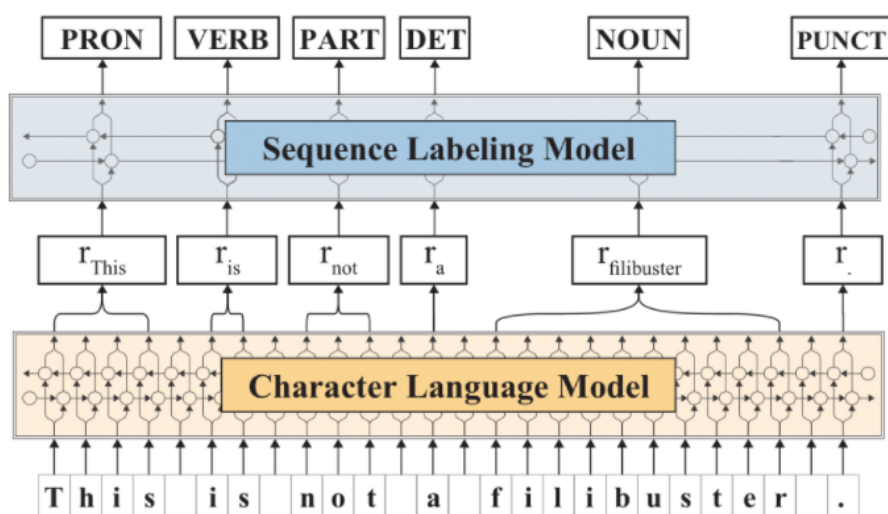


Figure 6 - architecture du modèle Flair; Schweter (2020)

“For training our PoS tagging models we use a BiLSTM with a hidden size of 256, a mini batch size of 8 and an initial learning rate of 0.1. We reduce the learning rate by a factor of 0.5 with a patience of 3. This factor determines the number of epochs with no improvement after which learning rate will be reduced. We train for a maximum of 500 epochs and use stochastic gradient descent as optimizer. For each language we train PoS tagging models and evaluate 3 runs and report an averaged accuracy score.” (Schweter, 2020)

Notre traduction :

"Pour l'entraînement de nos modèles d'annotation morphosyntaxique, nous utilisons une BiLSTM avec une taille cachée de 256, une taille de mini lot de 8 et un taux d'apprentissage initial de 0,1. Nous réduisons le taux d'apprentissage par un facteur de 0,5 avec une patience de 3. Ce facteur détermine le nombre d'époques sans amélioration après lequel le taux d'apprentissage sera réduit. Nous entraînons pour un maximum de 500 époques et utilisons la descente de gradient stochastique comme optimiseur. Pour chaque langue, nous formons des modèles d'étiquetage de PoS et évaluons 3 cycles d'apprentissage et rapportons un score d'exactitude moyen."

L'utilisation de Flair est décrite dans cette annexe: [1.2.6 Flair](#).

3.3 Avantages et inconvénients pour le corse

Ce tableau récapitulatif sert à montrer les avantages et les inconvénients de chaque outil dans le contexte de notre projet.

Outil	Avantages	Inconvénients
Talismane	<ul style="list-style-type: none"> • Support pour le français et pour une langue peu dotée en ressources (l'occitan) 	<ul style="list-style-type: none"> • Pas de support pour l'italien, il serait très chronophage et pas trop utile d'entraîner un nouveau modèle sachant qu'il en existe dans d'autres outils
TreeTagger / TreeTaggerWrapper	<ul style="list-style-type: none"> • Utilisable en Python • Support pour l'italien et le français • Importante quantité de ressources et de documentation 	<ul style="list-style-type: none"> • Il existe des outils plus modernes qui sont peut-être mieux adaptés à la tâche
Stanford POS Tagger	<ul style="list-style-type: none"> • Utilisable en Python à l'aide de la librairie NLTK https://www.nltk.org/ 	<ul style="list-style-type: none"> • L'équipe a développé Stanza, qui est plus intuitif et qui a le plus de support de l'équipe
Stanza	<ul style="list-style-type: none"> • Utilisable en Python • Facile à utiliser • Support pour l'italien et le français • Mise à jour régulièrement 	<ul style="list-style-type: none"> • Documentation moins complètes que d'autres librairies
SpaCy	<ul style="list-style-type: none"> • Utilisable en Python • Facile à utiliser • Bien documentée • Support pour l'italien et le français • Mis à jour régulièrement 	<ul style="list-style-type: none"> • L'exécution est lente, cette librairie est peut-être mieux pour le traitement de textes courts
Flair	<ul style="list-style-type: none"> • Utilisable en Python • Permet de traiter des textes multilingues 	<ul style="list-style-type: none"> • Plus compliquée à utiliser et moins ressourcee que spaCy

Tableau 2 - comparaison des outils

Alors que tous ces outils sont utiles et peuvent servir dans la réalisation du projet, certains sont mieux adaptés pour le traitement du corse. Par exemple, TreeTagger et par extension TreeTaggerWrapper peuvent être utilisés pour leur rapidité d'exécution et leur flexibilité d'entraînement de modèle. Aussi, SpaCy et Stanza sont très bien documentés et très utilisés et nous avons donc mené des expérimentations pour voir lequel est le mieux adapté pour le traitement du corse. Ces expériences se font sur le même jeu de données afin de comparer l'exactitude obtenue par les modèles différents. Nous avons aussi comparé les performances obtenues sur la tâche d'apprentissage d'un nouveau modèle pour choisir la meilleure librairie pour ce projet.

Pour Talismane, l'entraînement de modèle est bien documenté, mais il n'est pas aussi simple à faire qu'avec d'autres librairies.

3.4 Licences de chaque outil

Alors qu'il existe plusieurs outils intéressants, il faut retenir que certains outils ne sont pas libres de droits et ne peuvent pas toujours être utilisés sans payer. Ci-dessous, ce tableau vise à donner des informations sur la licence de chacun des outils discutés. Dans notre cas, tous les outils peuvent être utilisés, libres de droits, dans la recherche.

Outil	Licence	Lien
Talismane	Affero GPL v3	Lien (en anglais) : https://www.gnu.org/licenses/agpl-3.0.html
TreeTagger / TreeTaggerWrapper	MIT	https://stringfixer.com/fr/MIT_license
Stanford POS Tagger	Licence publique générale GNU, version 2	Lien (en anglais) : https://www.gnu.org/licenses/old-licenses/gpl-2.0.html
Stanza	APACHE 2.0	Lien (en anglais) : https://www.apache.org/licenses/LICENSE-2.0
SpaCy	MIT	https://stringfixer.com/fr/MIT_license
Flair	MIT	https://stringfixer.com/fr/MIT_license

Tableau 3 - licences des outils

4. État de l’art sur l’étiquetage morphosyntaxique des langues peu dotées

Avant de procéder à la création d’un outil nous permettant d’annoter le corse, il est nécessaire d’analyser les approches précédentes. Cela permet de donner des idées pour notre cas et de voir s’il n’existe pas d’autres ressources que nous pouvons utiliser par la suite.

4.1 Étiquetage morphosyntaxique des langues peu dotées, exemple de l’alsacien

En tant que premier exemple de l’outillage d’une langue peu dotée en ressources, nous avons le travail mené par Bernhard et Ligozat (2013) sur le traitement automatique de l’alsacien.

Dans l’article, il est noté que le traitement automatique de l’alsacien présente plusieurs difficultés. Tout d’abord, il y avait le problème de la non standardisation des graphies des dialectes alsaciens. Pour un traitement optimal et bien réussi, il faut que les outils soient capables de gérer la variation.

Ensuite, la langue ayant plusieurs dialectes et formes différents des mots présents, l’article discute d’une approche très intéressante pour traiter les mots dits “hors norme”. La solution la plus fréquente consiste à remplacer les mots hors norme par leur forme canonique dans une des deux langues, soit l’allemand ou l’alsacien (Mosquera et al., 2012). C’est-à-dire, les chercheurs vont choisir une forme ‘normale’ à utiliser pour remplacer des variations d’un mot. Cette stratégie vise à normaliser les corpus et de rendre plus efficace et précis le traitement.

Lorsqu’il n’existe pas forcément des outils pour le traitement d’une langue peu dotée, l’utilisation des corpus parallèles afin de permettre d’aligner les traductions et de trouver leurs équivalents. Cette approche permet de pourvoir projeter des annotations dans la plupart des articles qui utilisent les corpus parallèles.

Plusieurs autres méthodes pour le traitement des textes “hors norme” sont proposées. Une méthode, utilisée pour le traitement des textes en dialecte labourdien par Hulden et al. (2011), est d’utiliser des corpus parallèles et des méthodes d’apprentissage afin d’apprendre des transformations.

Les deux méthodes proposées sont :

1. “La première extrait des règles phonologiques de remplacement en repérant les sous-chaînes différentes dans les mots alignés d’un corpus parallèle.”
2. “La seconde est similaire à la première mais utilise également les contre-exemples pour chaque règle afin de déterminer les contextes les plus appropriés pour appliquer une règle de transformation.”

(Bernhard & Ligozat, 2013)

La stratégie finalement employée par Bernhard et Ligozat (2013) était de choisir cinq documents variés, y compris des phrases tirées d'un dictionnaire multilingue Français - Allemand - Alsacien - Anglais. Le but était de créer des corpus parallèles. La raison derrière le choix du dictionnaire était la présence de diverses variantes graphiques, ce qui est utile dans le traitement d'une langue qui a une variation graphique importante comme l'alsacien ou le corse.

Le corpus a été ensuite traité par des étiqueteurs allemands afin d'évaluer leur performance.

Les résultats montrent que le traitement en transposant des textes dans une langue similaire est très intéressant. "La méthode consiste à transposer les mots outils des textes alsaciens vers leurs équivalents en allemand standard. Cette transposition nécessite pour seule ressource un lexique bilingue des mots outil." (Bernhard & Ligozat, 2013). Les résultats obtenus montrent aussi qu'il est possible de faire annoter des textes sans transposition. Dans cet exemple de l'alsacien, le traitement de textes Wikipédia en langue originale a donné une précision de 71% (TreeTagger) et 53% (Stanford POS). Par contre, l'exactitude après la transposition des textes a augmenté jusqu'à 75% (TreeTagger) et jusqu'à 85% (Stanford POS).

Cette analyse montre que cette approche, après la tokénisation, pourrait être utile dans le cadre du projet d'étiquetage morphosyntaxique du corse. Il serait utile d'effectuer des expérimentations afin de voir son utilité sur le traitement des corpus italiens - français - corses transposés.

Par contre, il reste toujours possible d'entraîner un modèle TreeTagger pour le traitement d'une langue qui n'est pas dotée de beaucoup de ressources.

4.2 Étiquetage morphosyntaxique des langues peu dotées, travail mené sur l'occitan, le picard et l'alsacien

Un projet particulièrement intéressant pour l'analyse de la langue corse est le projet RESTAURE, un projet mené par Bernhard et al. (2018). L'objectif du projet est défini comme suit⁸ :

En français : "L'objectif global du projet RESTAURE est de fournir des ressources informatiques et des outils de traitement pour trois langues régionales de France : l'alsacien, l'occitan et le picard." (Notre traduction)

Donc, dans le cadre du projet actuel, il sera utile d'analyser et d'utiliser certaines des méthodes présentées dans le projet RESTAURE.

⁸ "The overall objective of the RESTAURE project is to provide computational resources and processing tools for three regional languages of France: Alsatian, Occitan and Picard." (Bernhard et al., 2018).

En faisant des recherches sur les méthodes les plus répandues sur l'analyse des langues peu dotées, on constate qu'il existe plusieurs méthodologies similaires. Une idée très intéressante qui est présente dans plusieurs articles est l'idée d'utiliser soit un jeu d'étiquettes développé pour une langue similaire, soit un corpus parallèle. Pour les étiquettes, il est courant de commencer par une analyse d'un jeu d'étiquettes existant et de le modifier pour mieux convenir aux besoins de la langue à analyser, mais il y a une autre stratégie qui est aussi utile. La deuxième stratégie consiste à utiliser les étiquettes des Universal Dependencies et aussi de les adapter aux besoins du projet. Ce choix fait partie d'une méthodologie plus large qui englobe également le choix d'annoter des corpus ou pas, le choix de faire du transfert depuis une autre langue, et même le choix des outils d'apprentissage.

Dans le cadre du projet sur les langues peu dotées en ressources en France mené par Bernhard et al. (2018), l'équipe a décidé de configurer un jeu d'étiquettes simple et uniforme afin de pouvoir s'en servir dans l'analyse des trois langues. Le jeu d'étiquettes uni était ensuite légèrement modifié pour traiter chacune des trois langues.

Le guide d'annotation était pourtant différent pour chaque langue analysée. Il dépend des particularités morphosyntaxiques de chaque langue et nécessite donc une mise à jour du jeu d'étiquettes pour mieux correspondre à la langue.

Pour choisir et constituer les corpus utilisés dans le projet, il fallait d'abord trouver des textes libres de droit, sinon il ne serait pas possible de rendre disponible l'intégralité des corpus. Ceci représente un défi dans l'analyse des trois langues étudiées. Regardant dans les méthodes utilisées par (Bernhard et al., 2018), on trouve qu'il y a plusieurs possibilités et endroits où trouver des textes pour un corpus, notamment :

- Wikipédia, s'il existe une section créée pour la langue
- Des publications locales/en ligne où la langue se parle
- Des bibliothèques virtuelles
- Des ressources et bases de données développées pour des projets précédents

La disponibilité de ses ressources va bien sûr dépendre de leurs licences d'exploitation.

Dans (Bernhard et al., 2018), en commençant par l'alsacien, six annotateurs ont été choisis qui attribuaient des étiquettes de chaque token mais aussi une traduction en français. Le travail a été réparti entre les annotateurs de la façon suivante :

- A1 a annoté tous les 21 documents
- A2 et A3 ont annoté 6 et 5 des 21 documents
- A4 a pris des décisions et a corrigé tous les 21 documents
- A5 et A6 ont surtout proposé des corrections pour les documents annotés

Ensuite, l'accord inter-annotateur a été calculé et des améliorations ont été proposées, comme l'automatisation de quelques tâches de correction.

En occitan, un corpus en dialecte languedocien a été créé puis corrigé par quatre annotateurs. Ensuite, Talismane, qui a été entraîné sur le corpus languedocien, (Vergez-Couret & Urieli, 2014) a été utilisé pour analyser un plus grand corpus contenant les six dialectes de l'occitan. Ces annotations ont été corrigées par trois annotateurs à l'aide de l'outil AnaLog (Lay & Pincemin, 2010) pour en créer le corpus final. Finalement, l'accord inter-annotateur a été calculé. L'accord a été calculé au début et au milieu de l'annotation manuelle, et l'accord s'améliore au milieu de l'étude.

Le troisième exemple, l'annotation du picard, a été géré différemment encore. Le corpus, en format csv, a été tokenisé manuellement. Trois locuteurs de picard ont travaillé sur le corpus pendant une période de 11 mois. Le premier annotateur a annoté 20 textes, et ensuite il a discuté avec le deuxième annotateur pour trouver des corrections éventuelles. Le deuxième annotateur a ensuite annoté le reste des textes. Enfin, toutes les annotations ont été révisées par les annotateurs deux et trois. Le troisième annotateur n'effectue que des révisions. Le problème posé par l'analyse du picard est que les dictionnaires ne sont pas complets et le défi de trouver une bonne traduction est toujours présent.

Selon l'article, il est prévu d'utiliser les corpus annotés pour développer des étiqueteurs morphosyntaxiques qui peuvent prendre en compte la variation spatiale des trois langues.

4.3 Outiller plusieurs dialectes à partir d'un seul dialecte, le cas de l'occitan

Selon le travail mené par (Vergez-Couret & Urieli, 2014), il est possible d'annoter plusieurs dialectes en utilisant un corpus disponible qu'en un seul dialecte. L'étude, qui se base sur une approche axée sur le *machine learning*/apprentissage supervisé, analyse des corpus de deux dialectes différents de l'occitan, le languedocien et le gascon. Pour le corpus d'entraînement, il n'y a qu'un seul dialecte qui est analysé, les résultats ayant toutefois atteint une précision de 89% (Vergez-Couret & Urieli, 2014).

Le défi présenté par cet article est celui de la création de corpus et de lexiques qui nécessite beaucoup d'effort, particulièrement pour des langues peu dotées ou pour lesquelles il faut créer des corpus à partir d'une quantité limitée de ressources. Un autre défi qui revient encore et encore dans toutes les recherches menées sur des langues peu dotées est le fait que ces langues n'ont souvent pas des graphies standardisées et qu'elles ont souvent des variations dialectales.

4.3.1 Talismane

La méthodologie présentée par (Vergez-Couret & Urieli, 2014) concerne l'utilisation de Talismane, un outil d'étiquetage automatique qui fonctionne déjà dans le traitement de l'anglais et du français.

En ce qui concerne le paramétrage de Talismane dans l'analyse de l'occitan, les mêmes traits utilisés dans l'analyse de l'anglais et du français ont été utilisés.

4.3.2 Création de lexique et de corpus d'entraînement

Selon (Vergez-Couret & Urieli, 2014), la création d'un lexique pour la langue occitane, et plus précisément pour le dialecte languedocien a été réalisée à l'aide d'un dictionnaire de la langue ainsi que le lexique Apertium (Martinez, s. d.). Pour les formes fléchies, un script a été créé pour les générer automatiquement pour chaque adjectif, nom et participe passé verbal.

Le corpus d'entraînement, par contre, a été extrait à partir d'un seul roman.

Ce choix a été fait afin d'entraîner un modèle sur un dialecte avant de l'utiliser dans le traitement des deux dialectes.

Le corpus, en total, consiste de 2 500 tokens, les 1 000 premiers ayant été analysés par trois annotateurs différents et les 1 500 derniers ayant été analysés par un seul annotateur.

4.3.3 Analyse des corpus

Les trois corpus à analyser par Talismane, à l'aide du corpus d'entraînement "mono dialectal" ont été choisis afin de représenter deux dialectes de l'occitan ainsi que deux variétés d'un dialecte. La question principale dans leur analyse était "Est-il préférable d'annoter un plus grand corpus d'entraînement, ou de compiler un plus grand lexique ?" (notre traduction)⁹ (Vergez-Couret & Urieli, 2014)

Afin d'y répondre, l'équipe a décidé, lors de l'analyse, de diviser le corpus d'entraînement en deux et de créer plusieurs sous-lexiques. En total, il y avait :

- trois options de corpus d'entraînement
- cinq options de lexique
- deux options de règles à appliquer
- et donc trente évaluations par corpus d'évaluation

4.3.4 Analyse des résultats

L'exactitude varie selon le corpus et les règles appliquées, variant entre 66,17% et 94,15%. La variante du corpus utilisé pour l'entraînement présente la plus grande exactitude, cependant, les résultats observés lors de l'analyse des autres variantes restent intéressants.

L'équipe a toujours des idées pour améliorer l'analyse des autres dialectes, en raison d'une exactitude relativement basse atteinte pour un dialecte en particulier, le gascon. La raison du manque de résultats de qualité pour le dialecte gascon est tout simplement la quantité limitée

⁹ "is it better to annotate a larger training corpus, or compile a larger lexicon?"

de ressources car il s'agit d'un dialecte très peu doté de ressources. Après l'ajout d'une seule règle ciblant un phénomène spécifique qui n'est pas présent dans les autres dialectes, l'exactitude a augmenté de 3%. Mais, afin d'augmenter encore l'exactitude, il faudrait malheureusement plus de ressources pour la langue, y compris un lexique spécifique au dialecte.

4.3.5 Conclusions

L'équipe a montré qu'il est possible d'avoir de bons résultats sans énormément de ressources, mais que cela nécessite un lexique plutôt complet. Ils ont trouvé qu'il est plus utile de favoriser un plus grand lexique qu'un plus grand corpus d'entraînement pour le développement de ressources dans une analyse supervisée. Cela est intéressant dans le cadre de ce projet car on dispose d'un très petit corpus d'entraînement et d'évaluation, mais d'un corpus nettoyé assez complet pour la création des plongements lexicaux. Cette étude montre qu'il est possible d'avoir des résultats intéressants avec seulement une relativement petite quantité de données.

4.4 TAL pour la langue corse

La langue corse n'est, actuellement, pas très outillée. Il existe pour le moment quelques ressources numériques, et quelques bases de données développées pour la langue. Le but de Kevers & Retali-Medori (2020) est donc d'améliorer la disponibilité de ressources pour la langue et d'augmenter la quantité de ressources pour la langue. Ils ont comme projet de se servir de la BDLC (*Banque de Données Langue Corse*) afin de développer plus d'outils, et, à long terme de créer un *Basic Language Resource Kit* (BLARK) (Krauwert, 2003).

4.4.1 La BDLC

Ce projet, qui date de 1986 est l'évolution et expansion du projet NALC (*Nouvel Atlas Linguistique et ethnographique de la Corse*) qui a été créé en 1975. Les données linguistiques trouvées dans la BDLC sont issues d'entretiens avec des locuteurs natifs en corse. Selon Kevers et Retali-Medori, (2020), pendant les entretiens semi-dirigés, les questions alternent entre des questions précises telles que "Comment dit-on telle chose ?" et des questions plutôt ouvertes comme "Comment fait-on telle chose ?" (par exemple "comment fait-on le pain ?") afin d'enregistrer le mot corse qui correspond au français et de recueillir des témoignages (ethnotextes) qui ont la forme de textes libres. Un exemple d'une question de (Kevers & Retali-Medori, 2020) : "Cum si dici "tailler la vigne" in corsu?". Suivant les entretiens, les données sont traitées, analysées et mises sur le site de la BDLC.

Par contre, dans l'utilisation de la ressource, il existe un défi déjà discuté. Le défi qui revient encore et encore est celui de la variation de la langue corse. Sur le site de la BDLC, il existe énormément de données et beaucoup de travail a été effectué pour enregistrer et montrer la variation de la langue, ce qui est son objet, mais par contre la variation commence à poser

problème dans l'analyse de la langue à l'aide des outils TAL. Par exemple, le mot “coccinelle” donne 84 correspondances, dont 21 lemmes. Ces correspondances sont soit des différences phoniques, soit des différences graphiques. Cette question est abordée en plus de détail dans le travail de Kevers et al. (2019).

4.4.2 Développement de ressources et d'outils pour le corse

Selon (Kevers & Retali-Medori, 2020), il existe en 2020 seulement 93 ressources (Leixa et al., 2014) pour la langue corse, un tiers venant directement du projet BDLC. La plupart des autres ressources listées sont des journaux, des blogs, des articles scientifiques, etc. Il existe aussi *Infcor* et le *Wiktionary* corse. La majorité des ressources disponible pour la langue ne sont malheureusement pas directement applicables aux études TAL de la langue corse, mis à part le corpus W2C¹⁰, le réseau sémantique BabelNet¹¹ et les ressources créées dans le cadre du projet Crubadan.

4.4.3 Dictionnaire

Malgré la variation dialectale, il reste possible d'exploiter la BDLC pour traiter les données.

Un premier dictionnaire LADL (format des entrées : forme, lemme.codes_grammaticaux_sémantiques:code_flexionnels/commentaire) a été créé par l'équipe, qui contient “20.875 formes, dont 17.860 formes simples (se rapportant à 10.224 lemmes) et 3.015 formes composées (se rapportant à 2.244 lemmes)” (Kevers et al., 2019). Le dictionnaire initial a été appliqué à un corpus d'ethnotextes et environ 49% des occurrences ont été reconnues. Le dictionnaire, quand il sera plus développé, deviendra une ressource très pertinente au projet de l'étiquetage morphosyntaxique de la langue corse.

4.4.4 Lemmatisation

Le travail fait sur la lemmatisation de la langue corse a plusieurs buts, y compris la réalisation d'un étiqueteur morphosyntaxique.

La lemmatisation de la langue est aussi concernée par la variation et le travail et le développement est toujours en cours. Il est dit que, “Cette procédure est en cours de précision sur différents points importants liés aux variations dialectales et à la non normalisation : choix du lemme entre les différentes versions attestées selon les régions, gestion des variations dues à l'utilisation variable des accents.” (Kevers et al., 2019)

Le projet de lemmatiseur s'est réorienté et concentré sur l'annotation morphosyntaxique, laissant les questions relatives aux lemmes pour des développements ultérieurs.

¹⁰ <https://ufal.mff.cuni.cz/w2c>

¹¹ <https://babelnet.org/>

Illustré dans la figure 7 ci-dessous, la procédure envisagée pour annoter de manière semi-automatique les ethnotextes de la BDLC est la suivante :

- Un texte du dictionnaire bilingue (CO-FR) DELAF ou un ethnotexte est ouvert ainsi qu'un dictionnaire dérivé de cette même base de données.
- Le texte est tokénisé et le dictionnaire est appliqué.
- Une liste de formes non reconnues est obtenue en sortie.
- Une étude lexicologique est menée sur les formes non reconnues, ce qui permet d'enrichir le dictionnaire initial. Les formes erronées, si elles existent, sont ensuite corrigées.
- Les textes corrigés sont ouverts par le lemmatiseur, ils sont tokenisés encore et le dictionnaire est appliqué. Il ne reste plus aucun mot non reconnu par le dictionnaire à ce stade.
- La dernière étape est la désambiguïsation, les analyses manquantes sont ajoutées et les analyses excédentaires sont éliminées.

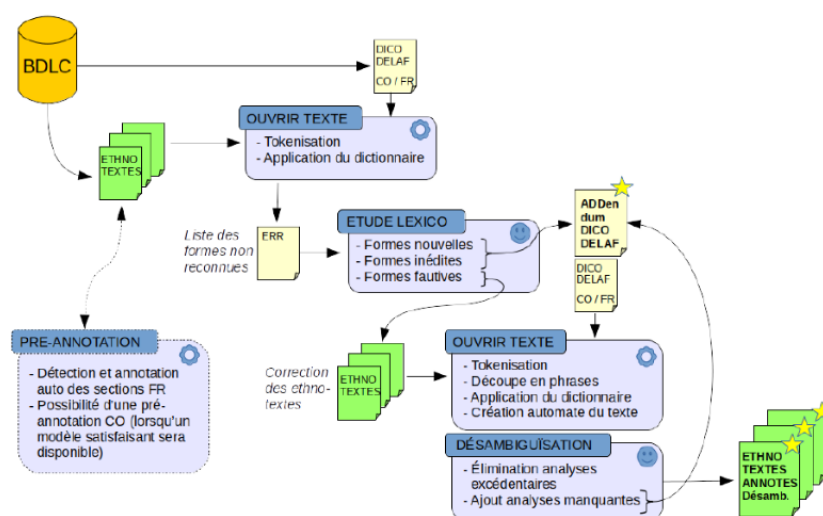


Figure 7 - processus d'annotation des ethnotextes, (Kevers et al., 2019)

Le travail actuel ne concerne pas encore l'entraînement d'un modèle tel que TreeTagger (Schmid, 1994), Kevers et al., (2019) préfèrent développer leurs textes lemmatisés avant de procéder à une analyse complète. Mais, en même temps, ils explorent l'idée d'utiliser un étiqueteur déjà entraîné pour une autre langue, l'italien. Cette idée vient du travail mené par (Bernhard & Ligozat, 2013; Vergez-Couret, 2013; Bernhard et al., 2018) sur l'occitan et l'alsacien.

4.5 Applications possibles dans le cadre du projet de corse

Cette section sert à concrétiser l'idée d'utiliser des ressources développées pour des langues proches sur le corse. Il a été donc envisageable de me servir des outils déjà développés pour le français et/ou l'italien.

Dans le cas de l'alsacien, le travail consistait à utiliser des outils existants pour le traitement de l'allemand et de transposer des mots alsaciens à la place. Ce travail nécessitait comme ressource un corpus bilingue allemand/alsacien afin d'assurer une bonne analyse par la machine. Dans le cadre de ce projet, il n'était pas nécessaire de constituer des corpus bilingues corse/italien, mais ces approches montrent l'idée d'utiliser des ressources développées pour une langue sur le traitement d'un autre.

5. Méthodologie

Le choix d'une méthodologie a été très important car il détermine la direction et l'organisation des expériences de création d'un étiqueteur morphosyntaxique corse. Il était nécessaire d'organiser les idées, évaluer les approches et ensuite adapter notre approche.

Les étapes principales de la méthodologie se résument en quatre parties :

1. Le prétraitement, y compris la tokenisation
2. Le choix du corpus pour la création de plongements lexicaux
3. Le choix de corpus pour l'entraînement et l'évaluation des modèles
4. Le choix de modèle à entraîner

5.1 Prétraitement

La première problématique posée est celle de la tokenisation. Alors que normalement on peut supposer que la tokenisation peut se faire facilement à l'aide d'un tokeniseur existant pour une langue similaire, cela n'est pas le cas pour le corse. Son orthographe pose problème, notamment en raison des apostrophes présentes qui peuvent affecter le découpage des mots. Par exemple :

- `nant'à` = `nant'+à`
- `liberalla` = `liberà+ella`
- `s'è` = un seul token

Il était donc primordial d'analyser les tokeniseurs existants déjà dans les librairies spaCy et Flair. Après il a fallu voir s'il existe un modèle qui serait capable de couper en tokens un corpus corse. Cela n'est actuellement pas une réalité et il a donc fallu que nous créions un tokeniseur. Le tokeniseur, en raison du niveau d'analyse sur les mots et non sur les sous-mots ou les lemmes, a pu être relativement simple.

J'ai créé un tokeniseur qui prend en entrée un fichier .txt et qui sort un fichier .csv avec un mot affiché par ligne. Ce tokeniseur ne découpe les mots qu'à la frontière du mot. Le code a

été créé en février 2022, puis a ensuite été adapté en février 2023 avant de commencer nos expériences. Une discussion sur le code utilisé dans nos expériences se trouve ici : [7.5 Notre tokeniseur](#) et le code se trouve dans l'annexe [1.3 Tokeniseur](#).

5.2 Choix du corpus

Il existe déjà des corpus en format XML¹² de la bible en langue corse, un corpus Wikipédia et un corpus en format XML du site <https://www.apiazetta.com/>.

Il existe également le corpus du Canopé de Corse¹³ (Kevers et MacLean, 2023), (Kevers, 2022) qui a été initié lors d'un stage à Corte en été 2022. L'élaboration du corpus a continué jusqu'à fin 2022 et il a été publié début 2023. Le corpus utilisé dans le cadre de ce mémoire consiste en une version partielle du corpus car le corpus n'était pas terminé au moment des expériences. Ce corpus contient des œuvres littéraires, de la littérature jeunesse et pour enfants ainsi que des manuels scolaires et des documents historiques. Ce corpus, d'une taille de plus de 260 000 mots organisés et structurés en paragraphes, sera une ressource très utile dans le cadre de ce projet. Si on ajoute aussi les textes dont le traitement n'a pas abouti à un document très propre, la taille de ce corpus dépasse les 400 000 mots.

Pour mener ce projet à bien, le corpus du Canopé de Corse a été utilisé. Ce corpus est bien mis en forme, vérifié et il est un corpus de bonne qualité nous permettant de mener des expériences sans se soucier de la qualité de nos données.

5.3 Corpus annoté

À l'Université de Corse, un corpus annoté de cent phrases a été créé par Alice Millour (2022) et il a été utilisé dans le cadre du projet.

Ces cent phrases ont été annotées uniquement par elle. Le format du corpus est d'un document contenant un token par ligne, séparé par une tabulation, et ensuite la catégorie grammaticale.

Tuttu	PRON
què	PRON
cù	ADP
e	DET
territurali	NOUN
di	ADP
u	DET
2015	NUM
chianu	PRON

¹² <https://bdlc.univ-corse.fr/tal/index.php?page=res>

¹³ <https://www.ortolang.fr/market/corpora/corpus-canope-de-corse>

da PART
vene VERB
subbituADV
dopu ADV
. PUNCT

D'après un retour de Stella Retali-Medori, j'ai appris qu'il est possible que le corpus d'entraînement et le corpus d'évaluation contiennent des erreurs parce qu'en réalité il faut séparer *chì* de *anu* qui est le verbe avoir (*avè*) à la Pe6 au présent de l'indicatif.

Les catégories grammaticales qui sont utilisées dans l'étiquetage sont les suivantes :

- ADJ : adjectif
- ADP : adposition
- ADV : adverbe
- AUX : auxiliaire
- CCONJ : conjonction de coordination
- DET : déterminant
- INTJ : interjection
- NOUN : nom
- NUM : numéral
- PART : particle
- PRON : pronom
- PROPN : nom propre
- PUNCT : ponctuation
- SCONJ : conjonction subordonnée
- SYM : symbole
- VERB : verbe
- X : autre

Ces catégories sont des catégories Universal Dependencies (de Marneffe et al., 2021), ces catégories sont très utilisées et sont exploitées également par les librairies spaCy et Flair. L'avantage est que nous utilisons un seul jeu d'étiquettes pour tous les modèles que nous analysons) ¹⁴

Ce corpus, dans l'intérêt d'entraîner nos modèles sur un corpus annoté, a été séparé en un corpus d'entraînement et un corpus de validation. Le corpus d'entraînement contient 80 % des phrases (soit 80 phrases) alors que le corpus de validation contient 20 % des phrases.

¹⁴ La liste originale avec des définitions plus précises se trouve ici (en anglais) : <https://universaldependencies.org/u/pos/>.

Cette répartition permet d'assurer que les données soient différentes à chaque étape, ce qui permet une juste analyse et un entraînement abouti.

5.4 Entraînement de modèle

Un choix important est le choix d'entraîner un modèle. Nous avons exploré plusieurs possibilités, abordées dans les sections suivantes, pour entraîner des modèles. Nos approches, dans un premier temps, explorent les modèles existants pour l'italien et pour le français. Notre seconde approche explore la création de plongements lexicaux corses créés à partir de nos corpus Canopé. Cette approche permet d'avoir un modèle corse, mais nous allons en créer plusieurs afin d'examiner la performance de plusieurs outils pour notre cas spécifique.

Nous avons donc utilisé le même corpus d'entraînement et de validation pour chaque expérience, mais avec des modèles et des bibliothèques différents. Les deux bibliothèques utilisées sont spaCy et Flair. Cela nous permet d'avoir une vision globale sur les possibilités d'entraînement et de trouver l'outil, le modèle et l'approche les plus performants.

5.5 Processus

Pour résumer, il nous fallait plusieurs choses pour assurer le bon déroulement des expériences. Il était nécessaire de définir le corpus d'entraînement et le corpus de validation. Ensuite, il était nécessaire de définir le corpus qu'il fallait utiliser dans la génération de nos divers plongements lexicaux. Ce corpus a dû être prétraité pour que les mots soient bien pris en compte un par un et que les apostrophes à l'intérieur des mots ne disparaissent pas. Finalement, nous avons décidé d'entraîner des modèles à l'aide des bibliothèques spaCy et Flair.

6. Corpus

Pendant l'été 2022, j'ai effectué un stage au sein de l'Université de Corse dans le but de créer un corpus corse d'une bonne qualité à partir des documents du Canopé de Corse¹⁵. Le stage a été financé par le projet DiViTal¹⁶ qui vise à :

“accroître la vitalité et la visibilité de plusieurs langues de France : l'alsacien, le corse, l'occitan et le poitevin-saintongeais. Il se positionne à la croisée de la linguistique descriptive et de la linguistique de corpus. Son but principal est la constitution de ressources, en particulier de corpus bruts et annotés...”

6.1 Canopé de Corse

Alors qu'il existe plusieurs ressources dont un blog, des traductions littéraires et des livres¹⁷, nous n'avons pas eu un corpus de grande taille propre avec des niveaux de langue variés. Les ressources existantes avant ce stage sont discutées par Kevers et al., (2021) La création d'un tel corpus a donc été objet d'un stage de deux mois pendant l'été 2022.

Nous avons eu accès à plus de cinquante œuvres en corse, grâce à une convention passée entre l'Université de Corse et Canopé de Corse. Plusieurs contiennent également une traduction française. Ce chiffre ne cesse d'augmenter et plusieurs nouvelles œuvres y ont été rajoutées depuis que nous avons commencé le travail de conversion de ces œuvres en corpus. Ces œuvres, du site datant de juin 2022 ont donc été la cible de la création de notre corpus.

¹⁵ <http://www.educorsica.fr/ouvrages>

¹⁶ <https://divital.gitpages.huma-num.fr/fr/>

¹⁷ <https://bdlc.univ-corse.fr/tal/index.php?page=res>

6.2 Corpus existants

Corpus	Nombre de mots (approximatif)	Commentaires
Wikipédia	919 000	Le corpus a été écrit par différentes personnes et il n'y a pas de contrôle éditorial.
A Piazzetta	504 000	Le corpus, vu qu'il a été extrait d'un blog, est susceptible de contenir des fautes, notamment dans les commentaires des internautes. A noter aussi, les commentaires contiennent un mélange de corse et de français.
La Bible	771 000	Le niveau de langue est assez soutenu. Le corpus n'est constitué que d'un seul document, il n'y aura donc pas énormément de variation dans le contenu.

Tableau 4 - comparaison des corpus existants

6.3 Avantages d'un corpus nettoyé

Pour une langue peu dotée de ressources, il est important de pouvoir maximiser l'utilité du peu de ressources auxquelles nous avons accès. Certes, il est intéressant d'utiliser un corpus tel que Wikipédia car il a une taille importante et une quantité de mots non négligeable, mais un corpus de cette taille est difficile à analyser pour se rendre compte des biais. Ce phénomène est exploré en détail par Millour & Fort (2020). Plus important encore, même après un prétraitement, une vérification manuelle n'est pas tout à fait possible. Mais, on peut faire un parcours de surface du corpus pour se rendre compte de sa qualité. Quand il s'agit d'une langue peu dotée de ressources, il est important de pouvoir explorer le corpus et s'assurer de la qualité du contenu. Avec un corpus comme celui de Wikipédia, il est donc difficile de vérifier la qualité et l'uniformité des données.

Un corpus nettoyé comprend plusieurs avantages. Le premier concerne la qualité des données. Avec un corpus nettoyé, nous identifions quelles données sont importantes et utiles

pour le traitement. Les incohérences ou les passages mal transcrits peuvent donc être éliminés. Les erreurs peuvent aussi être repérées pendant le nettoyage et ensuite supprimées. Le nettoyage, un long processus, est nécessaire pour une qualité élevée de nos données et permet d'avoir un corpus de base qui est fiable et qui pourra être augmenté par la suite.

Un deuxième avantage, qui sera abordé ultérieurement de façon plus détaillée, concerne la performance des modèles. En effet, un modèle entraîné sur des données qui sont fiables, et qui ne comprennent pas une quantité significative d'erreurs, donnera de meilleurs résultats qu'un modèle entraîné sur des données de basse qualité. Le modèle serait alors moins bon sur des données de moins bonne qualité. Cette hypothèse est à tester, mais nous n'allons pas travailler sur des corpus qui sont plus « bruités » dans le cadre de ce mémoire.

En ce qui concerne l'analyse des données, qu'elle soit manuelle ou automatique, un corpus nettoyé a ses avantages. Il donnera plus facilement des résultats fiables concernant des statistiques importantes telles que le nombre de mots du corpus et le nombre de mots uniques. Ces statistiques peuvent être affectées par un corpus non nettoyé qui comprend des erreurs d'orthographe ou des problèmes de transcription ou même des variantes. Un corpus contient certains mots qui sont dans une langue différente à la place. Ce dernier problème est important dans le corpus d'A Piazzetta (Kervers et al., 2021) car le corpus contient aussi des commentaires des internautes qui ont tendance à mélanger le français et le corse. Globalement, pour un corpus, plus le corpus est propre, plus il peut être valorisé pour en tirer des enseignements d'un point de vue linguistique.

Le dernier avantage est la fiabilité en elle-même. Les données fiables et d'une quantité connue permettent de savoir ce qui se trouve dans le corpus. Pour un corpus qui est représentatif de la langue, il faudrait mettre en place un processus de sélection de documents. Notre corpus permettra d'effectuer nos premières analyses des outils d'annotation morphosyntaxique et nous permettra d'évaluer les modèles actuellement disponibles et de prendre conscience de ce qu'il faudra améliorer pour avoir des résultats de plus en plus précis.

6.5 Problèmes rencontrés pendant le traitement

Le traitement des documents divers qui constituent le corpus final n'était pas toujours simple et nécessitait une approche adaptée à chaque type de document.

6.5.1 Littérature pour enfants

Pour cette catégorie dans laquelle nous trouvons le plus d'œuvres, il y avait plusieurs choses qu'il fallait prendre en compte afin de bien traiter ces documents.

La première chose qu'il fallait faire était de définir ce qui constitue un paragraphe. Pour une catégorie telle que la littérature pour adultes, les paragraphes sont déjà très bien définis. Par

contre, dans la littérature pour enfants, les styles d'écriture et de mise en page sont très différents d'un auteur à un autre. Un paragraphe pouvait être composé de plusieurs phrases ou même un seul mot. Les lignes pouvaient avoir plusieurs mots ou être découpées de plusieurs façons différentes. Mais, le point commun entre les livres de cette catégorie était la longueur. Les pages ne contenaient pas beaucoup de texte. Donc, pour simplifier la tâche, un paragraphe dans cette catégorie, pour la plupart des livres, était une page complète. Cette décision a également supprimé la plupart des problèmes liés à la mise en page et la complexité des lignes individuelles.

La deuxième chose qu'il fallait gérer était la reconnaissance de caractères. Certains livres, qui étaient scannés avant de devenir des PDFs traitables, avaient des caractères qui n'étaient pas reconnus. Ce problème s'appliquait aussi à des portions de texte de taille et/ou de couleur différente. Pour certains documents, il était possible de corriger manuellement les fautes présentes. Mais pour d'autres documents, il était très difficile de récupérer le texte sans tout transcrire à la main. Le pour et le contre ont dû être pesés pour chacun de ces cas difficiles. Nous avons pris en considération notamment la taille du texte, si un texte comportait de graves problèmes et qu'il n'était pas très long, il n'a pas été traité. Mais, finalement, la plupart des textes ont pu être récupérés.

Après, considérant la démarche simplifiée du traitement, nous avons pu récupérer facilement les traductions françaises de la fin d'une bonne partie des livres. Cette version pourrait à terme constituer un corpus parallèle qui serait intéressant pour des expériences de traduction automatique du corse, mais cela ne fait pas partie des tâches à réaliser pendant le stage.

6.5.2 Littérature de jeunesse

Cette catégorie était définitivement la plus facile à traiter. Les problèmes courants sont proches de ceux rencontrés dans la littérature pour enfants. Ces problèmes étaient les limites de paragraphes parfois difficilement repérées et l'addition des mots de tailles de police et de couleurs différentes.

L'erreur la plus souvent corrigée était le titre des chapitres. Dans la plupart de ces livres, le format du titre du chapitre était différent du reste du texte, ce qui est normal. Par contre, notre script n'arrivait pas à bien séparer ou parfois même inclure les titres dans nos paragraphes. Donc, pour une partie significative de livres de cette catégorie, il était nécessaire d'insérer manuellement le titre du chapitre dans le corpus.

L'autre chose à laquelle il fallait faire attention était les limites de paragraphes. Comme chaque livre avait une mise en page différente, il fallait adapter le script pour pouvoir prendre en compte les marqueurs de limite de paragraphes particuliers dans chaque livre.

Globalement, le traitement de cette section s'est passé sans trop de difficultés mais il fallait adapter légèrement nos méthodes et nos scripts pour bien traiter chacun des livres.

6.5.3 Littérature pour adultes

La littérature pour adultes n'était pas la catégorie la plus difficile, mais la catégorie qui nécessitait le plus de temps de traitement.

Les limites de paragraphes n'étaient pas toujours respectées par notre script, ce qui obligeait une vérification manuelle du découpage en paragraphes. Ce découpage prenait assez de temps, et aussi avec la fatigue, il y a forcément des erreurs qui n'ont pas été retrouvées.

Un autre problème récurrent était celui des notes de bas de page, qui étaient parfois reconnues comme faisant partie des paragraphes et donc prises en compte dans le découpage des paragraphes. Certains livres avaient des notes de bas de page d'une certaine taille et donc étaient faciles à repérer et à enlever. D'autres notes, par contre, étaient d'une taille variée et entraînent donc plus de modifications manuelles.

La vérification a été faite au niveau des paragraphes, et pour économiser du temps consistait à regarder les premières et dernières lignes des paragraphes pour être sûr de la délimitation.

L'autre difficulté était le traitement des versions françaises, quand il y en avait. Ces versions avaient souvent la même mise en forme, mais parfois des phrases, notamment des dialogues se trouvaient dans des paragraphes différents. Alors qu'en corse, le découpage était différent.

L'alignement des versions françaises et corses était la tâche la plus longue à faire, mais après plein de vérifications, nous avons réussi à traiter et créer quelques petits corpus alignés. Le travail global n'est pas fini, mais il s'agit d'une première étape.

6.5.4 Livres scolaires sur la Corse

Les livres scolaires sur la Corse sont des livres contenant des informations sur chaque région de l'île. Ces livres mettent en valeur l'histoire et la géographie des régions et sont donc une source assez importante d'informations sur l'île. Par contre, ces livres contiennent aussi beaucoup de graphiques, de notes de bas de page, d'images et autres qui empêchent un traitement simple.

Pour bien traiter ces documents il fallait effectuer plus de prétraitements que dans les catégories précédentes. Nous avons enlevé des images, des légendes et des phrases décrivant les images et légendes associées. Ce prétraitement a été réalisé directement sur le fichier PDF afin d'en générer une version « nettoyée » des éléments de mise en page qui n'étaient pas pertinents et qui constituaient des obstacles pour la conversion en texte. Nous avons discuté plusieurs fois de ce qu'il était important à conserver dans ces textes et nous avons pris des décisions nous permettant d'extraire le plus de textes possible.

Pour ce prétraitement, dans un premier temps nous avons enlevé les images à l'aide du logiciel Draw de la suite LibreOffice. Ensuite, nous avons parcouru les textes à la recherche

des éléments servant à décrire les images enlevées car ces informations n'étaient plus utiles. Finalement, nous avons traité les documents comme les précédents, mais en faisant plus attention aux limites des paragraphes. Les limites de paragraphes étaient assez faciles à repérer par notre script, ce qui ne correspondait pas à mes attentes. A l'issue du stage, ces documents étaient partiellement traités et il faut un peu plus de temps pour que les documents soient prêts à diffuser sous forme de corpus.

6.5.5 Poésie

Parmi les documents que nous avons à disposition pour transformer en corpus de textes, il y avait de la poésie. Cette catégorie était difficile à transformer car nos outils étaient plus adaptés au repérage des paragraphes. La poésie, avec une mise en forme différente pour chaque poème, posait donc problème.

Nous avons pris la décision dans le cas d'un poème difficile à traiter, de ne pas le traiter. La raison principale derrière cette décision est que le traitement d'un poème prend plus de temps, et n'ajoute que très peu de mots au corpus final.

Le corpus, (Kevers et MacLean, 2023), (Kevers, 2022) a été finalisé et publié le 17 janvier 2023. Il est disponible au téléchargement sur le site d'Ortolang.

6.6 Statistiques sur le corpus final

Le corpus a été divisé en quatre groupes selon le genre du texte comme détaillé dans le tableau suivant.

Catégorie	Documents PDF total
Littérature enfantine	17
Littérature jeunesse	5
Littérature (pour adultes)	8
Livres scolaires sur la Corse	10
Total	40

Tableau 5 - catégories des documents du corpus

En regardant le tableau, il faut également prendre en compte le fait que le nombre d'œuvres ne correspond pas forcément à la longueur du texte. Par exemple, la section littérature pour enfants compte les textes les moins longs car il s'agit de très courtes histoires.

Ces statistiques viennent de la version du corpus à ma disposition depuis la fin du stage en

juillet 2022. Les statistiques sont différentes pour le corpus officiel de décembre 2022 (Kevers, 2022).

Taille du corpus corse : **270 845 mots**

Mots uniques du corpus : **41 540 mots**

Taille du corpus français : **18 693 mots qui correspondent à 11 documents**

La taille du corpus a été trouvée à l'aide de l'outil WC intégré dans Linux.

Pour les mots uniques du corpus, un autre traitement était nécessaire. Les mots du corpus ont été écrits ligne par ligne dans un fichier texte par un script Python que j'ai fait et ensuite la commande *uniq* a été utilisée pour donner le nombre de mots uniques du corpus. Une approche plus simplifiée aurait été de mettre les mots dans une structure de type *set* de Python. Des expressions régulières ont aussi été utilisées pour enlever la ponctuation qui se trouvaient à l'extérieur des mots et assurer un compte fiable.

7. Tokénisation

- Avant de commencer les divers entraînements des modèles dans le but de développer un étiqueteur morphosyntaxique, les données textuelles doivent être prétraitées. La première étape est la tokénisation des mots dans un corpus

Ces étapes ont été effectuées sur notre corpus dans la réalisation de ce projet, et elles sont abordées en plus de détail ici.

7.1 Un bref aperçu du processus et des outils disponibles

La tokenisation est le processus de décomposition des phrases en unités plus petites et plus faciles à gérer. Ces unités correspondent à des unités lexicales et peuvent varier des signes de ponctuation aux mots complexes. Sans ce processus, le modèle à entraîner est incapable de reconnaître la structure des phrases et ne peut faire aucune prédiction basée sur l'ordre des mots.

Ensuite, pendant l'étape de vectorisation, les séquences d'entrée sont représentées sous forme de vecteurs afin de rendre l'entrée utilisable par le modèle à entraîner. Un avantage notable de cette représentation est la capacité du modèle à "apprendre" les relations entre les mots d'une phrase. Cela n'est pas le cas de toutes les représentations vectorielles, mais sera le cas des outils de vectorisation utilisés au sein de ce projet. Certains modèles sont également capables d'identifier correctement les mots mal saisis ou mal orthographiés, car ils prennent en compte le contexte de chaque mot pendant la phase d'apprentissage.

7.1.1 Types de tokenisation

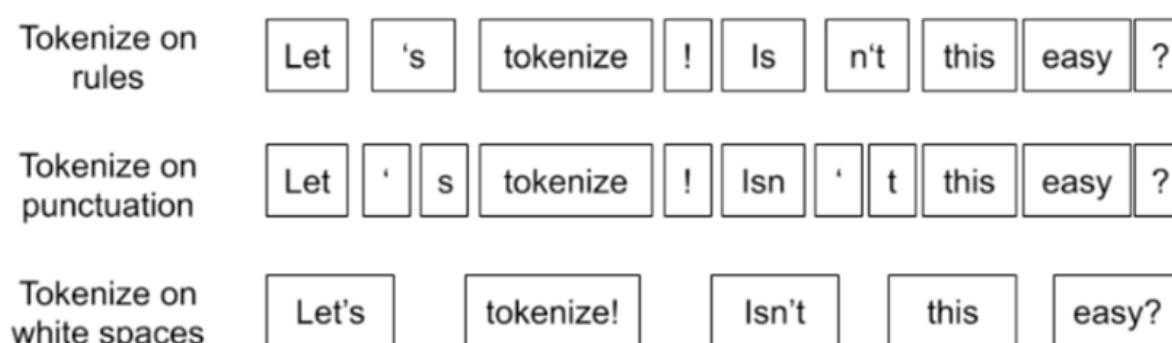


Figure 8 - types de tokenisation, (Horan, 2020)

La tokenisation peut être effectuée de différentes manières, mais il est important de choisir un tokeniseur adapté aux besoins et aux objectifs du projet. Par exemple, dans le cas où il serait utile de séparer des mots à l'espace, la tokenisation des *whitespace*, montrée sur la figure 8 pourrait être la solution la plus facile et la meilleure à mettre en œuvre. Une autre solution

serait de prendre en compte la ponctuation, mais cela peut être problématique pour des mots contenant des signes de ponctuation tels que l’apostrophe. La troisième approche de la tokenisation, sans doute la plus complexe, est l’approche basée sur des règles. Cette approche consiste à établir des règles grammaticales adaptées à la langue analysée afin d’assurer la qualité de la tokenisation.

7.3 Difficultés pour le corse

Il existe des tokeniseurs bien définis et bien complets à base de règles pour des langues dotées de plus de ressources comme le français ou l’anglais. Prenons l’exemple du modèle BERT, (Devlin et al., 2019). BERT, pour l’anglais, comprend un tokeniseur statistique avec un vocabulaire restreint mais contenant les mots et les sous-mots les plus récurrents de la langue. Ce tokeniseur est performant, permet de bien gérer la langue pour des tâches non spécifiques, et est modifiable. Cela est intéressant dans le cas où il faudrait utiliser un tokeniseur mieux adapté avec un vocabulaire plus spécialisé, par exemple pour des tâches dans le domaine médical.

Quelques caractéristiques du tokeniseur sont :

- Sa taille de vocabulaire de plus de 30 000 mots
- Des mots non reconnus ont le token [UNK]
- Des règles pour séparer la ponctuation des mots des phrases

Si nous prenons par exemple notre corpus corse, il serait possible de créer un *tokeniseur* à base de règles, mais en raison de la taille du corpus, il serait difficile de créer un *tokeniseur* aussi complet que celui des autres langues mieux dotées de ressources.

Pour rappel, notre corpus a environ 40 000 mots uniques, ce qui n’est pas beaucoup, mais à terme il faudrait prendre en compte plusieurs enjeux tels que les mots composés. Souvent, nous trouvons des apostrophes à l’intérieur des mots, ce qui mène parfois à un mauvais découpage par un tokeniseur classique qui n’a pas été développé pour la langue. Certains des mots doivent être séparés des apostrophes alors que d’autres doivent les garder.

Afin de pouvoir gérer ces difficultés, la création d’un tokeniseur de qualité nécessite une collaboration avec des locuteurs natifs et le développement de plus de corpus annotées.

7.5 Notre tokeniseur

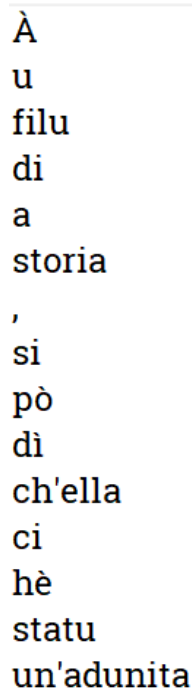
Pour ce projet, nous avons évalué des modèles d’annotation morphosyntaxique automatique sur notre corpus de 100 phrases annotées. Pour le tokeniseur, tokeniser sur les *whitespaces* sera le traitement le plus simple à ce stade. Nous n’avons pas accès à un corpus qui permette d’élaborer un *tokeniseur* plus sophistiqué avec un grand vocabulaire donc tokeniser sur les *whitespaces* paraît être la décision la plus logique.

J'ai codé un *tokeniseur* simple en Python qui prend un fichier texte en entrée et qui sort un fichier texte avec un mot par ligne. Le script cible les *whitespaces*, c'est-à-dire que les unités seront séparées après les espaces. À ce stade dans l'analyse de la langue, cette solution est la plus simple à mettre en place. Le script est rapide à exécuter et ne stocke qu'une ligne à la fois en mémoire. Ce format est nécessaire pour l'éventuelle prédiction d'étiquettes morphosyntaxiques.

Trattendu si di geulugia, si pò di ch'ella hè spartuta in dui a Balagna. In a so parte à nordu-punente, si stende un tarritoriu sedimintariu furmatu da e valle d'Ostriconi è di u Reginu : issu tarrenu abbastanza fe-rtile hà parmessu u sviluppu di rughjoni agriculi chì pruducianu assai. In a parte à u miziornu, truvemu a Balagna cristallina, fatta anzi tuttu di petre granitiche. Hè perciò chì a Balagna spicche da l'altre pieve di u Cismonte, chì appartenenu tutte à u tarritoriu matticciosu. Issa particolarità a ritruvemu in u patrimoniu architetturale, postu chì l'omu si hè sappiutu ghjuvà di e risorse ch'ella porghje issa forma di petra, ch'ella sia pè a custruzione o pè e faccende di tutti i ghjorni. Trattendu si di tupugraffia, a maiò parte di i paisaghji hè supranata da alte cime chì, da u miziornu à u miziornu-livante, franchenu i 2 000 metri. Sò disposte à arcu di chierchju è sò sulcate da

Figure 9 - texte brut

L'entrée ressemble à la figure 9 et il s'agit du texte séparé en paragraphes.



À
u
filu
di
a
storia
,
si
pò
di
ch'ella
ci
hè
statu
un'adunita

Figure 10 - texte tokénisé

La sortie ressemble à la figure 10, contenant un mot par ligne.

Le code se trouve dans l'annexe [1.3 Tokeniseur](#).

Ce script est basé sur le script partagé dans la section [5. Méthodologie](#) et s'agit d'une meilleure version du script précédent. Les améliorations sont :

- L'addition de la possibilité d'exécuter en ligne de commande
- Une meilleure gestion de la mémoire
- Une amélioration de la mise en forme qui permet de mieux comprendre le code
- L'ajout de plus de commentaires

Il était aussi une possibilité d'ajouter des expressions régulières afin de mieux découper le corpus et permettre un découpage de corpus futurs. Mais, pour découper le corpus actuel en mots individuels, cette approche fonctionne suffisamment bien.

8. Création de *word embeddings*/plongements lexicaux

8.1 Introduction

Dans cette section nous allons voir la différence entre plusieurs plongements lexicaux qui ont été testés dans l'entraînement de nos modèles d'annotation.

Les modèles d'annotation morphosyntaxique existants ont deux options :

1. Créer automatiquement les plongements lexicaux
2. Prendre en entrée des plongements pré-définis
3. Prendre des plongements existants pour une langue et les spécialiser pour une nouvelle langue

Dans le deuxième cas, nous avons plus de liberté pour choisir une représentation vectorielle permettant d'avoir des résultats optimaux dans nos analyses. Dans le cas aussi où nous avons un corpus de taille restreinte, il est important de bien choisir les outils et les procédures dans le but d'optimiser à un maximum les résultats obtenus.

8.2 Word2Vec

Word2Vec (Mikolov et al., 2013) est un groupe de modèles développé par une équipe de recherche chez Google. Word2Vec est l'un des outils les plus connus dans la création de plongements lexicaux. Développé en 2013, il est toujours utilisé et toujours utile. Ces plongements, qui sont des représentations vectorielles visent à modéliser la signification sémantique et syntaxique dans un corpus de texte donné. Les modèles de type Word2Vec utilisent deux stratégies principales afin de générer leurs représentations vectorielles des mots. Ce sont le *Continuous Bag of Words* et la méthode *skip gram*.

8.2.1 Continuous Bag of Words

Pour une représentation du type *continuous bag of words*, une fenêtre de contexte va être définie autour de chaque mot du texte.

Le *Continuous Bag of Words*, ou CBoW est appelé ainsi parce qu'il traite chaque contexte de mots comme un sac de mots ou "bag-of-words" en anglais. Le principe d'un sac est que l'ordre des mots n'est pas respecté. C'est-à-dire que l'ordre des mots dans le contexte n'est pas pris en compte. Cette propriété permet au modèle de capturer les relations entre les mots indépendamment de leur ordre dans la phrase. Ce modèle permet donc de prédire un mot basé sur les mots qui se trouvent autour du mot.

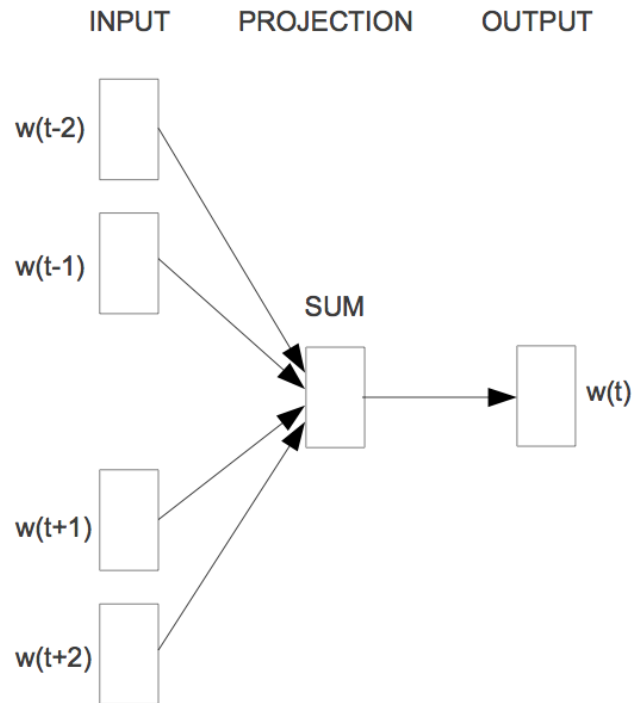


Figure 11 - continuous bag of words, (Mikolov et al., 2013)

Comme nous pouvons le constater du diagramme, le modèle prend en entrée les mots autour d'un certain mot et les utilise pour prédire le mot au milieu.

8.2.2 Skip Gram

Pour compléter le modèle CBoW, Word2Vec utilise aussi un modèle de type *skip gram*.

Dans le cas d'un modèle de type *skip gram*, nous pouvons dire que c'est comme un modèle CBoW, mais dans le sens inverse. Alors qu'avec un modèle CBoW nous essayons de prédire un mot basé sur son contexte, avec un modèle skip gram nous essayons de prédire les mots autour d'un mot.

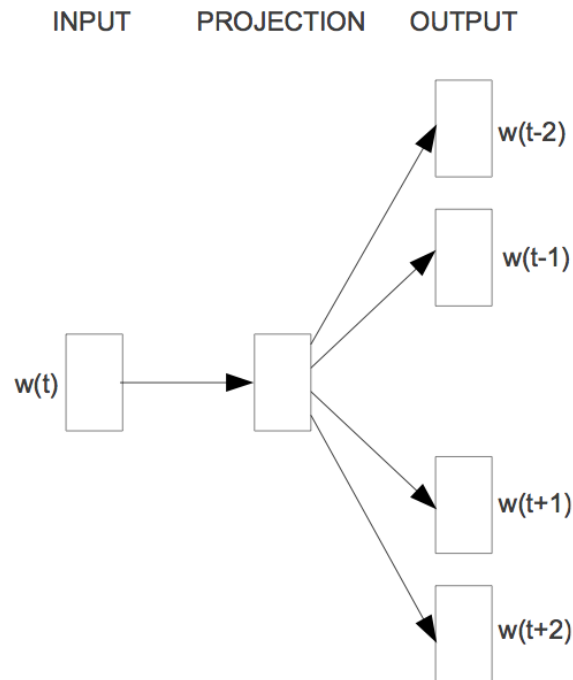


Figure 12 - skip gram, de (Mikolov et al., 2013)

En entrée nous voyons qu'il y a un mot qui sera pris en compte, puis le modèle va essayer de prédire les deux mots qui viennent avant et les deux mots qui viennent après.

8.3 FastText / Floret

FastText (Bojanowski et al., 2016), comme le nom le suggère, est une librairie très rapide de création de plongements lexicaux. La librairie a été développée par Facebook et elle est libre de droits permettant à tout le monde de l'utiliser. L'équipe travaille toujours sur le projet et il est donc mis à jour et documenté régulièrement. L'un des avantages de la librairie est la possibilité d'intégrer très facilement les plongements lexicaux dans un pipeline développé pour spaCy.

L'utilisation de la librairie pour générer des vecteurs ne peut pas être plus simple. L'exemple du site officiel montre la facilité de l'exécution :

```
$ mkdir result
$ ./fasttext skipgram -input data/fil9 -output result/fil9
```

Figure 13 - génération des vecteurs, (Bojanowski et al., 2016)

Comme nous pouvons le voir, il suffit de faire appel à la librairie, choisir un type de représentation vectorielle (skipgram ou CBoW) et d'ajouter les chemins relatifs vers les fichiers d'entrée et de sortie. Cette commande crée un fichier de type .vec qui contient donc un fichier avec un vecteur par mot. Ce fichier permet d'avoir des plongements lexicaux de manière rapide qui peuvent être ensuite utilisés dans nos analyses.

Floret, créé par Bojanowski et al., (2016) utilise un algorithme différent qui permet d'extraire plus d'informations des vecteurs en réduisant la taille des vecteurs. La différence principale entre FastText et Floret est décrite ci-dessous :

Pour stocker les vecteurs de mots et de sous-mots dans un format plus compact, nous utilisons un algorithme qui est utilisé depuis longtemps par spaCy : les "Bloom embeddings". Les Bloom embeddings (également appelés "*hashing trick*", ou connus sous le nom de HashEmbed dans la bibliothèque d'apprentissage automatique thinc de spaCy) peuvent être utilisés pour stocker des représentations distinctes dans une table compacte en séparant chaque entrée en plusieurs lignes dans la table. En représentant chaque entrée comme la somme de plusieurs lignes, où il est peu probable que deux entrées entrent en collision sur plusieurs *hashes*, la plupart des entrées auront une représentation distincte.¹⁸ (Notre traduction de Bojanowski et al., (2016))

Un exemple concret pour montrer ce processus vient de Boyd & Warmerdam, (2022)¹⁹ :

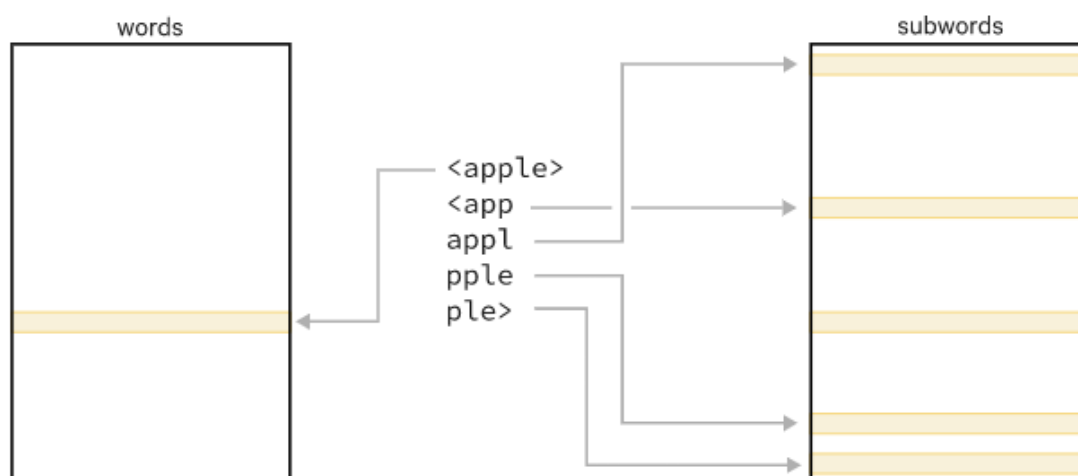


Figure 14 - séparation en mots et en sous-mots, Boyd & Warmerdam, (2022)

¹⁸ In order to store word and subword vectors in a more compact format, we turn to an algorithm that's been used by spaCy all along: Bloom embeddings. Bloom embeddings (also called the "hashing trick", or known as HashEmbed within spaCy's ML library thinc) can be used to store distinct representations in a compact table by hashing each entry into multiple rows in the table. By representing each entry as the sum of multiple rows, where it's unlikely that two entries will collide on multiple hashes, most entries will end up with a distinct representation.

¹⁹ <https://explosion.ai/blog/floret-vectors>

Dans cet exemple, des plongements ont été créés pour le mot “apple” (pomme) mais aussi pour des n-grammes de trois à quatre lettres que l’algorithme a trouvé pertinents à la représentation en plongements lexicaux. Cette analyse approfondie nous permettra d’exploiter à un maximum notre jeu de données afin d’avoir une analyse aussi complète que possible. Cela est aussi la différence entre FastText et un outil comme Word2Vec, qui ne prend en compte que les mots et non les n-grammes.

Un autre avantage de cet outil est la possibilité de créer rapidement nos plongements lexicaux. Cet outil est très rapide et permet, en quelques secondes de convertir une grande quantité de données en plongements lexicaux. La vitesse du calcul fait de cet outil le plus rapide que nous utilisons dans ce projet.

Un dernier avantage est l’existence de plongements lexicaux déjà créés pour 157 langues. Parmi les langues se trouvent le corse. Nous pouvons donc générer des plongements lexicaux à partir de nos corpus, mais aussi comparer la performance avec les plongements lexicaux corses de leur site.

Par contre, les corpus utilisés dans la génération de leurs plongements lexicaux corses viennent des corpus Wikipédia et Common Crawl. Nous avons déjà discuté de la fiabilité du corpus Wikipédia, et les mêmes arguments sont valables pour Common Crawl. Il est très possible que nos corpus soient plus performants une fois convertis en plongements lexicaux.

8.3.1 Intégration dans spaCy

Des vecteurs de type FastText/Floret peuvent être facilement ajoutés dans un *pipeline* spaCy créé pour l’annotation morphosyntaxique à partir d’une seule ligne de code :

```
spacy train config.cfg --path.vectors fichier.vec
```

L’intégration de ces représentations dans les pipelines de spaCy sera abordée en plus de détail dans la section [9. Utilisation et entraînement de spaCy pour le corse](#), mais il est intéressant de noter que FastText a été développé avec une intégration dans spaCy déjà prise en compte.

8.3.3 Utilité dans notre projet

Les premiers résultats obtenus après avoir testé la génération de plongements lexicaux de type FastText sont assez prometteurs. Les analyses paraissent assez complètes si nous regardons la documentation et la génération de plongements lexicaux est faite en très peu de temps.

L’outil est très simple à utiliser, puissant et facile à ajouter dans notre *pipeline* pour le projet. Ces plongements seront donc utilisés et testés contre d’autres plongements afin de voir lesquels sont les plus optimaux pour la tâche d’annotation morphosyntaxique.

8.4 Comparaison de Word2Vec et FastText

Alors qu'il serait très intéressant de comparer des résultats de plusieurs sortes de plongements lexicaux, je pense qu'il y a une représentation qui pourrait mieux fonctionner pour les fins de notre projet.

Word2Vec génère des plongements pour chaque mot, ce qui convient très bien à notre projet, mais il ne prend en compte que des mots et non des sous-mots. En plus, il n'arrive pas à bien gérer des mots qui ne se trouvent pas dans son vocabulaire. Word2Vec n'a pas non plus des vecteurs prédéfinis en corse, ce qui voudrait dire qu'il n'est peut-être pas très adaptable à de nouvelles langues.

FastText, en comparaison, est la méthode la plus récente entre FastText et Word2Vec. Les ressources semblent être plus récentes et plus pertinentes. FastText a déjà des représentations vectorielles du corse, qu'il faudra tester, ce qui montre que l'équipe derrière essaie de rendre disponible l'outil au plus grand nombre d'utilisateurs possible.

La chose la plus intéressante pour nos recherches est l'inclusion des sous-mots dans le traitement par FastText. Cette analyse permettrait d'extraire autant d'informations que possible de notre jeu de données restreint. Pour les analyses abordées dans les prochaines sections, nous allons donc utiliser les plongements lexicaux de type FastText. Nous allons comparer les résultats obtenus par FastText et Floret. Nous pouvons aussi comparer les résultats obtenus de leurs plongements et de nos propres plongements obtenus à partir du corpus du Canopé de Corse (Kevers et MacLean, 2023), (Kevers, 2022).

9. Utilisation et entraînement de spaCy pour le corse

spaCy, contenant des modèles déjà adaptés pour l'italien ainsi que la possibilité de créer nos propres modèles, sera utilisée pour voir quelles solutions sont les mieux adaptées à notre projet.

9.1 Processus

9.1.1 Pipeline d'entraînement

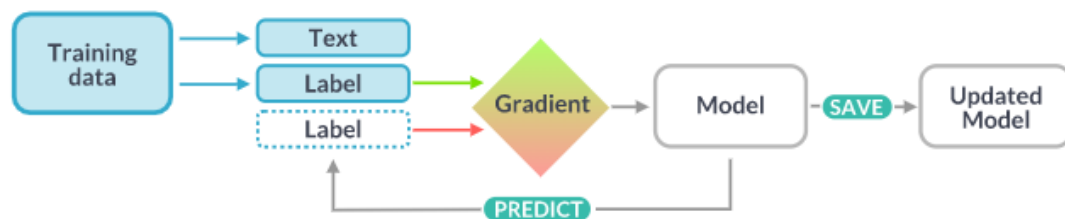


Figure 15 - pipeline d'entraînement, (Honnibal & Montani, 2017)

La figure 15 est issue du site de spaCy (Honnibal & Montani, 2017). Elle sert à décrire et montrer le processus d'entraînement à un haut niveau. Pour résumer le processus tel qu'il est décrit :

- Les données d'entraînement, converties en un format accepté par spaCy et contenant le token avec son étiquette sont utilisées.
- Les données sont séparées en texte, et l'étiquette ou *label* à prédire.
- Le gradient indique comment les valeurs de poids doivent être modifiées pour que les prédictions du modèle deviennent plus proches des étiquettes fournies.
- Le modèle est alors entraîné sur nos données, et les prédictions sont faites afin de calculer l'exactitude du modèle.
- Ce processus est itératif et se répète jusqu'à obtenir des résultats satisfaisants, ou jusqu'à ce que les résultats ne s'augmentent plus.

9.1.2 Format des données en entrée

Les données en entrée étaient sauvegardées en format CONLL. Ce format contient uniquement les tokens et les étiquettes associées avec une tabulation entre les deux colonnes. Ce premier format est nécessaire au traitement, mais il reste une dernière étape avant de pouvoir utiliser notre corpus avec spaCy. Il faut exécuter la commande suivante :

```
python -m spacy convert ./train.gold.conll ./corpus
```

Cette commande permet de convertir les données en format .spacy, qui est un format DocBin sérialisé contenant un ou plusieurs objets de type Doc, selon (Honnibal & Montani, 2017).

9.1.3 Expériences avec spaCy

Nous avons réalisé cinq expériences uniques, afin de voir quel modèle du corse fonctionne le mieux pour l'annotation automatique avec spaCy.

1. Pour commencer, nous avons supposé que le modèle italien arrivera à prédire au moins un pourcentage de mots corses supérieur à un choix aléatoire. La première expérience est donc l'annotation automatique à l'aide du modèle italien de base, c'est-à-dire avec les plongements lexicaux de base du modèle italien de spaCy.
2. La deuxième expérience utilise un modèle corse. Nous avons converti le corpus du Canopé en plongements lexicaux à l'aide de l'outil FastText.
3. La troisième expérience utilise également un modèle corse, mais celui créé par FastText sur les données de Wikipédia et de Common Crawl.
4. Ensuite, nous avons utilisé Floret, un outil basé sur FastText, pour tester le modèle corse que nous avons créé à partir du corpus Canopé.
5. Finalement, pour comparer, nous allons tester la performance d'un modèle français pour voir s'il y a des différences significatives de performance.

L'organisation des fichiers pour chaque expérience se trouve dans [la section 1.1 des Annexes](#).

9.3 Utilisation des plongements lexicaux pour l'italien de spaCy

Pour commencer, et pour avoir une expérience qui montre la précision d'un modèle italien sur notre corpus corse, la première expérience sera sans plongements lexicaux supplémentaires créés par nous. Les plongements sont donc les plongements italiens prêts à l'emploi de spaCy. Pour les utiliser, il suffit dans le fichier de configuration de préciser la langue. Si un modèle existe pour cette langue et que *vectors = null*, les plongements disponibles dans spaCy seront utilisés.

```
[nlp]
lang = "it"
```

Les chemins relatifs à définir dans le fichier de configuration sont donc les suivants :

```
[paths]
train = "CorPOS-UD_annot-AM_TRAIN_UPOS.conll.spacy"
dev = "CorPOS-UD_annot-AM_EVAL.conll.spacy"
vectors = null
```

Figure 16 - configuration des corpus d'entraînement et de développement

A noter, les corpus de *train* et de *dev* sont disjoints et ne contiennent pas les mêmes données. *Train* contient 80 % des données alors que *dev* contient 20 % des données. Pour l'instant, vu la taille restreinte de notre corpus, nous n'avons pas de corpus de test.

Ensuite, l'entraînement est lancé et spaCy prend en charge l'entraînement selon les spécifications du fichier de configuration. Avec cet outil, nous définissons que la tâche fait appel à leur modèle d'étiquetage morphosyntaxique ou *tagger*. Le modèle va aussi activer *tok2vec*, qui permettra à spaCy de vectoriser nos tokens afin de traiter les données en entrée.

Une fois l'entraînement lancé, nous voyons cet affichage dans le terminal :

```
===== Initializing pipeline =====
[2023-03-12 13:38:04,105] [INFO] Set up nlp object from config
[2023-03-12 13:38:04,120] [INFO] Pipeline: ['tok2vec', 'tagger']
[2023-03-12 13:38:04,126] [INFO] Created vocabulary
[2023-03-12 13:38:04,133] [INFO] Finished initializing nlp object
[2023-03-12 13:38:04,361] [INFO] Initialized pipeline components: ['tok2vec', 'tagger']
[+] Initialized pipeline

===== Training pipeline =====
[+] Pipeline: ['tok2vec', 'tagger']
[+] Initial learn rate: 0.001
```

E	#	LOSS TOK2VEC	LOSS TAGGER	TAG_ACC	SCORE
0	0	0.00	81.56	34.72	0.35
12	200	80.73	3283.26	74.45	0.74
29	400	21.16	62.33	75.34	0.75
49	600	9.37	18.61	73.65	0.74
73	800	0.81	1.31	74.45	0.74
103	1000	1.38	1.95	72.85	0.73
140	1200	0.05	0.06	72.42	0.72
185	1400	1.36	1.66	73.22	0.73
237	1600	0.04	0.04	73.43	0.73
304	1800	0.67	0.74	71.86	0.72
382	2000	22.05	19.44	73.74	0.74

Figure 17 - résultats de l'entraînement avec les plongements italiens de spaCy

Plusieurs champs sont alors affichés qui sont :

- E - le nombre d'itérations, *epoch* en anglais
- Loss Tok2Vec - la valeur de la perte de Tok2Vec
- Loss Tagger - la valeur de la perte du modèle d'étiquetage automatique
- Tag_Acc - l'exactitude du modèle d'étiquetage
- Score - l'exactitude arrondie

Pour Tok2Vec, ce paramètre est utilisé uniquement avec les plongements lexicaux prédéfinis dans spaCy et avec les plongements lexicaux de type Floret. spaCy a décidé d'ajouter du support pour les plongements lexicaux Floret en version 3.2 de la librairie²⁰. Lorsque nous

²⁰ <https://spacy.io/usage/v3-2>

utilisons des plongements lexicaux créés pour le corse avec un autre outil, Tok2Vec n'est pas affecté et reste donc à 0.

Pour notre première expérience, nous voyons qu'après 382 itérations, l'entraînement s'est arrêté. Au bout des 382 itérations, le modèle arrive à prédire la catégorie grammaticale des mots de nos corpus avec une exactitude (*SCORE*) de 75 %. Ce qui est intéressant, par contre, est que le modèle trouve sa meilleure performance après seulement 29 itérations. A la fin de ce processus, spaCy va sauvegarder automatiquement deux modèles : le dernier et celui avec la meilleure performance.

Ce que nous pouvons constater à ce stade est que le modèle est capable de prédire la bonne catégorie grammaticale trois fois sur quatre. C'est donc bien supérieur à un choix aléatoire et très bien pour un modèle qui n'est pas configuré spécifiquement pour le traitement du corse.

9.4 Entraînement du modèle corse - nos plongements FastText

Alors que le processus sera très similaire à celui de la précédente expérience, il faut commencer par la création de nos plongements lexicaux FastText.

```
===== Initializing pipeline =====
[2023-03-20 17:53:38,504] [INFO] Set up nlp object from config
[2023-03-20 17:53:38,519] [INFO] Pipeline: ['tok2vec', 'tagger']
[2023-03-20 17:53:38,524] [INFO] Created vocabulary
[2023-03-20 17:53:38,724] [INFO] Added vectors: pipeline_co
[2023-03-20 17:53:38,751] [INFO] Finished initializing nlp object
[2023-03-20 17:53:39,022] [INFO] Initialized pipeline components: ['tok2vec', 'tagger']
[+] Initialized pipeline

===== Training pipeline =====
[+] Pipeline: ['tok2vec', 'tagger']
[+] Initial learn rate: 0.001
```

E	#	LOSS TOK2VEC	LOSS TAGGER	TAG_ACC	SCORE
0	0	0.00	81.56	37.00	0.37
12	200	0.00	3199.46	74.70	0.75
29	400	0.00	80.19	74.11	0.74
49	600	0.00	24.32	74.70	0.75
73	800	0.00	1.71	73.22	0.73
103	1000	0.00	9.17	74.45	0.74
140	1200	0.00	0.74	72.67	0.73
185	1400	0.00	0.16	74.02	0.74
237	1600	0.00	0.29	74.02	0.74
304	1800	0.00	4.75	74.24	0.74

Figure 18 - résultats de l'entraînement avec nos plongements corses de FastText

Après l'entraînement, nous constatons des résultats similaires que l'expérience précédente, mais légèrement moins performants. Les résultats sont toujours bons pour un corpus de cette taille, mais il va falloir continuer à faire des expériences sur plus de données pour voir si la performance est vraiment moins bien que l'expérience précédente.

9.5 Entraînement du modèle corse - plongements corses disponibles sur le site de FastText

Pour cette expérience, le processus reste largement pareil au précédent. Cette fois, nous n'avons pas à créer des plongements lexicaux nous-mêmes et il faut tout simplement les récupérer sur le site de FastText²¹. Ces plongements ont été créés à partir du corpus du Wikipédia corse.

Le processus est de plus en plus rapide à exécuter car le seul changement est le chemin vers les plongements lexicaux.

```
===== Initializing pipeline =====
[2023-03-20 17:56:49,293] [INFO] Set up nlp object from config
[2023-03-20 17:56:49,305] [INFO] Pipeline: ['tok2vec', 'tagger']
[2023-03-20 17:56:49,310] [INFO] Created vocabulary
[2023-03-20 17:56:50,336] [INFO] Added vectors: pipeline_co
[2023-03-20 17:56:51,226] [INFO] Finished initializing nlp object
[2023-03-20 17:56:51,602] [INFO] Initialized pipeline components: ['tok2vec', 'tagger']
█ Initialized pipeline

===== Training pipeline =====
█ Pipeline: ['tok2vec', 'tagger']
█ Initial learn rate: 0.001
E   #      LOSS TOK2VEC   LOSS TAGGER   TAG_ACC   SCORE
---  ---
  0      0          0.00         81.56      35.27     0.35
 12     200          0.00       3262.83      74.92     0.75
 29     400          0.00         62.80      72.76     0.73
 49     600          0.00         32.47      74.87     0.75
 73     800          0.00          5.67      73.19     0.73
103    1000          0.00          5.61      73.10     0.73
140    1200          0.00          0.06      73.77     0.74
185    1400          0.00          2.41      72.08     0.72
237    1600          0.00          6.55      73.43     0.73
304    1800          0.00         10.85      73.65     0.74
```

Figure 19 - résultats de l'entraînement avec les plongements corses du site de FastText

Les résultats sont toujours bons, mais vu qu'il n'y a pas une différence significative entre les résultats ici et dans les deux expériences précédentes, il va falloir continuer à évaluer la performance de nos plongements lexicaux sur des corpus plus conséquents.

9.6 Entraînement du modèle corse - nos plongements Floret

²¹ <https://fasttext.cc/docs/en/crawl-vectors.html>

```

===== Initializing pipeline =====
[2023-03-20 18:10:00,136] [INFO] Set up nlp object from config
[2023-03-20 18:10:00,150] [INFO] Pipeline: ['tok2vec', 'tagger']
[2023-03-20 18:10:00,154] [INFO] Created vocabulary
[2023-03-20 18:10:00,431] [INFO] Added vectors: pipeline_co
[2023-03-20 18:10:00,795] [INFO] Finished initializing nlp object
[2023-03-20 18:10:00,970] [INFO] Initialized pipeline components: ['tok2vec', 'tagger']
[+] Initialized pipeline

===== Training pipeline =====
[+] Pipeline: ['tok2vec', 'tagger']
[+] Initial learn rate: 0.001
E      #      LOSS TOK2VEC  LOSS TAGGER  TAG_ACC  SCORE
-----
  0      0      0.00      81.56      38.19      0.38
 12     200     89.31     3613.02     76.27     0.76
 29     400     32.81      95.82     75.25     0.75
 49     600     14.20     27.56     73.90     0.74
 73     800      0.13      0.19     73.77     0.74
103    1000      4.34      5.76     73.77     0.74
140    1200      4.67      5.59     73.43     0.73
185    1400      0.04      0.05     73.34     0.73
237    1600      0.48      0.57     72.54     0.73
304    1800     31.08     29.33     73.90     0.74

```

Figure 20 - résultats de l'entraînement avec nos plongements Floret

Les plongements lexicaux Floret sont effectivement les plus performants dans l'annotation morphosyntaxique à l'aide de spaCy. Les résultats sont les meilleurs obtenus avec spaCy. En seulement douze itérations, le modèle dépasse les 75 % d'exactitude. Afin de continuer les expériences, il serait intéressant d'évaluer ce modèle sur un plus grand corpus afin de voir si le modèle est vraiment performant sur un corpus plus conséquent.

9.6.2 Représentation complète

Si nous enlevons l'argument -bucket de la commande Floret, nous allons avoir une représentation vectorielle qui est quarante fois plus grande. Le fichier contenant les plongements lexicaux fait plus de 6,2 Go.

```

[+] Creating blank nlp object for language 'it'
[2023-03-14 20:43:46,211] [INFO] Reading vectors from model.floret
2000000it [02:16, 14631.90it/s]
[2023-03-14 20:46:02,921] [INFO] Loaded vectors from model.floret
[+] Successfully converted 2000000 vectors
[+] Saved nlp object with vectors to output directory. You can now use
the path to it in your config as the 'vectors' setting in [initialize].

```

Figure 21 - initialisation d'une représentation complète des plongements Floret

L'initialisation de ces vecteurs a pris plus de temps que l'initialisation des vecteurs précédents.

```

===== Initializing pipeline =====
[2023-03-20 18:04:20,460] [INFO] Set up nlp object from config
[2023-03-20 18:04:20,471] [INFO] Pipeline: ['tok2vec', 'tagger']
[2023-03-20 18:04:20,475] [INFO] Created vocabulary
[2023-03-20 18:04:32,339] [INFO] Added vectors: pipeline_co
[2023-03-20 18:05:46,811] [INFO] Finished initializing nlp object
[2023-03-20 18:05:48,400] [INFO] Initialized pipeline components: ['tok2vec', 'tagger']
[+] Initialized pipeline

===== Training pipeline =====
[+] Pipeline: ['tok2vec', 'tagger']
[+] Initial learn rate: 0.001
E      #      LOSS TOK2VEC  LOSS TAGGER  TAG_ACC  SCORE
---  ---  ---
  0      0      0.00      81.56      34.67      0.35
 12     200     92.86     3615.37     73.31     0.73
 29     400     29.71      86.21     72.97     0.73
 49     600     10.51     19.75     72.97     0.73
 73     800      2.12      3.45     70.95     0.71
103    1000      3.33      4.34     72.85     0.73
140    1200      5.92      7.28     71.53     0.72
185    1400      6.02      6.19     72.42     0.72
237    1600      2.89      2.93     71.65     0.72
304    1800      7.42      7.12     71.07     0.71

```

Figure 22 - résultats de l'entraînement avec nos plongements complets de type Floret

Le processus reste le même, mais l'entraînement prend plus de temps. Notamment, les plongements lexicaux ont été ajoutés au *pipeline* en douze secondes, alors que d'habitude ils sont ajoutés en moins d'une seconde. L'entraînement a aussi pris plus de temps que les autres et mon ordinateur s'est figé pendant plusieurs minutes. Mais, finalement, les résultats sont légèrement moins bons que dans la précédente expérience avec seulement 50 000 vecteurs ajoutés.

9.7 Utilisation des plongements français de spaCy

Pour finir, nous allons examiner le modèle français de spaCy pour voir si les résultats changent et si les résultats sont toujours acceptables.

Donc, nous allons tout simplement changer la langue dans le fichier de configuration. Cela permet d'utiliser tout de suite les plongements lexicaux français de spaCy.

```

[nlp]
lang = "fr"

```

Ensuite, il faut enlever le chemin vers des plongements lexicaux. Cela va permettre à spaCy d'utiliser les plongements lexicaux du modèle français.

L'entraînement s'initialise encore à l'aide de la commande :

```
python -m spacy train config.cfg
```



```

===== Initializing pipeline =====
[2023-03-12 13:44:03,228] [INFO] Set up nlp object from config
[2023-03-12 13:44:03,241] [INFO] Pipeline: ['tok2vec', 'tagger']
[2023-03-12 13:44:03,246] [INFO] Created vocabulary
[2023-03-12 13:44:03,247] [INFO] Finished initializing nlp object
[2023-03-12 13:44:03,434] [INFO] Initialized pipeline components: ['tok2vec', 'tagger']
[+] Initialized pipeline

===== Training pipeline =====
[+] Pipeline: ['tok2vec', 'tagger']
[+] Initial learn rate: 0.001
E   #      LOSS TOK2VEC  LOSS TAGGER  TAG_ACC  SCORE
---  ---  -
  0      0      0.00      81.56      34.84      0.35
 12     200     80.73     3283.26     74.70     0.75
 29     400     21.16      62.33     75.59     0.76
 49     600      9.37     18.61     73.90     0.74
 73     800      0.81      1.31     74.92     0.75
103    1000      1.38      1.95     73.10     0.73
140    1200      0.05      0.06     72.67     0.73
185    1400      1.36      1.66     73.34     0.73
237    1600      0.04      0.04     73.56     0.74

```

Figure 23 - résultats de l'entraînement avec les plongements français de spaCy

Les résultats de cet entraînement sont assez surprenants. Le meilleur modèle que nous avons entraîné avait une exactitude de 76,27 % . Avec le modèle français, nous avons donc une exactitude de 75,59 % du meilleur modèle. Donc, nous constatons une exactitude correcte avec le modèle français de base, mais la meilleure exactitude se trouve à l'aide de notre modèle corse créé avec Floret..

9.8 Discussion sur les résultats

9.8.1 Résultats

Les modèles corses, italien et français arrivent à bien trouver la catégorie grammaticale des mots de notre jeu de données d'entraînement trois fois sur quatre en moyenne. Ces résultats sont au-dessus de mes attentes et sont très bons pour un premier entraînement.

Les résultats que nous voyons sont sur un corpus d'entraînement restreint de seulement cent phrases. Il est envisageable de refaire ces expériences avec un corpus plus long et plus détaillé, ce qui en théorie permettrait d'augmenter l'exactitude des modèles. L'ajout de plongements lexicaux créés à partir de plus grands corpus de textes augmenteraient aussi l'exactitude des modèles, mais il faut donc plus de ressources qui ne sont pas disponibles en ce moment.

9.8.2 Hypothèse

Avant de commencer les entraînements, nous avons supposé plusieurs choses. Premièrement, nous avons pensé que les meilleurs résultats seraient obtenus à l'aide d'un modèle italien.

Deuxièmement, nous pensions que nos vecteurs auraient un effet positif sur les résultats finaux. Nous pensions aussi que nos vecteurs, créés à partir d'un corpus propre de bonne qualité, seraient plus performant que les vecteurs FastText disponibles sur le site du module. Mon hypothèse était correcte, les plongements lexicaux les plus performants étaient ceux qui ont été générés par Floret à partir de notre corpus. La performance s'explique aussi en raison de l'entraînement Tok2Vec qui affecte les plongements Floret. Les vecteurs sont entraînés et modifiés lors de l'étape de l'entraînement, ce qui explique les valeurs élevées.

Il est également important de préciser que nous disposions uniquement de cent phrases, réparties dans un corpus d'entraînement (80 phrases) et un corpus de développement (20 phrases). Ces corpus ont été utilisés à la fois pour entraîner et valider les modèles. Cela signifie que les modèles ont utilisé l'ensemble de nos phrases pendant l'entraînement, ce qui peut introduire des biais dans leurs analyses.

10. Flair

10.1 Introduction

Flair est une librairie créée pour le TAL et qui est particulièrement utile dans la reconnaissance d’entités nommées et l’annotation morphosyntaxique. Les modèles disponibles sont, pour l’instant, moins nombreux que les modèles de spaCy. Cependant, la librairie gagne en popularité ces dernières années et sa performance sur la tâche d’annotation morphosyntaxique est similaire à la performance obtenue à l’aide de spaCy.

10.1.2 Données

Les données utilisées dans ces expériences sont les mêmes que nous avons utilisées dans les expériences précédentes. Nous avons toujours le même corpus de cent phrases annotées qui vont servir à nous donner un ordre d’idée de la performance possible avec les modèles d’annotation morphosyntaxique de Flair. Le corpus d’entraînement représente 80 % de nos données alors que le corpus de validation (dev) représente 20 % de nos données.

10.1.3 Modèles

Le modèle que nous allons tester est le modèle “pos-multi”, un modèle contenant des plongements lexicaux pour 12 langues, dont le français et l’italien. À terme, si la librairie est utile, il est possible d’ajouter le corse. Ce modèle a été entraîné sur les UD Treebanks (Nivre et al., 2020). À noter, il semble que le modèle “pos-multi” est la dernière version du modèle “upos-multi”. Le lien présent sur le site de Flair précise le nom “pos-multi” mais mène à la page du modèle “upos-multi”. Quelques tests ont été effectués sur les deux modèles, et le modèle “pos-multi” est le plus adapté à notre projet. Les expériences suivantes ont donc été faites à l’aide du modèle “pos-multi”.

10.1.4 Plongements lexicaux

Alors qu’il est possible d’ajouter nos propres vecteurs à Flair, l’ajout des vecteurs du type Floret, les plus performants des expériences avec spaCy, n’est pas possible dans l’immédiat. Nous proposons donc d’utiliser les plongements lexicaux Flair.

Nous allons tester trois vecteurs différents qui sont accessibles à l’intérieur de Flair :

1. Les vecteurs italiens
2. Les vecteurs multi-fast
3. Les vecteurs multi

Ma première hypothèse est que les vecteurs multi seront les plus performants. Ces vecteurs contiennent le plus de représentations et il est possible que cela aidera le modèle à reconnaître nos phrases en corse. Il sera aussi intéressant de voir la différence entre multi et multi-fast en termes de performance et en temps d'exécution de l'entraînement.

Une deuxième spécificité des plongements lexicaux Flair est la possibilité d'ajouter plusieurs plongements à la fois, à l'aide de la fonction "StackedEmbeddings". La recommandation de Flair est d'utiliser des plongements créés à partir des textes lus de la gauche à la droite ainsi que des plongements créés à partir des textes lus de la droite à la gauche. Ces plongements sont désignés par le nom de la langue et l'argument *forward* ou *backward*.

10.1.5 Les étiquettes

Flair utilise un jeu d'étiquettes des Universal Dependencies (de Marneffe et al., 2021) et ajoute également des étiquettes pour désigner le début et la fin d'une phrase. Les étiquettes sont :

<unk>, NOUN, DET, PUNCT, ADP, VERB, PRON, ADV, PROPN, ADJ, SCONJ, CCONJ, NUM, AUX, X, VERB+PRON, INTJ, PART, VEBR, <START>, <STOP>

L'étiquette <unk> désigne un token non reconnu. Les étiquettes <START> et <STOP> désignent le début et la fin d'une phrase.

10.1.6 Les mesures de performance

Les mesures de performance que nous allons utiliser dans l'analyse sont automatiquement calculées par Flair. Elles sont :

- Précision : La précision mesure la proportion d'étiquettes prédites par le modèle qui sont correctes par rapport à l'ensemble des étiquettes prédites.
- Rappel (*recall*): Le rappel mesure la proportion d'étiquettes correctes prédites par le modèle par rapport à l'ensemble des étiquettes appartenant à la classe.
- F1 : Le score F1 est une mesure de la performance globale du modèle, calculée en combinant la précision et le rappel. Il est aussi appelé la moyenne harmonique des deux mesures.
- Support : Le support mesure le nombre d'occurrences d'une classe dans l'ensemble de données d'entraînement. Il peut être utilisé pour évaluer la représentativité des classes dans l'ensemble de données et pour identifier les classes sous-représentées qui peuvent poser des problèmes pour la performance du modèle.
- Exactitude (accuracy)

- Dans Flair, l'exactitude est calculée à l'aide de la formule suivante :
 - $\text{vrais positifs} / (\text{vrais positifs} + \text{faux positifs} + \text{faux négatifs})$

10.2 Plongements italiens Flair

Pour commencer les expériences à l'aide de Flair, nous allons utiliser les plongements italiens qui sont déjà intégrés dans Flair. Un script d'entraînement complet se trouve [dans la section 11.2](#).

Le script, où nous allons exploiter les mêmes données, ne changera pas beaucoup lors de chaque entraînement. La seule chose qu'il faut spécifier est le choix de plongements lexicaux Flair. Les plongements lexicaux, ainsi que le modèle qui sera utilisé (pos-multi) sont téléchargés automatiquement par Flair.

Donc, pour la première expérience, nous allons définir les plongements lexicaux ici :

```
stacked_embeddings = StackedEmbeddings([  
  
    FlairEmbeddings('it-forward'),  
  
    FlairEmbeddings('it-backward'),  
  
    ])
```

Pour cette expérience, le nombre d'itérations est fixé à 10. Le seul paramètre qui changera pendant ces expériences est le nombre d'itérations (*epochs*). Des plongements figés sont utilisés et cette approche ne s'agit pas du *fine-tuning*. Il faut noter que l'entraînement s'arrêtera automatiquement si l'exactitude arrête d'augmenter d'une itération à l'autre.

```
trainer.train('resources/taggers/pos-multi',  
              learning_rate=0.1,  
              mini_batch_size=32,  
              max_epochs=10)
```

Les modèles de ce type sont plus lents que ceux de spaCy et les entraînements peuvent durer jusqu'à plusieurs minutes. Les résultats de cette première expérience, affichés automatiquement, sont les suivants :

	precision	recall	f1-score	support
NOUN	0,6716	0,8036	0,7317	56
DET	0,8235	0,8750	0,8485	48
PUNCT	0,8400	0,9545	0,8936	44
PRON	0,1964	0,6111	0,2973	18
ADP	0,8571	0,7500	0,8000	32
VERB	0,4706	0,2051	0,2857	39
ADV	0,1667	0,1429	0,1538	14
PROPN	0,3333	0,5000	0,4000	6
ADJ	0,0000	0,0000	0,0000	12
CCONJ	0,0000	0,0000	0,0000	6
SCONJ	0,0000	0,0000	0,0000	5
NUM	1,0000	0,2500	0,4000	4
AUX	0,0000	0,0000	0,0000	3
PART	0,0000	0,0000	0,0000	2
NUU	0,0000	0,0000	0,0000	1
X	0,0000	0,0000	0,0000	1
INTJ	0,0000	0,0000	0,0000	1
SYM	0,0000	0,0000	0,0000	1
accuracy			0,6075	293
macro avg	0,2977	0,2829	0,2673	293
weighted avg	0,5862	0,6075	0,5777	293

Tableau 6 - résultats de l'entraînement avec les plongements italiens de Flair

Cette première expérience, avec une exactitude plus faible que ce que nous avons pu voir avec spaCy, montre que la bonne catégorie grammaticale est choisie trois fois sur cinq. Cette

exactitude, sur 21 catégories montre que ce modèle, avec ces plongements lexicaux, peut être entraîné et raffiné sur un plus grand corpus de données textuelles corse. Mais, ce modèle n'est pas le meilleur pour le moment, et des plongements lexicaux qui ne viennent pas de Flair sont difficilement ajoutés aux entraînements de l'outil.

10.3 Plongements multilingues Flair - version compacte

Pour continuer les expériences à l'aide de Flair, nous allons utiliser les plongements multilingues qui sont déjà intégrés dans Flair. Nous allons garder le même format avec des plongements lexicaux entraînés dans les deux sens. Il est possible que ces plongements soient plus performants car le modèle à entraîner s'attend à des plongements lexicaux multilingues. Pour des raisons de performance, nous allons commencer par tester la version compacte des plongements lexicaux.

```
stacked_embeddings = StackedEmbeddings([  
  
    FlairEmbeddings('multi-forward-fast'),  
  
    FlairEmbeddings('multi-backward-fast'),  
  
    ])
```

Nous allons donc voir les résultats d'un entraînement simple avec dix itérations :

	precision	recall	f1-score	support
NOUN	0,3717	0,7500	0,4970	56
PUNCT	0,5070	0,8182	0,6261	44
DET	0,5357	0,6250	0,5769	48
VERB	0,3000	0,1538	0,2034	39
ADP	0,7333	0,3438	0,4681	32
ADV	0,1111	0,1429	0,1250	14
PRON	0,0000	0,0000	0,0000	18
ADJ	0,0000	0,0000	0,0000	12
CCONJ	0,0000	0,0000	0,0000	6
PROPN	0,0000	0,0000	0,0000	6
SCONJ	0,0000	0,0000	0,0000	5
NUM	0,0000	0,0000	0,0000	4
AUX	0,0000	0,0000	0,0000	3
PART	0,0000	0,0000	0,0000	2
NUU	0,0000	0,0000	0,0000	1
X	0,0000	0,0000	0,0000	1
INTJ	0,0000	0,0000	0,0000	1
SYM	0,0000	0,0000	0,0000	1
accuracy			0,4334	293
macro avg	0,1422	0,1574	0,1387	293
weighted avg	0,3603	0,4334	0,3677	293

Tableau 7 - résultats de l'entraînement avec les plongements multilingues compacts de Flair

Ces plongements lexicaux rendent le modèle moins performant, et, cette expérience nous a donné les pires résultats. Le modèle est capable de prédire la bonne catégorie grammaticale deux fois sur cinq. Cela n'est pas mauvais pour un modèle dont les plongements ne sont pas adaptés au corse, mais nous avons des modèles bien plus performants qui seraient plus utiles à utiliser pour la suite.

10.4 Plongements multilingues Flair

Malgré les résultats plutôt décevants que nous avons eus avec les plongements lexicaux compacts, il serait intéressant de voir s'il y a un gain de performance si nous utilisons les plongements lexicaux 'multi' normaux.

Il fallait donc définir les plongements que nous utilisons ici :

```
stacked_embeddings = StackedEmbeddings([  
  
    FlairEmbeddings('multi-forward'),  
  
    FlairEmbeddings('multi-backward'),  
  
    ])
```

Comme dans l'expérience précédente, nous allons voir les résultats d'un entraînement avec 10 itérations :

	precision	recall	f1-score	support
DET	0,5393	1,0000	0,7007	48
NOUN	0,6133	0,8214	0,7023	56
PUNCT	0,9778	1,0000	0,9888	44
VERB	0,4667	0,3590	0,4058	39
ADP	0,7879	0,8125	0,8000	32
ADD	0,2222	0,1667	0,1905	12
PRON	0,0000	0,0000	0,0000	18
ADV	0,0000	0,0000	0,0000	14
SCONJ	0,2500	0,4000	0,3077	5
PROPN	0,6667	0,3333	0,4444	6
CCONJ	0,0000	0,0000	0,0000	6
NUM	0,0000	0,0000	0,0000	4
AUX	0,0000	0,0000	0,0000	3
PART	0,0000	0,0000	0,0000	2
NUU	0,0000	0,0000	0,0000	1
X	0,0000	0,0000	0,0000	1
INTJ	0,0000	0,0000	0,0000	1
SYM	0,0000	0,0000	0,0000	1
accuracy			0,6280	293
macro avg	0,2513	0,2718	0,2522	293
weighted avg	0,5276	0,6280	0,5610	293

Tableau 8 - résultats de l'entraînement avec les plongements multilingues de Flair

Ces résultats, contre mes attentes, sont les meilleurs pour un entraînement avec Flair. Nous avons un gain d'exactitude de presque 20 % comparé aux résultats avec la version compacte des mêmes plongements lexicaux.

10.5 Plongements multilingues Flair, 100 itérations

Afin de voir la vraie performance du modèle ‘pos-multi’ avec les plongements ‘multi’, nous allons voir les résultats après une centaine d’itérations. Donc, le seul changement à faire dans le script d’entraînement est la valeur de la variable ‘max_epochs’.

```
trainer.train('resources/taggers/pos-multi',  
              learning_rate=0.1,  
              mini_batch_size=32,  
              max_epochs=100)
```

Après cet entraînement, nous voyons que l'entraînement s'est arrêté à 81 itérations.

Les résultats sont ci-dessous :

	precision	recall	f1-score	support
NOUN	0,7627	0,8036	0,7826	56
DET	0,8431	0,8958	0,8687	48
VERB	0,6038	0,8205	0,6957	39
PUNCT	0,9565	1,0000	0,9778	44
ADP	0,9677	0,9375	0,9524	32
PRON	0,5263	0,5556	0,5405	18
ADV	0,8333	0,3571	0,5000	14
ADJ	0,3750	0,2500	0,3000	12
PROPN	0,6250	0,8333	0,7143	6
CCONJ	1,0000	0,8333	0,9091	6
SCONJ	0,6667	0,4000	0,5000	5
NUM	0,7500	0,7500	0,7500	4
AUX	0,0000	0,0000	0,0000	3
PART	0,0000	0,0000	0,0000	2
NUU	0,0000	0,0000	0,0000	1
X	0,0000	0,0000	0,0000	1
INTJ	0,0000	0,0000	0,0000	1
SYM	0,0000	0,0000	0,0000	1
accuracy			0,7747	293
macro avg	0,4950	0,4687	0,4717	293
weighted avg	0,7560	0,7747	0,7567	293

Tableau 9 - résultats de l'entraînement avec les plongements multilingues de Flair sur 81 itérations

Ces résultats sont impressionnants et sont les meilleurs de Flair. Nous avons un gain d'exactitude de presque 15 % par rapport à notre dernier entraînement de dix itérations. Nous avons aussi un gain d'exactitude de plus de 1 % par rapport à notre meilleur entraînement

avec spaCy. Il faut noter que l'entraînement a duré presque 20 minutes alors que la plupart des entraînements ont pris seulement une ou deux minutes.

10.6 Plongements multilingues Flair, version compacte, 100 itérations

Afin de voir la vraie performance du modèle ‘pos-multi-fast’ et de comparer la performance avec celle de ‘pos-multi’, nous allons voir les résultats après une centaine d’itérations. Tous les paramètres de l’entraînement sont les mêmes, le seul changement à noter est le changement de plongements lexicaux.

	precision	recall	f1-score	support
NOUN	0.7636	0.7500	0.7568	56
DET	0.8980	0.9167	0.9072	48
PUNCT	0.9565	1.0000	0.9778	44
VERB	0.6136	0.6923	0.6506	39
ADP	0.7692	0.9375	0.8451	32
PRON	0.5909	0.7222	0.6500	18
ADV	0.6667	0.4286	0.5217	14
ADJ	0.4000	0.3333	0.3636	12
PROPN	0.7143	0.8333	0.7692	6
CCONJ	0.8000	0.6667	0.7273	6
SCONJ	0.5000	0.2000	0.2857	5
NUM	0.3333	0.2500	0.2857	4
AUX	0.5000	0.3333	0.4000	3
PART	0.0000	0.0000	0.0000	2
NUÙ	0.0000	0.0000	0.0000	1
X	0.0000	0.0000	0.0000	1
INTJ	0.0000	0.0000	0.0000	1
SYM	0.0000	0.0000	0.0000	1
accuracy			0.7577	293
macro avg	0.4726	0.4480	0.4523	293
weighted avg	0.7361	0.7577	0.7423	293

Tableau 10 - résultats de l'entraînement avec les plongements multilingues compacts de Flair sur 100 itérations

Ces résultats montrent une performance impressionnante, mais le meilleur modèle reste celui entraîné sur les plongements “multi-pos”. Ici nous constatons une baisse en exactitude de 1.7 points de pourcentage.

10.7 Discussion sur les résultats

Alors que Flair n’est pas aussi développé que spaCy, nous avons eu des résultats assez surprenants à l’aide de cet outil. Un ajout marquant par rapport à l’autre librairie que nous avons utilisé est la possibilité d’utiliser des plongements lexicaux dans deux sens. C’est-à-dire, les plongements lexicaux sont générés avec leurs contextes de gauche à droite et ensuite de droite à gauche. Cela permet de rajouter plus d’informations à l’analyse et permet ainsi à une augmentation de sa performance.

A noter aussi, nous avons eu les meilleurs résultats non avec le modèle italien, mais avec le modèle multi-pos, qui contient des plongements lexicaux pour plusieurs langues. Avec une exactitude de 77,47 %, ce modèle est définitivement le plus performant de Flair et donc le plus intéressant pour le cas de l’annotation morphosyntaxique du corse.

À nouveau, il faut également considérer que les modèles ont utilisé un petit jeu de données de cent phrases à la fois pour entraîner et valider le modèle. Les modèles ont donc utilisé l’ensemble de ces données lors des entraînements, ce qui peut introduire des biais dans leurs analyses.

11. Comparaison de Flair et spaCy

11.1 Introduction

Les deux librairies testées, spaCy (Honnibal & Montani, 2017) et Flair (Akbik et al., 2019), sont bien adaptées à la tâche d'annotation morphosyntaxique du corse. Notre métrique de performance, l'exactitude, a été mesurée pour toutes les expériences effectuées. Nous avons trouvé une exactitude de 76,27 % dans le cas de spaCy et de 77,47 % dans le cas de Flair. L'utilisation de Flair représente un gain de 1,57 points de pourcentage. Ces modèles ont été entraînés et évalués sur un petit jeu de données, avec les mêmes 20 phrases utilisées dans le corpus de développement. Cela peut introduire des biais dans les modèles.

Les deux librairies sont très bien documentées, mais il y a une différence significative dans leurs fonctionnalités. Flair permet de créer des plongements lexicaux des contextes de la gauche à la droite et de la droite à la gauche. D'après ce que j'ai pu constater de sa performance, cela a un impact positif sur l'entraînement. Malheureusement, spaCy ne dispose pas encore de la possibilité d'utiliser des plongements de ce type et la librairie ne peut utiliser un fichier de plongements lexicaux à la fois. Donc, ce mémoire cherche à trouver quelle librairie et quel modèle au sein de la librairie serait la mieux adaptée à notre cas.

Alors que les deux librairies testées attestent de performances significatives dans le cadre de notre projet, il n'y a clairement pas de meilleure librairie à utiliser. Il y a des différences dans le processus de traitement, y compris le niveau de Python requis et la facilité de modifier les plongements lexicaux.

11.2 Facilité d'utilisation

11.2.1 Flair

Flair (Akbik et al., 2019), la librairie la plus performante, est aussi la librairie qui nécessite le plus de compétences en programmation afin de s'en servir. Afin d'instancier leur module d'annotation morphosyntaxique, il faut modifier un script disponible sur le site et aussi en annexes dans la section [1.4 Flair](#).

Dans ce cas, ce qui nous intéresse le plus est le bloc d'initialisation de corpus ainsi que le bloc d'initialisation de plongements lexicaux (*embeddings*). Le code est alors plus technique que ce que nous avons pu voir avec spaCy, mais il y a plus d'options pour modifier le traitement.

11.2.1 spaCy

spaCy, comme nous l'avons vu précédemment, est utilisable en ligne de commande. Cette fonctionnalité permet une prise en main rapide nous permettant d'effectuer plus rapidement nos expériences. Pour utiliser spaCy en ligne de commande, il suffit de savoir naviguer dans

les dossiers sur l'ordinateur à l'aide de la commande *cd*. Pour le reste, il est possible de copier et coller les commandes en modifiant les commandes à l'aide des tutoriels mis en place sur le site officiel.

En ce qui concerne les paramètres, spaCy les gère non à l'aide d'un script Python mais à l'aide d'un fichier de configuration. Ce fichier ressemble un peu à du langage structuré mais il s'agit d'une approche sans code. Cette approche mise en place par spaCy permet aux utilisateurs non experts en traitement automatique de langues de pouvoir effectuer des expériences.

11.2.3 Comparaison

Alors que les deux approches sont bien documentées et n'ont pas trop de barrières d'entrée, il y a clairement une librairie qui permet d'avoir des résultats plus rapidement et facilement que l'autre. spaCy est la librairie qui est la plus accessible et qui permet une prise en main rapide, quel que soit le niveau en programmation de l'utilisateur.

Inversement, la librairie la plus facile à utiliser pour une personne sachant déjà manipuler du code Python serait Flair. Flair permet de regrouper l'intégralité du code nécessaire dans un seul fichier contrairement à spaCy qui nécessite un fichier de configuration et plusieurs commandes à exécuter.

En ce qui concerne l'utilisation des plongements lexicaux, l'avantage se trouve clairement au sein de spaCy. Avec une commande, les plongements lexicaux sont formatés en format .spaCy, ce qui permet d'utiliser des plongements lexicaux de plusieurs types. Dans Flair, il est préférable d'utiliser les plongements de type Flair, car le processus d'initialisation de plongements lexicaux n'est pas aussi clair ou simple.

Le choix de librairie dépend des besoins et des compétences de l'utilisateur. Les deux sont rapides et très performantes, mais il faudra prendre en compte les besoins du projet. Par contre, si des plongements lexicaux créés par l'utilisateur sont utilisés, il est préférable de se servir de spaCy. SpaCy a mis en place un processus de conversion de plongements lexicaux beaucoup plus simple que celui de Flair. Il est possible de convertir des plongements lexicaux avec une seule commande avec leur interface en ligne de commande. Cependant, après avoir testé la performance des deux, il est aussi préférable de faire des tests sur les deux afin de voir laquelle est la plus adaptée au projet.

11.3 Annotations de spaCy et Flair

Pendant l'entraînement de nos deux modèles les plus performants, nous avons pu constater une exactitude très proche des deux modèles. Il est donc intéressant de montrer des annotations faites par nos modèles afin de voir s'ils attribuent les mêmes catégories aux mêmes tokens ou si les modèles étiquettent différemment les phrases. Afin d'assurer une bonne analyse, les mêmes phrases seront annotées. Ces phrases se trouvent dans le corpus d'évaluation et permet d'avoir des phrases correctement annotées afin de comparer les deux

outils. Cette approche permettra de voir les erreurs de nos deux modèles les plus performants. Les phrases utilisées dans cette analyse sont :

1. Fighjulu, ed eccu, ùn ci hè omu ; È tutti l'acelli di u celu sò fughjiti.
2. Tuttu què cù e territoriali di u 2015 chianu da vene subbitu dopu.

11.3.1 Annotations

Dans les tableaux, les erreurs sont soulignées en rouge.

Token	Vraie catégorie	Prédiction spaCy	Prédiction Flair
Fighjulu	NOUN	VERB	NOUN
,	PUNCT	PUNCT	PUNCT
ed	CCONJ	PRON	DET
eccu	ADV	ADV	NOUN
,	PUNCT	PUNCT	PUNCT
ùn	ADV	ADV	ADV
ci	PRON	PRON	PRON
hè	VERB	VERB	VERB
omu	NOUN	NOUN	VERB
;	PUNCT	PUNCT	PUNCT
È	CCONJ	CCONJ	CCONJ
tutti	DET	ADV	VERB
l'	DET	DET	DET
acelli	NOUN	NOUN	NOUN
di	ADP	ADP	ADP
u	DET	DET	DET
celu	NOUN	NOUN	NOUN
sò	AUX	AUX	AUX
fughjiti	VERB	VERB	NOUN
.	PUNCT	PUNCT	PUNCT

Tableau 11 - comparaison des annotations, phrase 1

Dans le cas de spaCy, 17/20 des annotations sont correctes. Dans le cas de Flair, 15/20 des annotations sont correctes.

Token	Catégorie	Prédiction spaCy	Prédiction Flair
Tuttu	PRON	PROPN	ADV
què	PRON	VERB	PRON
cù	ADP	ADP	ADP
e	DET	DET	DET
territoriali	NOUN	NOUN	NOUN
di	ADP	ADP	ADP
u	DET	DET	DET
2015	NUM	NUM	NUM
chianu	PRON	VERB	VERB
da	PART	ADP	DET
vene	VERB	ADJ	NOUN
subbitu	ADV	ADV	ADJ
dopu	ADV	ADJ	NOUN
.	PUNCT	PUNCT	PUNCT

Tableau 12 - comparaison des annotations, phrase 2

Dans le cas de spaCy, 8/14 des annotations sont correctes. Dans le cas de Flair, 8/14 des annotations sont correctes.

11.3.3 Comparaison

A commencer par le tableau 10, nous voyons que les erreurs des deux outils sont différentes, mais il y a deux tokens qui sont mal annotés par les deux modèles. Le premier token, ‘ed’, est une conjonction. spaCy a attribué la catégorie de pronom alors que Flair a attribué la catégorie de déterminant. Le second token, ‘tutti’, est un déterminant. spaCy a attribué la catégorie adverbe alors que Flair a attribué la catégorie verbe. Pour les autres erreurs, nous constatons que les erreurs faites par chaque modèle sont différentes.

Ensuite, dans le tableau 11, nous voyons que les deux modèles ont fait six erreurs chacun. Nous voyons aussi que 5/6 des erreurs sont sur les mêmes tokens. Une des raisons possibles de ces erreurs concerne le corpus lui-même. Tel que discuté dans la section [5.3 Corpus](#)

[annoté](#), nos corpus d'entraînement et de validation ne sont pas parfaits et peuvent contenir des erreurs. Cela est le cas du token *chianu*, qui aurait dû être séparé en deux tokens séparés.

Ces résultats montrent que les deux modèles sont assez similaires dans leur fonctionnement. Dans la plupart des cas, les erreurs sont sur les mêmes tokens. Il y a donc de la variation dans les erreurs, mais les principales restent les mêmes. Les modèles ont donc généralisé autant que possible sur le peu de données qui se trouvent dans les corpus d'entraînement et de validation.

11.4 Suite du projet

A l'issue de plus de recherches sur le sujet il sera possible de mettre en place un outil d'annotation morphosyntaxique automatique de qualité. Malheureusement, à ce stade nous n'avons pas un outil qui obtient une performance assez conséquente, mais nous ne sommes pas loin. Afin d'élargir les capacités de notre modèle, il faut un corpus annoté de grande taille. Nous avons atteint les limites de notre corpus de cent phrases, mais avec une approche semi-automatique il serait possible d'élaborer un plus grand corpus.

Sachant que le modèle est capable d'annoter correctement trois tokens sur quatre, il serait possible, à terme, de :

- Faire annoter un plus long texte par le modèle spaCy/Flair
- Faire corriger les annotations par des locuteurs corses
- Réentraîner le modèle et évaluer sa performance

Avec une approche itérative comme cela, le modèle sera entraîné sur davantage de données lui permettant de gagner en performance.

En ce qui concerne la méthodologie utilisée, il serait intéressant d'exploiter des outils permettant une validation croisée sur notre petit jeu de données afin d'obtenir une évaluation plus robuste des performances. Cela permet, à chaque itération, d'utiliser une partie différente du corpus d'entraînement pour la validation. Malheureusement, cela n'a pas été possible à faire avec les fonctionnalités de base de spaCy et Flair.

Références bibliographiques

- Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., & Vollgraf, R. (2019). FLAIR: An easy-to-use framework for state-of-the-art NLP. *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, 54-59.
- Akbik, A., Blythe, D., & Vollgraf, R. (2018). Contextual String Embeddings for Sequence Labeling. *Proceedings of the 27th International Conference on Computational Linguistics*, 1638-1649. <https://aclanthology.org/C18-1139>
- Atlas des langues en danger dans le monde—UNESCO Bibliothèque Numérique*. (s. d.). Consulté 11 mai 2022, à l'adresse <https://unesdoc.unesco.org/ark:/48223/pf0000189451>
- Bernhard, D., & Ligozat, A.-L. (2013). Es esch fäscht wie Ditsch, oder net? Étiquetage morphosyntaxique de l'alsacien en passant par l'allemand. *TALARE 2013*, 209-220. <https://hal.archives-ouvertes.fr/hal-00838355>
- Bernhard, D., Ligozat, A.-L., MARTIN, F., Bras, M., Magistry, P., Vergez-Couret, M., Steible, L., Erhart, P., Hathout, N., Huck, D., Rey, C., Reynés, P., Rosset, S., Sibille, J., & Lavergne, T. (2018, mai). Corpora with Part-of-Speech Annotations for Three Regional Languages of France : Alsatian, Occitan and Picard. *11th edition of the Language Resources and Evaluation Conference*. <https://hal.archives-ouvertes.fr/hal-01704806>
- Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python : Analyzing text with the natural language toolkit*. O'Reilly Media, Inc.
- Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*.
- Boyd, A., & Warmerdam, V. (2022, août 23). *floret : Lightweight, robust word vectors* ·

- Explosion*. Explosion. <https://explosion.ai/blog/floret-vectors>
- Dalbera-Stefanaggi, M.-J. (1978). *Langue corse : Une approche linguistique*. Éditions Klincksieck.
- de Marneffe, M.-C., Manning, C. D., Nivre, J., & Zeman, D. (2021). Universal Dependencies. *Computational Linguistics*, 47(2), 255-308.
https://doi.org/10.1162/coli_a_00402
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv.
<https://doi.org/10.48550/arXiv.1810.04805>
- Collectivité de Corse. (s. d.). *Inchiesta sociolinguistica nant'à a lingua corsa*. Direzione lingua corsa. Consulté 11 mai 2022, à l'adresse
https://www.isula.corsica/linguacorsa/Inchiesta-sociolinguistica-nant-a-a-lingua-corsa_a123.html
- Ghjambattista. (s. d.). *Grammaire corse : Accueil*. Consulté 30 décembre 2021, à l'adresse
<https://gbatti-alinguacorsa.pagesperso-orange.fr/grammaire/grammaire.htm>
- Heiden, S., Magué, J.-P., & Pincemin, B. (2010). *TXM : Une plateforme logicielle open-source pour la textométrie - conception et développement*. 2(3), 1021.
<https://halshs.archives-ouvertes.fr/halshs-00549779>
- Honnibal, M., & Montani, I. (2017). *spaCy 2 : Natural language understanding with {B}loom embeddings, convolutional neural networks and incremental parsing*. <https://spacy.io/>
- Horan, C. (2020, janvier 28). *Tokenizers : How machines read*. FloydHub Blog.
<https://blog.floydhub.com/tokenization-nlp/>
- Hulden, M., Alegria, I., Etxeberria, I., & Maritxalar, M. (2011). Learning word-level dialectal variation as phonological replacement rules using a limited parallel corpus.
Proceedings of the First Workshop on Algorithms and Resources for Modelling of

- Dialects and Language Varieties*, 39-48. <https://aclanthology.org/W11-2605>
- Kevers, L. (2022). *CCdC - Le Corpus Canopé de Corse*. UMR 6240 CNRS LISA - Université de Corse. <https://hal.science/hal-03912288>
- Kevers, L., Guéniot, F., Ghjacumina Tognotti, A., & Retali-Medori, S. (2019). Outiller une langue peu dotée grâce au TALN : L'exemple du corse et de la BDLC (Tooling up a less-resourced language with NLP : the example of Corsican and BDLC). *Actes de la Conférence sur le Traitement Automatique des Langues Naturelles (TALN) PFIA 2019. Volume II : Articles courts*, 371-380. <https://aclanthology.org/2019.jeptalnrecital-court.23>
- Kevers, L., MacLean, C. (2023). Corpus Canopé de Corse (CCdC) [Corpus]. ORTOLANG (Open Resources and TOols for LANGuage) - www.ortolang.fr, v1, <https://hdl.handle.net/11403/corpus-canope-de-corse/v1>.
- Kevers, L., Retali Medori, S., & Tognotti, A. G. (2021). *A Survey of Language Technologies Resources and Tools for Corsican* [Research Report]. UMR 6240 CNRS LISA - Université de Corse. <https://hal.archives-ouvertes.fr/hal-03228733>
- Kevers, L., & Retali-Medori, S. (2020). Towards a Corsican Basic Language Resource Kit. *Proceedings of the 12th Language Resources and Evaluation Conference*, 2726-2735. <https://aclanthology.org/2020.lrec-1.332>
- Krauwer, S. (2003). *The Basic Language Resource Kit (BLARK) as the First Milestone for the Language Resources Roadmap*.
- langues-cultures-de-france. (s. d.). *La langue corse | Réseau Langues et cultures de France*. Consulté 31 mars 2022, à l'adresse <http://www.langues-cultures-france.org/la-langue-corse/>
- Lay, M.-H., & Pincemin, B. (s. d.). *Pour une exploration humaniste des textes : AnaLog*. 12.
- Leclerc, J. (s. d.). *L'île de Corse (France)*. Consulté 31 mars 2022, à l'adresse

- <https://www.axl.cefai.ulaval.ca/europe/corsefra.htm>
- Leixa, J., Mapelli, V., & Choukri, K. (2014). *INVENTAIRE DES RESSOURCES LINGUISTIQUES DES LANGUES DE FRANCE*.
- Martinez, F. S. (s. d.). *Empleo de métodos no supervisados basados en corpus para construir traductores automáticos basados en reglas*. 164.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space* (arXiv:1301.3781). arXiv.
<https://doi.org/10.48550/arXiv.1301.3781>
- Millour, A. (2022). Corpus annoté de cent phrases en corse [Corpus non publié].
- Millour, A., & Fort, K. (2020). Text Corpora and the Challenge of Newly Written Languages. *Proceedings of the 1st Joint Workshop on Spoken Language Technologies for Under-Resourced Languages (SLTU) and Collaboration and Computing for Under-Resourced Languages (CCURL)*, 111-120.
<https://aclanthology.org/2020.sltu-1.15>
- Mosquera, A., Lloret, E., & Moreda, P. (2012). *Towards Facilitating the Accessibility of Web 2.0 Texts through Text Normalisation*. 9-14.
- Nivre, J., de Marneffe, M.-C., Ginter, F., Hajič, J., Manning, C. D., Pyysalo, S., Schuster, S., Tyers, F., & Zeman, D. (2020). Universal Dependencies v2 : An Evergrowing Multilingual Treebank Collection. *Proceedings of the Twelfth Language Resources and Evaluation Conference*, 4034-4043. <https://aclanthology.org/2020.lrec-1.497>
- Pointal, L. (2019). *TreeTagger Python Wrapper's documentation ! —TreeTagger Python Wrapper 2.3 documentation*. <https://treetaggerwrapper.readthedocs.io/en/latest/>
- Qi, P., Zhang, Y., Zhang, Y., Bolton, J., & Manning, C. D. (2020). Stanza : A Python Natural Language Processing Toolkit for Many Human Languages. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System*

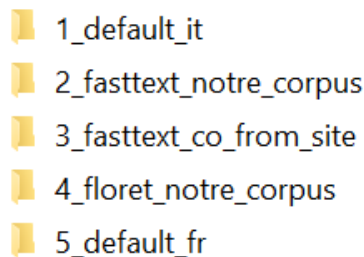
- Demonstrations*. <https://nlp.stanford.edu/pubs/qi2020stanza.pdf>
- Retali Medori, S., & Kevers, L. (2022). La morphologie dans la Banque de Données Langue Corse : Bilan et perspectives. *Corpus*, 23. <https://doi.org/10.4000/corpus.7115>
- Schmid, H. (1994). *TreeTagger*. <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/>
- Schweter, S. (2020). *Contextualized String Embeddings for PoS Tagging : A Multilingual Evaluation* [Python]. <https://github.com/stefan-it/flair-pos-tagging> (Original work published 2019)
- Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - NAACL '03, 1*, 173-180.
<https://doi.org/10.3115/1073445.1073478>
- Urieli, A. (s. d.). *Home · joliciel-informatique/talismane Wiki*. GitHub. Consulté 26 mars 2022, à l'adresse <https://github.com/joliciel-informatique/talismane>
- Urieli, A. (2013). *Robust French syntax analysis : Reconciling statistical methods and linguistic knowledge in the Talisman toolkit*. Université de Toulouse II-Le Mirail.
- Vergez-Couret, M., & Urieli, A. (2014). Pos-tagging different varieties of Occitan with single-dialect resources. *Proceedings of the First Workshop on Applying NLP Tools to Similar Languages, Varieties and Dialects*, 21-29.
<https://doi.org/10.3115/v1/W14-5303>
- Wikipédia:Lumière sur/Musée Solomon R. Guggenheim. (2022). In *Wikipédia*.
https://fr.wikipedia.org/w/index.php?title=Wikip%C3%A9dia:Lumi%C3%A8re_sur/Mus%C3%A9e_Solomon_R._Guggenheim&oldid=191820390

1. Annexes

1.1 SpaCy

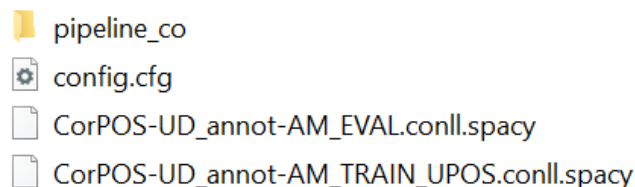
1.1.1 Organisation des dossiers

Il est primordial de bien s'organiser avant d'entreprendre des expériences avec plusieurs modèles de langue différents. J'ai donc décidé de numéroté chaque expérience, ce qui facilite l'organisation et l'exécution des entraînements.



A1 - organisation des dossiers

À l'intérieur de chaque dossier se trouvent les mêmes fichiers, ce qui permet d'exécuter les commandes d'entraînement de spaCy sans avoir à changer les chemins relatifs à l'intérieur du fichier de configuration.



A2 - organisation de l'intérieur des dossiers

Le dossier contient donc quatre choses :

1. Les plongements lexicaux contenus dans le dossier 'pipeline_co'
2. Le fichier de configuration
3. Les données d'évaluation afin de calculer la précision du modèle
4. Les données d'entraînement

À noter, les dossiers default_it et default_fr n'ont pas de dossier pipeline_co car ce dossier contient les plongements lexicaux pour chaque expérience.

1.1.2 Configuration

Afin de commencer à entraîner un modèle spaCy, nous avons deux options : mettre du code spaCy dans un fichier Python ou utiliser l'interface disponible en ligne de commande. J'ai décidé de poursuivre avec la deuxième option.

L'utilisation de spaCy en ligne de commande est par design, assez simple. L'interface permet d'effectuer toutes les étapes du traitement alors que normalement il faudrait écrire un script afin de gérer toutes les étapes nécessaires. La façon dont l'interface en ligne de commande gère toutes les commandes possibles est avec un fichier de configuration. Ce fichier, ci-dessous, contient plusieurs options qui facilitent la mise en place d'une procédure de traitement automatique pour une des tâches prises en charge par spaCy. Dans notre cas, nous allons utiliser la fonction de "tagger". Cette fonction correspond à l'argument "pipeline" dans le fichier de configuration et sa valeur par défaut est ["parser", "tagger", "ner"].

```

1  # This is an auto-generated partial config. To use it with 'spacy train'
2  # you can run spacy init fill-config to auto-fill all default settings:
3  # python -m spacy init fill-config ./base_config.cfg ./config.cfg
4  [paths]
5  train = null
6  dev = null
7  vectors = null
8  [system]
9  gpu_allocator = null
10
11  [nlp]
12  lang = "it"
13  pipeline = []
14  batch_size = 1000
15
16  [components]
17
18  [corpora]
19
20  [corpora.train]
21  @readers = "spacy.Corpus.v1"
22  path = ${paths.train}
23  max_length = 0
24
25  [corpora.dev]
26  @readers = "spacy.Corpus.v1"
27  path = ${paths.dev}
28  max_length = 0
29
30  [training]
31  dev_corpus = "corpora.dev"
32  train_corpus = "corpora.train"
33
34  [training.optimizer]
35  @optimizers = "Adam.v1"
36
37  [training.batcher]
38  @batchers = "spacy.batch_by_words.v1"
39  discard_oversize = false
40  tolerance = 0.2
41
42  [training.batcher.size]
43  @schedules = "compounding.v1"
44  start = 100
45  stop = 1000
46  compound = 1.001
47
48  [initialize]
49  vectors = ${paths.vectors}

```

A3 - fichier de configuration de spaCy

Ce fichier de base sera automatiquement rempli par la commande :

```
python -m spacy init config base_config.cfg --lang it --pipeline tagger
```

Normalement, les champs remplis par spaCy permettent d'exécuter l'entraînement, à condition que les chemins vers les plongements et les données d'entraînement et d'évaluation soient remplis. Si ce n'est pas le cas, il y a une autre commande très utile qui affiche les problèmes qui sont présents dans le fichier.

```
python -m spacy debug data config.cfg
```

Cette commande montre s'il y a des champs vides ou mal remplis et s'exécute en moins d'une seconde. Il est donc une étape nécessaire pour s'assurer que l'entraînement est prêt à commencer.

Une fois que le fichier de configuration est rempli, l'entraînement peut commencer. S'il est nécessaire de changer des paramètres, des chemins ou autre, tout se fait directement dans le fichier. Cette procédure permet une exécution plutôt facile à démarrer et facile à changer dans le cas où il faudra effectuer plusieurs entraînements, comme dans notre cas.

1.1.3 Initialisation des plongements lexicaux

Les plongements lexicaux de chaque dossier ont été créés de plusieurs façons différentes, et chacun des processus sera détaillé dans la section expliquant l'expérience. Les plongements lexicaux ont tous été convertis en un format accepté par spaCy, à l'aide d'une commande :

```
spacy init vectors it <PLONGEMENTS> <DOSSIER_SORTIE>
```

Cette commande a été exécutée sur chacun des fichiers contenant des plongements lexicaux et la sortie a été stockée dans le dossier relatif à l'expérience.

1.1.4 Création de plongements FastText et initialisation de l'entraînement

Tout d'abord, le module s'installe à l'aide des commandes suivantes :

```
$ git clone https://github.com/facebookresearch/fastText.git
$ cd fastText
$ make
```

A4 - installation de FastText

Ensuite, nous prenons notre corpus Canopé brut tokenisé en entrée et nous exécutons la commande suivante :

```
./fasttext skipgram -input data/fichier -output result/fichier
```

Cette commande permet de générer les plongements lexicaux sans plus de code ou d'autres démarches. Pour un corpus de cette taille, les plongements lexicaux sont créés en quelques secondes seulement. L'outil sort des plongements de type *skip gram*.

Ensuite, il faut convertir les plongements lexicaux en un format accepté par spaCy. Cela est également fait avec une seule commande :

```
python -m spacy init vectors it model.vec pipeline_co
```

Dans cette commande, *init vectors* est la commande, *it* la langue et *model.vec* le fichier contenant les plongements lexicaux. Le dossier *pipeline_co* est créé et les plongements en format spaCy y sont sauvegardés.

Ensuite, le chemin vers le dossier contenant les plongements est ajouté au chemin dans le fichier de configuration. Après que les chemins sont définis, il est toujours une bonne idée d'exécuter la commande :

```
python -m spacy debug config config.cfg
```

Celle-ci permet de vérifier la conformité du fichier et regarde s'il y a des erreurs de syntaxe ou de chemin.

Finalement, nous pouvons entraîner notre modèle sur les mêmes données mais avec nos plongements de type FastText.

```
python -m spacy train config.cfg
```

1.1.5 Utilisation des plongements lexicaux du site de FastText

Une fois les plongements récupérés, il faut commencer par exécuter la commande :

```
python -m spacy init vectors it model.vec pipeline_co
```

Ensuite, nous ajoutons les chemins vers les plongements et les jeux de données d'entraînement et d'évaluation. Il faut exécuter la commande :

```
python -m spacy debug config config.cfg
```

Une fois les éventuelles erreurs corrigées, nous pouvons exécuter la dernière commande :

```
python -m spacy train config.cfg
```

1.1.6 Représentation compacte des plongements FastText

Floret fait partie de FastText, mais ses représentations vectorielles sont différentes. L'installation de l'outil se fait de la même manière que FastText, mais à l'aide des commandes suivantes :

```
git clone https://github.com/explosion/floret
cd floret
make
```

A5 - installation de Floret

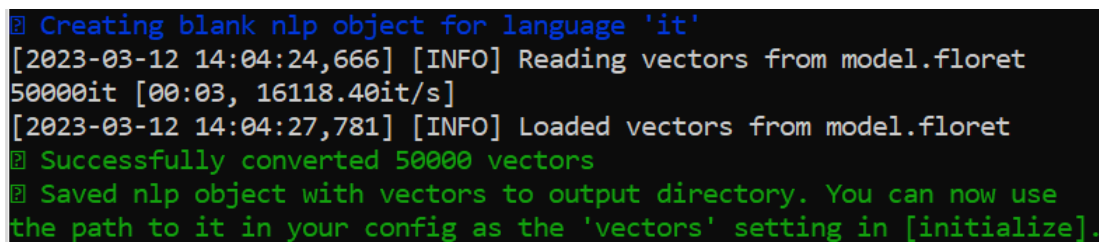
Pour nos plongements lexicaux, nous allons utiliser la commande recommandée par Floret qui créera des plongements lexicaux de type *Continuous Bag of Words* avec des sous mots de quatre et de cinq caractères. Cette commande recommandée va aussi générer une représentation compacte de 50 000 vecteurs.

La commande à exécuter est donc :

```
../floret/./floret cbow -dim 300 -minn 4 -maxn 5 -mode floret -hashCount 2 -bucket
50000 -input ../texte_complet.txt -output model
```

Ensuite, spaCy est déjà prêt à gérer des plongements de type Floret et la commande *init vectors* permet de spécifier que les plongements lexicaux sont de type Floret.

```
python -m spacy init vectors it model.floret pipeline_co --mode floret
```



```
❏ Creating blank nlp object for language 'it'
[2023-03-12 14:04:24,666] [INFO] Reading vectors from model.floret
50000it [00:03, 16118.40it/s]
[2023-03-12 14:04:27,781] [INFO] Loaded vectors from model.floret
❏ Successfully converted 50000 vectors
❏ Saved nlp object with vectors to output directory. You can now use
the path to it in your config as the 'vectors' setting in [initialize].
```

A6 - initialisation des plongements Floret dans spaCy

Ensuite, les chemins sont changés, le fichier est débogué et l'entraînement peut commencer à l'aide de la même commande.

```
python -m spacy train config.cfg
```

1.1.7

1.2 Utilisation des outils en TAL

1.2.1 TreeTagger

L'utilisation de `treetaggerwrapper` (Pointal, 2019) à l'aide de Python est très simple, mais nécessite une installation préalable de TreeTagger sur la machine. Il est également nécessaire d'avoir installé un interpréteur PERL. Pour créer un programme qui prend en entrée un fichier `.txt` en français et qui sort un fichier `.txt` annoté, il suffit d'écrire :

```
import sys
import treetaggerwrapper

def tag(entree, sortie):

    tagger = treetaggerwrapper.TreeTagger(TAGLANG='fr')
    tagger.tag_file_to(entree, sortie, encoding='utf8')

if __name__ == '__main__':
    infile = sys.argv[1]
    outfile = sys.argv[2]
    tag(infile, outfile)
```

A7 - utilisation de treetaggerwrapper

Il est également possible d'extraire un fichier `.csv` et de définir plus de paramètres pour une analyse plus détaillée. La sortie de cet outil sera un corpus contenant sur chaque ligne le token, la catégorie grammaticale et le lemme.

1.2.2 SpaCy

En tant que test, j'ai adapté le code proposé sur le site pour montrer comment il traite des textes en français.²²

²² Le texte utilisé dans l'entrée vient de (« Wikipédia », 2022):

```

import spacy

#Télécharger tokenizer, étiqueteur français
nlp = spacy.load("fr_core_news_sm")

# Documents à traiter
text = ("Il fut créé pour être un lieu d'exposition de "
        "l'art d'avant-garde d'artistes modernistes "
        "tels que Vassily Kandinsky et Piet Mondrian")
doc = nlp(text)

# Analyser la syntaxe
print("Groupes nominaux:", [chunk.text for chunk in doc.noun_chunks])
print("Verbes:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

# Trouver les entités nommées
for entity in doc.ents:
    print(entity.text, entity.label_)

```

A8 - utilisation de spaCy pour la reconnaissance des entités nommées

La sortie de ce code est le suivant :

```

Groupes nominaux: ['Il', "d'exposition de l'art d'avant-garde"]
Verbes: ['créer']
Vassily Kandinsky PER
Piet Mondrian PER

```

A9 - sortie du code spaCy REN

On peut voir dans ce petit exemple que le verbe “être” ne figure pas dans la liste de verbes. Il y a aussi une étiquette AUX pour les verbes auxiliaires. Aussi, on voit que la reconnaissance d’entités nommées a fonctionné parfaitement, les deux noms de personnes se trouvant affichés avec la bonne étiquette.

Un deuxième test, cette fois de son étiqueteur morphosyntaxique a été fait :

```
import spacy

nlp = spacy.load("fr_core_news_sm")
text = ("Il fut créé pour être un lieu d'exposition de "
        "l'art d'avant-garde d'artistes modernistes "
        "tels que Vassily Kandinsky et Piet Mondrian")
doc = nlp(text)
for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_)
```

A10 - utilisation de spaCy pour l'annotation morphosyntaxique

Il existe plusieurs traits qu'on pourrait analyser, mais les plus importants sont le mot (text), le lemme, la catégorie grammaticale (pos) et l'étiquette (tag). La sortie des cinq premiers mots est le suivant :

```
Il il PRON PRON__Gender=Masc|Number=Sing|Person=3
fut être AUX AUX__Mood=Ind|Number=Sing|Person=3|Tense=Past|VerbForm=Fin
créé créer VERB VERB__Gender=Masc|Number=Sing|Tense=Past|VerbForm=Part|Voice=Pass
pour pour ADP ADP
être être AUX AUX__VerbForm=Inf
```

A11 - sortie du code spaCy annotation morphosyntaxique

Il existe quelques précisions données par SpaCy, la voix par exemple, qui ne seront pas forcément nécessaires dans nos analyses du corse, mais on peut constater que l'analyse est complète.

En tout, l'utilisation est très simple et spaCy est très documentée. Il existe des tutoriels sur le site et des instructions à copier/coller pour commencer à travailler sans trop de délai pour apprendre le fonctionnement de l'outil.

1.2.3 Talismane

Talismane, un outil écrit entièrement en Java, est cependant exécutable en ligne de commande. Pour une analyse simple du français, sans modifications, l'utilisation n'est pas compliquée. Il suffit de télécharger trois fichiers, dont talismane-distribution, le fichier de configuration pour le français et le fichier frenchLanguagePack. Talismane-distribution doit être ensuite dézippé et les deux autres fichiers doivent être ensuite copiés dans le dossier où le talismane-distribution avait été dézippé.

Ensuite, Talismane peut être mis en marche à l'aide de la ligne de commande. Tout le processus est détaillé sur le GitHub du projet :

<https://github.com/joliciel-informatique/talismane/wiki>

(Urieli, s. d.)

Alors que le projet de Talismane est très développé pour l'analyse du français, de l'anglais et de l'occitan, l'outil n'est pas directement applicable au projet d'annotation morphosyntaxique du corse sans la création et l'entraînement d'un modèle pour la langue corse. Il serait préférable de travailler avec un logiciel ou une librairie qui a déjà un modèle créé pour une langue plus proche du corse, comme l'italien. Cependant, l'outil et son fonctionnement restent très intéressants dans l'analyse morphosyntaxique et pourraient être exploités à un moment ultérieur.

1.2.4 Stanford POS Tagger

Pour l'utilisation de Stanford POS Tagger, il faut d'abord avoir installé Java sur la machine. Ensuite, pour la majeure partie des opérations possible, il faut utiliser le logiciel en ligne de commande. Le logiciel s'exécute à l'aide de commandes telles que :

```
To run the tagger from the command line, you can start with the provided
script appropriate for you operating system:
./stanford-postagger.sh models/ws-j-0-18-left3words-distsim.tagger sample-input.txt
stanford-postagger models\ws-j-0-18-left3words-distsim.tagger sample-input.txt
The output should match what is found in sample-output.txt|
```

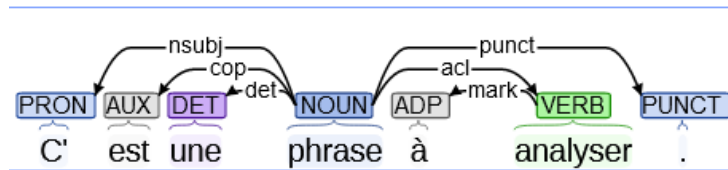
A12 - utilisation de Stanford POS Tagger

Il existe trois modes en Stanford POS :

1. Étiquetage
2. Entraînement de modèle
3. Test

Pour l'étiquetage il faut choisir le modèle souhaité, un modèle simple ainsi qu'un modèle bidirectionnel sont à disposition. Pour tout le processus, il suffit de lire le fichier README.txt présent après le téléchargement du logiciel. L'utilisation n'est pas compliquée et l'outil est très modifiable aux besoins du projet d'étiquetage morphosyntaxique. Il est facile de définir un autre jeu d'étiquettes et de définir des règles correspondant à la langue analysée.

Par contre, il est possible d'utiliser Stanford POS Tagger à l'aide d'une interface Python. La suite du projet est la création de Stanford CoreNLP, qui regroupe plusieurs projets du Stanford NLP Group. Ce projet, qui est très polyvalent peut s'intégrer aux projets écrits en Go, Java, Javascript, Perl, PHP, Python, R, Ruby, parmi d'autres. Il existe aussi une interface en ligne disponible à : <https://corenlp.run/>. La version en ligne est pourtant moins complète que les autres, mais elle est très utile pour des tests. Un exemple de la sortie du site :



A13 - Notre phrase analysée par <https://corenlp.run/>,

Dans le cadre du projet du corse, il sera préférable d'utiliser les bibliothèques Python qui permettent d'exploiter les modèles de Stanford. La première, qui est un outil d'étiquetage qui peut se servir des modèles de Stanford, est NLTK. Mais, elle n'est plus l'option la plus intéressante car le groupe Stanford NLP développe actuellement sa propre bibliothèque Python, appelée Stanza. Une discussion de la bibliothèque Stanza suit.

1.2.5 Stanza

Stanza, étant complètement intégrée dans Python, a une utilisation très simple et bien documentée. En tant de premier test, Stanza propose le code suivant :

```
import stanza
stanza.download('en')_# download English model
nlp = stanza.Pipeline('en')_# initialize English neural pipeline
doc = nlp("Barack Obama was born in Hawaii.")_# run annotation over a sentence

print(doc)
print(doc.entities)
```

A14 - utilisation de Stanza pour la reconnaissance des entités nommées

Il faut d'abord importer la bibliothèque, qui se fait de façon habituelle. Ensuite, il faut initialiser le chargement du modèle de langue souhaité, ici l'anglais en l'occurrence. Puis il faut initialiser le pipeline neuronal et ensuite faire annoter le texte.

En sortie, on trouve toute une liste de caractéristiques morphosyntaxiques qui ont été annotées. Ci-dessous on trouve un exemple de sortie pour le mot Barack dans la phrase initialisée dans le code :

```
"id": 1,
  "text": "Barack",
  "lemma": "Barack",
  "upos": "PROPN",
  "xpos": "NNP",
  "feats": "Number=Sing",
  "head": 4,
  "deprel": "nsubj:pass",
  "start_char": 0,
```

```
"end_char": 6,  
"ner": "B-PERSON"
```

Par contre il est possible de raccourcir les résultats avec une ligne de code comme le suivant :

```
for sentence in doc.sentences:  
    for word in sentence.words:  
        print(word.text, word.lemma, word.pos)
```

A15 - options à utiliser dans Stanza

Ce code rend la sortie bien plus lisible, toute la phrase prend moins de place que l'entrée du premier mot dans le dernier exemple :

```
Barack Barack PROP  
Obama Obama PROP  
was be AUX  
born bear VERB  
in in ADP  
Hawaii Hawaii PROP  
. . PUNCT
```

Tout cela est pour montrer qu'alors que Stanza traite énormément de caractéristiques linguistiques de chaque token, elle est très facilement adaptée aux besoins du projet. L'ajout du corse à cet outil permettrait aux chercheurs de continuer et d'approfondir des recherches sur la langue et permettrait à terme de créer encore plus de ressources numériques pour la langue.

1.2.6 Flair

L'utilisation de Flair est simple et bien documentée, il existe plusieurs tutoriels proposés par le site pour faire de l'étiquetage jusqu'à l'entraînement d'un nouveau modèle rapidement. Pour l'instant, il existe plus de support pour la reconnaissance d'entités nommées que pour l'étiquetage des parties du discours. Mais, une fonctionnalité intéressante est la possibilité d'étiqueter un texte multilingue. Cette option est expliquée en détail sur le GitHub du projet et contient un tutoriel qui montre la simplicité avec laquelle est mise en place.

```

from flair.data import Sentence
from flair.models import SequenceTagger

# télécharger le modèle
tagger = SequenceTagger.load('pos-multi')

# text en anglais et en allemand
sentence = Sentence('George Washington went to Washington. Dort kaufte er einen Hut.')

# attribuer les étiquettes
tagger.predict(sentence)

# affichage des résultats
print(sentence.to_tagged_string())

```

A16 - utilisation de Flair pour la reconnaissance des entités nommées

Les résultats de l'analyse sont :

George <PROPN> Washington <PROPN> went <VERB> to <ADP> Washington <PROPN>
 . <PUNCT> Dort <ADV> kaufte <VERB> er <PRON> einen <DET> Hut <NOUN> .
 <PUNCT>

Il serait préférable d'ajouter quelques lignes de code afin de rendre plus lisibles les résultats.

Mais, cet outil reste intéressant pour ces capacités d'analyse, sa flexibilité ainsi que la possibilité de créer et d'adapter un modèle pour une langue. L'outil sera toutefois utile, mais il ne sera pas la seule ressource numérique à utiliser dans le traitement du corse.

1.3 Tokeniseur

```

import re
import sys

def tokenize(input_file, output_file):
    """Séparez le texte en mots et en signes de ponctuation,
    garder les apostrophes au sein des mots.

    Args:
        input_file (txt): texte à tokeniser
        output_file (txt): texte tokenisé
    """
    with open(input_file, 'r', encoding="utf-8") as file:
        text = file.read()
        text = text.strip()

```

```

# regex pour tous les contextes et apostrophes possibles
tokens = re.findall(r"\b\w+(?:['\"`'`'_,-]*\w+)*\b|[\^\w\s']|\n",
text)

with open(output_file, 'w', encoding="utf-8") as file:
    for token in tokens:
        if token == "\n":
            continue
        else:
            file.write(token + '\n')

#exécutable en ligne de commande
if __name__ == "__main__":
    file1 = sys.argv[1]
    file2 = sys.argv[2]
    tokenize(file1, file2)

```

1.4 Processus de conversion de PDF en TXT

Cette section a été inspirée par un document détaillant le processus de traitement et de conversion de documents créé dans le cadre d'un stage à l'Université de Corse en été 2022. Le processus de traitement a été défini dans le cadre du projet BDLC et mis en œuvre durant le stage. Il est abordé par Kevers, (2022).

1.4.1 Outils nécessaires

Les outils que j'ai utilisés dans le traitement de ces documents étaient les suivants :

- Terminal Linux
- pdftotext : <http://www.xpdfreader.com/index.html>
- pdftk : normalement inclus dans une distribution Linux : <https://www.pdfabs.com/tools/pdftk-the-pdf-toolkit/>
- Éditeur de text : Atom/VScode

Pour développer un peu, j'utilise un ordinateur avec Windows 10. Pour changer le système d'exploitation et pouvoir exploiter les outils intégrés dans Linux, j'ai trouvé une solution.

*Windows Subsystem for Linux*²³ est un outil permettant d'utiliser une distribution Ubuntu en ligne de commande. Notamment, il est possible d'utiliser la ligne de commande Windows, mais aussi Linux, ce qui permet d'avoir accès aux outils puissants de Linux. Il existe d'autres solutions, plus ou moins adaptées comme l'utilisation de *Virtual Box*²⁴, qui permet de lancer un ordinateur virtuel Linux.

Pour le choix d'éditeur de texte, il s'agit de l'éditeur avec lequel on est le plus confortable, dans mon cas VSCode²⁵, un éditeur de texte et de code développé par Microsoft et disponible sous Windows, Mac et Linux.

1.4.2 Structure du répertoire

Pour commencer à traiter un document, il faut organiser les fichiers nécessaires au traitement. La structure suivante a été générée par la commande *tree*.

```
├── chapters
│   ├── pdf
│   ├── txt
│   ├── txt_modifs_manuelles
│   └── xml
├── fichier2TEI.py
├── pdftotext.sh
├── reflow.sh
├── texte_original.pdf
└── xmlreflow.py
```

Chaque livre, même ceux qui n'avaient qu'un chapitre, avait cette même structure. La structure est la même car elle facilite le traitement, nos scripts peuvent s'exécuter s'ils se trouvent dans le bon dossier. L'avantage est qu'il n'est pas nécessaire de changer les chemins relatifs à l'intérieur des scripts. Une description plus détaillée des fichiers Python et des fonctions des dossiers suit.

1.4.3 Étapes de traitement

Les étapes, globalement, étaient les suivantes :

- Découper chaque document pdf en chapitres
 - Regarder et noter les numéros de début et de fin de chapitre
 - Créer un script pour faire appel au module pdftk et découper le pdf en un document par chapitre
- Convertir chaque document (chapitre) en format xml avec des informations d'où les mots se trouvaient sur la page

²³ <https://learn.microsoft.com/fr-fr/windows/wsl/install>

²⁴ <https://www.virtualbox.org/>

²⁵ <https://code.visualstudio.com/docs>

- Créer un script pour faire appel au module pdftotext et découper en un document par chapitre, à priori ces commandes peuvent être ajoutées au script précédent
- Vérifier la qualité des documents
- Éditer un script écrit par Laurent Kevers pour séparer les documents en paragraphes et sortir des fichiers txt et xml

Dans la pratique, ce processus est plus complexe et sera abordé en détail ici. Pour commencer il fallait tout simplement regarder le pdf pour en avoir une idée du niveau de difficulté de traitement. Ensuite, pour s'organiser, il fallait mettre à jour un fichier intitulé `metadata_files.csv` où il y avait des informations sur la progression du traitement de chaque document. Cet état des lieux permet de prioriser le traitement des différents ouvrages en fonction de leur intérêt, de leur taille et de la difficulté de traitement anticipée.

Ensuite, il fallait exécuter le script Bash `pdftotext.sh`. Ce script avait une double fonction. Le découpage en chapitres est réalisé en faisant appel à `pdftk` pour chaque chapitre, la plage de pages à extraire, qui est passé en paramètre, ayant été repérée manuellement au préalable. Ensuite, la sortie de cet outil a été récupérée et traitée par l'outil `pdftotext`²⁶, qui convertit des fichiers de format pdf en fichiers de format xml qui gardent l'information d'où les mots se trouvaient sur la page. Les fichiers de sortie étaient stockés automatiquement dans les dossiers pdf et xml.

Un exemple des commandes faites par le script Bash se trouve ci-dessous :

```
1  #!/bin/bash
2
3  pdftk i-setti-mulini.pdf cat 8-11 output chapters/pdf/i-setti-mulini_Ch01.pdf
4
5
6  pdftotext -bbox-layout chapters/pdf/i-setti-mulini_Ch01.pdf chapters/xml/i-setti-mulini_Ch01.xml
```

A17 - commandes de découpage de PDF et de conversion en XML

L'information sur la position des mots sur la page est encodé en forme des variables XMin, XMax, YMin et YMax comme ci-dessous :

²⁶ <https://www.xpdfreader.com/pdftotext-man.html>

```

2  <head>
3  <title></title>
4  <meta name="Creator" content="pdftk-java 3.0.9"/>
5  <meta name="Producer" content="itext-paulo-155 (itextpdf.sf.net-lowagie.com)"/>
6  <meta name="CreationDate" content=""/>
7  </head>
8  <body>
9  <doc>
10 <page width="382.677000" height="538.583010">
11   <flow>
12     <block xMin="142.814900" yMin="36.000410" xMax="241.526900" yMax="85.464410">
13       <line xMin="151.598900" yMin="36.000410" xMax="232.760900" yMax="58.482410">
14         <word xMin="151.598900" yMin="36.000410" xMax="216.488900" yMax="58.482410">Capitulu</word>
15         <word xMin="221.780900" yMin="36.000410" xMax="232.760900" yMax="58.482410">4</word>
16       </line>
17       <line xMin="142.814900" yMin="62.982410" xMax="241.526900" yMax="85.464410">
18         <word xMin="142.814900" yMin="62.982410" xMax="241.526900" yMax="85.464410">L'arghjinteru</word>
19       </line>
20     </block>
21   </flow>

```

A18 - exemple des fichiers XML

La prochaine étape est l'utilisation du script `xmlreflow.py`. Ce script va effectuer plusieurs actions :

1. Ouvrir le(s) fichier(s) xml générés lors de la dernière étape
2. Repérer quelles parties il va falloir enlever du document final, à noter les notes de bas de page, les numéros de page ou autre partie ne faisant pas partie du texte principal
3. En regardant les coordonnées indiquées dans le xml, définir les variables `startX`, `endX`, `startY` et `endY` en fonction de ce que nous souhaitons garder. Ces variables vont servir à définir une fenêtre sur la page du texte que nous voulons garder. Donc, cette démarche permet de supprimer le haut et le bas de la page, qui ne contient normalement pas de texte intéressant.
4. Conserver ou commenter ces 3 conditions pour voir ce qui fonctionne mieux pour séparer les documents en paragraphes :

```

81     if currentX>startX and currentX<endX :
82         |     paragBefore=True
83         |     if lineXmax < (endX-5) :
84         |         paragAfter=True
85         |         |         |         |         |         |         |         |
86     elif printBlock == False:
87         |     paragBefore=True

```

A19 - conditions pour la séparation en paragraphes

À noter dans l'exemple ci-dessus :

Les lignes 86-87 ont été écrites pour séparer en paragraphes chaque page. Cela est le cas de la plupart des livres pour enfant d'avoir un paragraphe par page.

Aussi, pour le traitement de plusieurs documents (chapitres) il est préférable d'utiliser un script bash comme dans l'étape précédente. Ce processus permet de traiter l'ensemble des documents en même temps, ce qui réduit ensuite le temps passé sur la conversion des documents.

La sortie de `xmlreflow.py` est un fichier txt par chapitre avec un caractère de séparation de paragraphes. Les fichiers se trouvent dans le dossier `txt` et aussi dans le dossier `txt_modifs_manuelles`.

Ensuite il faut corriger les documents à la main. Après quelques vérifications et modifications du script, il reste souvent des petites erreurs dans la sortie des scripts, ce qu'il faut revoir et corriger manuellement. Chaque document avait des erreurs différentes et il n'était donc pas possible de cibler toutes les erreurs probables dans nos scripts. Celles-ci étaient ensuite documentées. Finalement, si nous avons fait des erreurs dans les corrections manuelles, il était facile de récupérer la version originale non corrigée car nous avons les deux dossiers, `txt` et `txt_modifs_manuelles`.

Le dernier traitement pour réussir à convertir les documents du pdf en xml TEI est l'exécution du script `2TEI.py`. L'objectif de ce script est de rassembler les différents chapitres, de les mettre au format xml TEI qui permet la numérotation des chapitres et paragraphes, ainsi que l'ajout de l'en-tête TEI qui contient les métadonnées. Ce script prendra en entrée le(s) fichier(s) txt modifié(s) manuellement et créera un seul fichier xml avec tout le document encodé en xml TEI et un seul fichier txt avec le texte de tous les chapitres.

Les étapes à suivre sont :

1. Ouvrir le fichier `2TEI.py` qui se trouve dans le dossier du document
2. Éditer les métadonnées, relire
3. Exécuter une première fois à l'aide de la commande :
 - a. `2TEI.py < DIR > < OUT > < NBdocs > < TITLE <NBtokens>`
 - b. `<NBtokens>` n'est pas disponible à ce stade et est remplacé par un symbole ou une valeur quelconque.
4. Récupérer le fichier `TEI.txt`
 - a. La première exécution du script a généré une version consolidée au format `TXT` qui permet un décompte du nombre de token à l'aide de la commande `wc`
5. Exécuter le script une deuxième fois, mais avec la bonne valeur dans le paramètre `<NBtokens>`

Cette double-exécution du script permet de trouver le nombre de tokens contenus dans le document final. Cette démarche était la plus simple.

1.5 Flair

```
from flair.datasets import UD_ENGLISH
from flair.embeddings import WordEmbeddings
from flair.models import SequenceTagger
from flair.trainers import ModelTrainer

# 1. load the corpus
corpus = UD_ENGLISH().downsample(0.1)
print(corpus)

# 2. what label do we want to predict?
label_type = 'upos'

# 3. make the label dictionary from the corpus
label_dict = corpus.make_label_dictionary(label_type=label_type)
print(label_dict)

# 4. initialize embeddings
embeddings = WordEmbeddings('glove')

# 5. initialize sequence tagger
model = SequenceTagger(hidden_size=256,
                        embeddings=embeddings,
                        tag_dictionary=label_dict,
                        tag_type=label_type)

# 6. initialize trainer
trainer = ModelTrainer(model, corpus)

# 7. start training
trainer.train('resources/taggers/example-upos',
             learning_rate=0.1,
             mini_batch_size=32,
             max_epochs=10)
```