CHAPITRE 1

INTRODUCTION

LES LANGUES

Le traitement du langage naturel est un domaine multidisciplinaire. Vu son importance dans nos jours, surtout avec l'augmentation de la quantité des informations textuels, il est primordial d'avoir une idée comment traiter automatiquement ces textes. Afin de le faire, il existe plusieurs niveaux : phonétique, phonologique, orthographe, morphologie, syntaxe, sémantique, pragmatique et discours. Chaque application de ce domaine nécessite un ou plusieurs niveaux de la langue. Bien sûr, comme tout domaine, ceci présente quelques défis. Ce chapitre est dédié à l'introduction de ce domaine en présentant les niveaux de traitement d'une langue et les applications de ce domaine, et en discutant quelques défis.

Le traitement automatique du langage naturel (TALN) est l'ensemble des méthodes permettant de rendre le langage humain accessible aux ordinateurs. Il est appelé, aussi, par le nom traitement automatique des langues (TAL). C'est un domaine multidisciplinaire qui implique : la linguistique (étude du langage), l'informatique (traitement automatique de l'information) et l'intelligence artificielle (ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine). De nos jours, ce domaine obtient plus d'intention suite à ces applications dans la vie quotidienne. Parmi les applications qui motivent ce domaine (elles seront reprises après), on peut citer :

- Augmenter la productivité en utilisant des applications comme la traduction automatique et le résumé automatique (pourtant ces deux applications sont loin d'être parfaites)
- Service Clientèle : la réponse automatique aux questions des clients en utilisant les chatbots (question-réponse et reconnaissance de voix).
- Surveillance de la réputation : on utilise l'analyse des sentiments pour savoir si les clients sont heureux avec ses produits ou non.
- La publicité : en scannant les réseaux sociaux et les courriels, on peut savoir qui est intéressé par ses produits. Ceci permet aux entreprises de viser l'audience de la publicité.
- Connaissance du marché (Market intelligence) : surveiller les compétiteurs afin de se tenir au courant des évènements liés à l'industrie.

1 Histoire

Le langage, d'après Larousse, est la "*capacité, observée chez tous les hommes, d'exprimer leur pensée et de communiquer au moyen d'un système de signes vocaux et éventuellement graphiques (la langue)*". Du même, la langue est définie par "*Système de signes vocaux, éventuellement graphiques, propre à une communauté d'individus, qui l'utilisent pour s'exprimer et communiquer entre eux : La langue française, anglaise.*". Donc, le langage désigne une capacité de communiquer, tandis que la langue représente l'outil de communication. Je ne veux pas présenter ici l'histoire du TALN du point de vu "langue", mais du point de vu "évolution de l'intelligence artificielle (IA)". Comme toutes les histoires, notre histoire commence par : il était une fois, un mathé-

maticien appelé “Alan Turing” qui a proposé une expérience pour tester qu’une machine soit “*consciente*”. Ce test est connu sous le nom de “Test de Turing”. Depuis cette proposition, le domaine de l’IA a reconnu des hauts et des bas. La figure 1.1 illustre quelques points dans cette histoire riche des évènements.

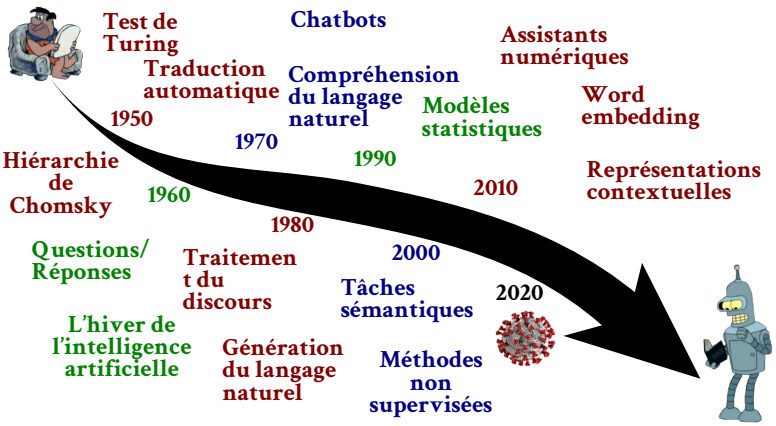


Figure 1.1 : *Quelques points historiques du TALN.*

1.1 Naissance de l’IA et âge d’or

Les années **195x** ont vu le départ du domaine de l’IA ainsi que le domaine du TALN. Parmi les premiers acteurs dans le domaine du TALN, on peut mentionner international business machines (IBM). Leurs recherches se sont portées sur la traduction et le résumé automatique du texte. Parmi les points importants de cette période, on peut citer :

— **1951** Shannon a exploité les modèles probabilistes des langages naturels (Shannon, 1951).
— **1954** Expérimentation Georgetown-IBM pour traduire automatiquement 60 phrases du russe vers l’anglais.
— **1956** Chomsky a développé les modèles formels de syntaxe.
— **1958** Luhn (IBM) a expérimenté sur le résumé automatique du texte par extraction (Luhn, 1958)

Le domaine a fleuri durant les années **196x**. Parmi les caractéristiques de cette ère, l’attention donnée à la conception des **chatbots**. Quelques points qui marquent cette décennie sont les suivants :

— **1961** Développement du premier analyseur syntaxique automatique à U. Penn. (Joshi, 1961 ; Harris, 1962)
— **1964** Weizenbaum a mis au point “ELIZA”, une simulation d’une psychothérapeute au sein du laboratoire MIT AI.
— **1964** Bobrow a mis au point “STUDENT”, conçu pour lire et résoudre des problèmes de mots trouvés dans les livres d’algèbre de lycée (Bobrow, 1964).
— **1967** Brown corpus, le premier corpus électronique.

1.2 Hiver de l’IA

Les années **197x** ont reconnu une croissance de développement des chatbots et les tâches connexes. Parmi ces dernières, on trouve : la compréhension de la langue et la reconnaissance de la parole. Il faut savoir qu’à partir de cette période, on a vu l’abondement de quelques concepts comme le connexionnisme (**1969**), le rapport “Sir James Lighthill”, les coupes budgétaires de Defense Advanced Research Projects Agency (DARPA), etc. C’était une époque sombre pour l’IA. Quelques points importants peuvent être résumés dans la liste suivante :

— **1971** Winograd (MIT) a développé "SHRDLU", un programme de compréhension du langage naturel (Winograd, 1971).
— **1972** Colby (Stanford) a créé "PARRY" un **chatbot** qui simule une personne avec la schizophrénie paranoïde.
— **1975** "MARGIE" un système qui fait des inférences et des paraphrases à partir des phrases en utilisant la représentation conceptuelle du langage.
— **1975** "DRAGON", un système pour la reconnaissance automatique de la parole en utilisant les modèles de Markov cachés (Baker, 1975).

Les années **198x** ont reconnu l'avancement dans les tâches syntaxiques et morpho-syntaxiques. Ils ont vu des recherches sur la représentation et l'extraction de la connaissance. Malgré l'hiver, il y avait encore des chercheurs qui croyaient en l'IA. Voici quelques points résumant ces années :
— **1980** "KL-One", représentation de connaissance pour le traitement de la syntaxe et la sémantique (Bobrow et Webber, 1980).
— **1986** "TRUMP", analyseur de langage en utilisant une base lexicale (Jacobs, 1986).
— **1987** HPSG (head-driven phrase structure grammar, traduction française : grammaire syntagmatique guidée par les têtes) (Sag et Pollard, 1987).
— **1987** "MUC" conférence sur l'extraction des données financée par "DARPA".
— **1988** Utilisation des models de markov cachés dans l'étiquetage morpho-syntaxique (Church, 1988).
— Solutions symboliques sur le traitement du discours et la génération du langage naturel.

1.3 Printemps de l'IA

Les années **199x** ont été les années de l'approche statistique. Plusieurs tâches ont été proposées en se basant sur cette approche comme la traduction automatique, les analyseurs syntaxiques, etc. Afin de supporter cette approche, on a vu aussi la création des corpus. Parmi les points importants de cette période, on peut citer :
— **1990** Une approche statistique pour la traduction automatique (Brown et al., 1990)
— **1993** Pen **TreeBank**, un corpus annoté de l'anglais (Marcus et al., 1993)
— **1995** **Wordnet**, une base lexicale pour l'anglais (Miller, 1995)
— **1996** "SPATTER", un analyseur lexical statistique basé sur les arbres de décision (Magerman, 1996)
— Popularité des méthodes statistiques et de l'évaluation empirique

Les années **200x** sont marquées par l'arrivé du niveau sémantique. En plus, des techniques comme les réseaux de neurones et l'apprentissage non supervisé ont commencé à prendre leurs places. Parmi les sujets traités durant cette période, on peut citer :
— **2003** Les modèles probabilistes de langues en utilisant les réseaux de neurones (Bengio et al., 2003)
— **2006** "Watson" (IBM), un système de question/réponse.
— Utilisation de l'apprentissage non supervisé et semi-supervisé comme alternatives à l'apprentissage purement supervisé.
— Déplacer le focus sur les tâches sémantiques.

Les années **201x+** ont connu l'intégration des solutions intelligentes dans des produits commerciales. Ceci est marqué principalement par le développement des assistants numériques personnels ; en anglais : intelligent personal assistant (IPA) ou intelligent virtual assistant (IVA). Aussi, l'intégration de la sémantique dans des tâches comme par exemple la recherche d'information. Parmi les points de cette ère, on peut mentionner :

— **2011** "Siri" (Apple) un assistant numérique personnel. Il a été suivi par "Alexa" (Amazon, 2014) et "Google Assistant" (2016)
— **2014** Word embedding (Lebret et Collobert, 2014)
— **2018** Apparition des représentations contextuelles (des modèles de langue pré-entraînés) : **ULMfit** (fast.ai) (Howard et Ruder, 2018), **ELMo** (AllenNLP) (Peters et al., 2018), **GPT** (OpenAI) (Radford et al., 2018), **BERT** (Google) (Devlin et al., 2018), **XLM** (Facebook) (Lample et Conneau, 2019)

2 Niveaux de traitement d'un langage

Reprenons la définition du langage par Larousse : "*capacité, observée chez tous les hommes, d'exprimer leur pensée et de communiquer au moyen d'un système de signes vocaux et éventuellement graphiques (la langue)*". Ce langage est doté d'une sémantique ainsi qu'une structure. La capacité d'un langage naturel (humain) comprend plusieurs fonctions linguistiques (niveaux) qui sont représentées dans la figure 1.2.

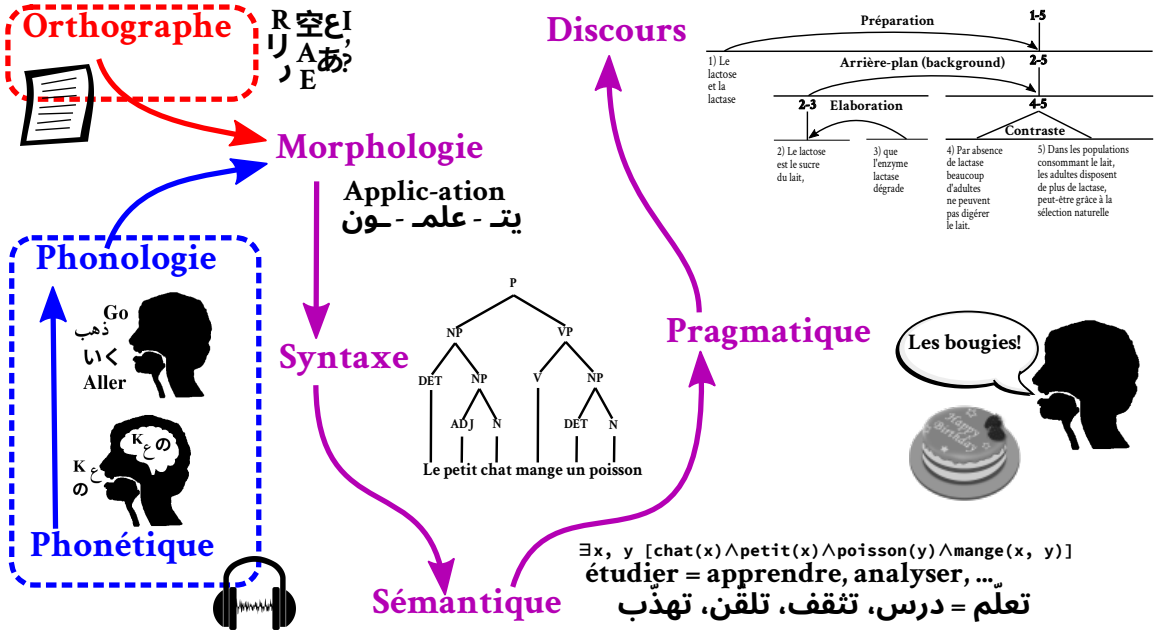


Figure 1.2 : *Niveaux de traitement d'un langage naturel*

2.1 Phonétique

La phonétique est l'étude des sons ou phones produits par l'appareil phonatoire humain. Elle n'est pas dépendante d'une langue précise. Il y a plusieurs branches dans la phonétique :

— **Phonétique articulatoire** : C'est la division la plus anatomique et physiologique. Elle décrit comment les voyelles et les consonnes sont produites ou "articulées" dans des diverses parties de la bouche et de la gorge.
— **Phonétique acoustique** : C'est la branche qui a les affinités les plus étroites avec la physique. Elle étudie les ondes sonores qui transmettent les voyelles et les consonnes dans l'air du locuteur à l'auditeur.
— **Phonétique auditive** : C'est la branche qui intéresse le plus les psychologues. Elle examine la manière dont le cerveau de l'auditeur décode les ondes sonores en voyelles et consonnes initialement prévues par le locuteur.

Dans la reconnaissance et la synthèse de la parole, en générale, on s'intéresse par la phonétique acoustique (comment les lettres sont représentées sous forme des ondes) et la phonétique articulaire (comment peut-on encoder les voyelles et les consonnes sur machine). Une façon pour encoder les voyelles et les consonnes est de les classifier selon les points d'articulation (voir la figure 1.3). Les consonnes sont divisées, selon la voie orale, en 5 groupes :

- **labial** à l'aide des lèvres. Ex., [b], [p], [m], [f], [v]
- **apicale** avec la pointe de la langue ou sa partie antérieure. Ex., [t], [d], [n], [r], ش [ʃ], ث [θ], ذ [ð]
- **dorsal** avec la partie postérieure de la langue. Ex., [c], [k], [g], [q], غ [ɣ]
- **pharyngale** au niveau du pharynx. Ex., ح [ħ], ع [ʕ]
- **glottale** au niveau de la glotte. Ex., [h], أ [ʔ]

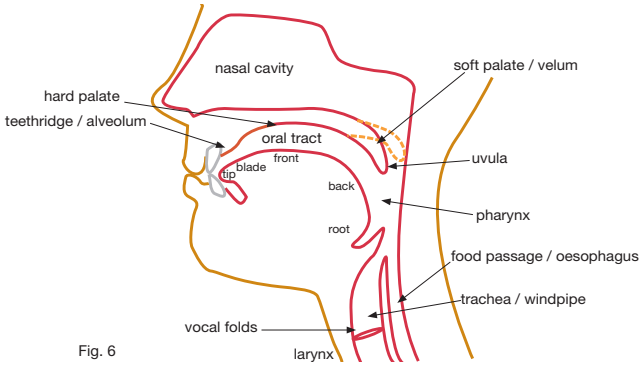


Figure 1.3 : *Voie orale (Ball, 2009).*

International Phonetic Alphabet (IPA) est un alphabet utilisé pour la transcription phonétique des sons du langage parlé. La parole est découpée en segments sonores distincts (phones). A chaque phone, il est attribué un symbole unique. La figure 1.4 représente une partie de l'alphabet phonétique internationale.

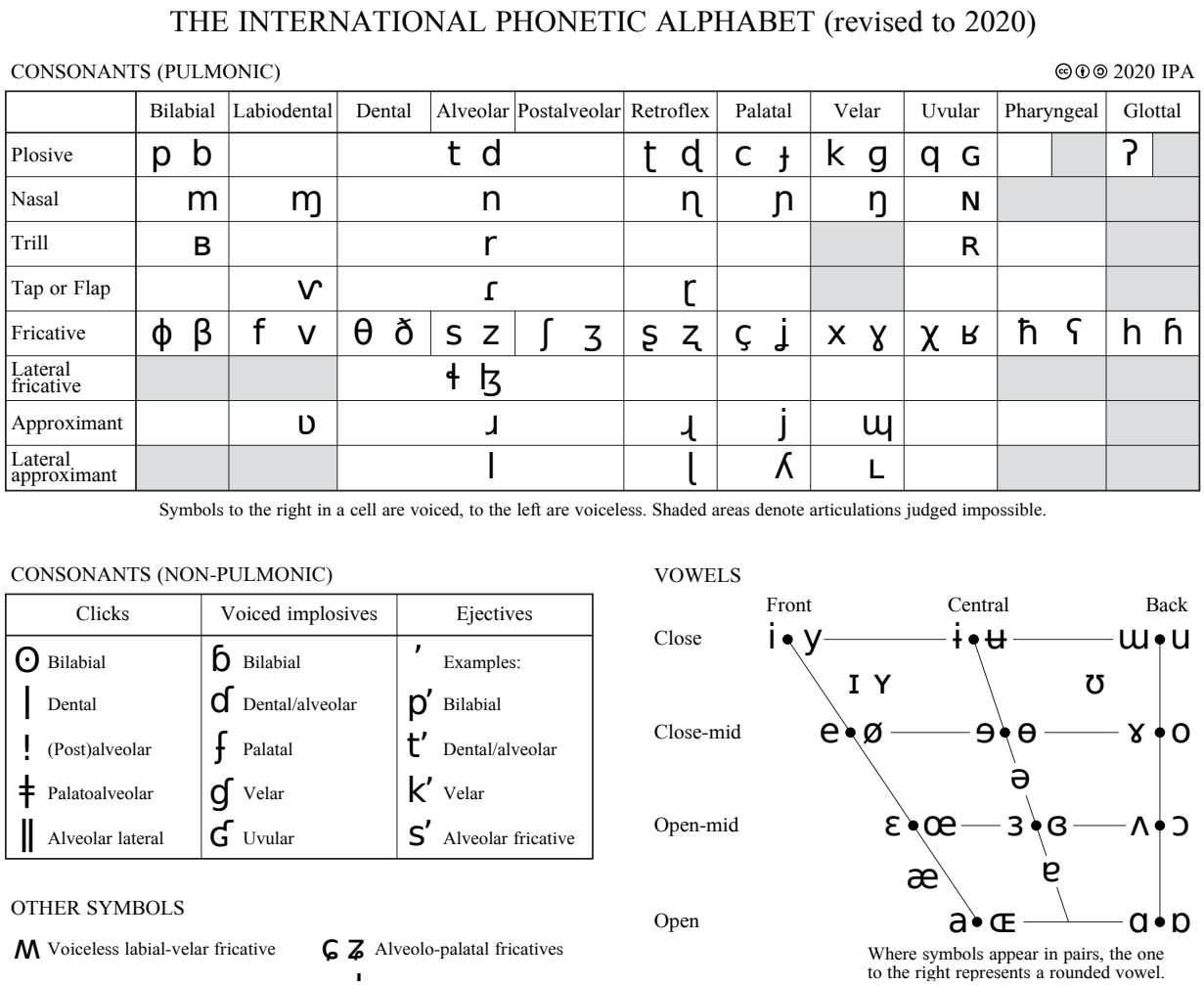
2.2 Phonologie

La phonologie est l'étude des sons ou phonèmes d'une langue donnée ; elle s'intéresse aux sons en tant qu'éléments d'un système. Contrairement au phone, un phonème est une unité abstraite qui peut correspondre à plusieurs sons. Il est lié à la langue étudiée. Par exemple, en français, le *r* peut se prononcer (en phonétique) : roulé [r], grasseyé [ʁ], ou normal (parisien) [ʁ]. Il est toujours transcrit de la même façon, exemple rat /rat/. En arabe, on trouve les consonnes ر [r] et غ [ɣ] qui ont deux phonèmes différents : /r/ et /ɣ/ respectivement. Exemple, غريب /ɣæri ːb/ (étranger). Donc, la phonologie s'intéresse à la transcription des sons par rapport à une langue donnée.

2.3 Orthographe

L'orthographe est l'ensemble des règles d'écriture d'une langue ; c'est l'étude des types et de la forme des lemmes/monèmes. Un lemme est une unité lexicale ; il désigne une unité autonome constituant le lexique d'une langue. L'unité significative la plus petite dans une langue est appelée "graphème". Les systèmes d'écriture sur lesquels l'orthographe est basé sont regroupés en 4 classes (selon les graphèmes) :

- **logographique** : un mot est constitué d'un à plusieurs logogrammes. Ce dernier est un graphème unique notant un lemme (mot). Exemple, Kanji (Japonais) : 日 , 本 , 語. Un logogramme peut être prononcé de différentes manières. Par exemple, 楽(tano)し(shi)い(i)(agréable), 音(on)楽(gaku)(musique)



THE INTERNATIONAL PHONETIC ALPHABET (revised to 2020)

CONSONANTS (PULMONIC) — © 2020 IPA

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap or Flap		ⱱ		ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative				ɬ ɮ							
Approximant		ʋ		ɹ		ɻ	j	ɰ			
Lateral approximant				l		ɭ	ʎ	ʟ			

Symbols to the right in a cell are voiced, to the left are voiceless. Shaded areas denote articulations judged impossible.

CONSONANTS (NON-PULMONIC)

Clicks	Voiced implosives	Ejectives
ʘ Bilabial	ɓ Bilabial	ʼ Examples:
ǀ Dental	ɗ Dental/alveolar	pʼ Bilabial
ǃ (Post)alveolar	ʄ Palatal	tʼ Dental/alveolar
ǂ Palatoalveolar	ɠ Velar	kʼ Velar
ǁ Alveolar lateral	ʛ Uvular	sʼ Alveolar fricative

OTHER SYMBOLS

ʍ Voiceless labial-velar fricative

ɕ ʑ Alveolo-palatal fricatives

ɺ

VOWELS

	Front	Central	Back
Close	i • y	ɨ • ʉ	ɯ • u
	ɪ ʏ		ʊ
Close-mid	e • ø	ɘ • ɵ	ɤ • o
		ə	
Open-mid	ɛ • œ	ɜ • ɞ	ʌ • ɔ
	æ	ɐ	
Open	a • ɶ		ɑ • ɒ

Where symbols appear in pairs, the one to the right represents a rounded vowel.

Figure 1.4 : *Alphabet phonétique internationale, https://www.internationalphoneticassociation.org/IPAcharts/IPA_chart_orig/pdfs/IPA_DejaVu_2020_full.pdf [visité le 2021-09-08]*

— **syllabique** : un mot est constitué de plusieurs symboles, chacun représente un syllabe (son vocalisé). Exemple, Hiragana (Japonais) : る /ru/, た /ta/, め /me/, の /no/ ; Katakana (Japonais) : セ /se/, ク /ku/ ;

— **alphabétique** : un mot est composé des lettres, chacune représente un phonème. Exemple, Le latin : A, B, C, etc. L'arabe : ب /b/, ت /t/, ه /h/. Il est à noter que l'arabe utilise un sous-système alphabétique appelé **abjad**. Dans ce sous système, il n'y a pas de voyelles ; il y a que des consonnes.

2.4 Morphologie

La morphologie est concernée par l'étude de la formation des mots, y compris la façon dont les nouveaux mots sont inventés dans les langues du monde. Elle étudie la façon dont les formes des mots varient en fonction de leurs utilisations dans les phrases. La plus petite unité d'une langue avec sa propre signification est appelée "morphème" (Ex., les noms propres, les suffixes, etc.). Les mots, dans plusieurs langues, sont formés par composition des morphèmes en suivant des règles. Toutes les formes grammaticales ayant le même sens forment un ensemble appelé "lexème" (Ex. [former, formation, formateur, forment, formez, ...]). Le mot choisi parmi ces formes pour représenter le lexème est appelé "lemme" (Ex. former). Chaque mot possède une catégorie grammaticale (nom, verbe, etc.) qui peut appartenir à :

— **la classe ouverte** contenant les catégories grammaticales qui changent leurs formes selon des traits

grammaticaux (par exemple, pluriel et singulier). Dans le français, cette classe contient : les adjectifs, les noms, les verbes, les déterminants et les pronoms.

— **la classe fermée** où les mots d'une catégorie grammaticale n'acceptent pas de changement. Dans le français, cette classe comporte : les adverbes, les articles, les conjonctions, les interjections et les prépositions.

J'ai déjà mentionné que les mots sont formés par composition de morphèmes dans plusieurs langues. Si vous avez remarqué : dans cette phase on peut dédier qu'il existe des langues où les mots ne sont pas formés autant. Selon la façon de former les mots, on peut classifier une langue selon la typologie morphologique suivante :

— **Langues isolantes/analytiques** : chaque mot est constitué d'un et d'un seul morphème. Les modifications morphologiques sont peu nombreuses, voire absentes. Parmi ces langues : mandarin, vietnamien, thaï, khmer, etc.. Exemple, 四个男孩 /sì ge nánhái/ "quatre garçons" (lit. "quatre [entité de] masculin enfant")

— **Les langues flexionnelles/synthétiques** : les mots sont formés d'une **racine** en plus de morphèmes supplémentaires.

  — **Langues agglutinantes** : les morphèmes sont toujours clairement différentiables phonétiquement l'un de l'autre. Parmi ces langues : finnois, turc, japonais, etc.. Exemple, 行く /iku/, 行きます /ikimasu/,

  — **Langues fusionnelles** : il n'est pas toujours aisé de distinguer les morphèmes de la racine, ou les morphèmes les uns des autres. Parmi ces langues : arabe, anglais, français, etc.. Exemple, كِتَاب، كُتُب، أَوَأَعطَينَاكُمُوه, foot, feet

Pour former des nouveaux mots, on utilise deux méthodes : la flexion et la dérivation. Dans la morphologie flexionnelle, les mots formés ne changent pas de catégories grammaticale (un verbe reste un verbe après formation) et ils ne créent pas de nouveaux lexèmes (le mot formé doit avoir le même sens). Donc, les mots sont formés juste pour s'adapter à différents contextes grammaticaux (pluriel vs singulier, masculin vs féminin, etc.). La flexion peut être une conjugaison (verbes) ou une déclinaison (le reste des catégories grammaticales ouvertes). Comme exemple de la déclinaison, on peut citer la transformation du singulier au pluriel pour les noms. Parmi les méthodes utilisées par la flexion, on peut citer :

— **Affixation** : C'est la formation d'un nouveau mot en utilisant les préfixes (avant le mot original), les infixes (au milieu) et les suffixes (après). Un autre type des infixes est la transfixe trouvée chez les langues sémitiques (un infixe discontinu). Un exemple d'une flexion (dans ce cas : conjugaison) par préfixes : ذهب /dhahaba/ (il est allé, passé), يذهب /yadhhabu/ (il va, présent). Conjugaison par préfixe et transfixe : قَال /qāla/ (il a dit, passé), يقول /yaqūlu/ (il dit, présent). Un autre pour une flexion (déclinaison) par suffixes : étudiant (masculin-singulier), étudiantes (féminin-pluriel).

— **Redoublement** : doubler un mot pour former un autre mot ayant la même catégorie et sens. Exemple, super-duper, bye-bye, きらきら /kira-kira/ (briller).

Une flexion est décrite par les traits grammaticaux. Chaque langue définit un ensemble des traits grammaticaux [1] à utiliser avec les catégories grammaticales. On peut trouver un trait grammatical dans une langue et pas dans une autre. Parmi les traits grammaticaux, on peut citer les suivants :

— **Nombre** : représente la quantité. singulier (il), duel (هما), triel, paucal, pluriel (ils), etc.

— **Personne** : décrit le rôle qu'occupent les acteurs d'un dialogue. première (je, nous), deuxième (tu, vous), troisième (il, ils, elle, elles), etc.

1. Vous pouvez savoir plus sur ces traits en suivant ce lien : https://universaldependencies.org/u/feat/index.html [visité le 2021-09-08]

- **Genre** : divise les noms en se basant sur le sexe. masculin (il, ils), féminin (elle, elles), neutre (it), commun (utilisé pour indiqué que c'est masculin/féminin contre neutre).
- **Cas** : représente le rôle sémantique par rapport au verbe. nominatif (sujet d'un verbe), accusatif (objet direct), datif (objet indirect), etc.
- **Temps** : représente le temps d'une action. passé, présent, future.
- **Aspect** : accompli (une action qui a été accomplie dans le temps)/inaccompli, progressif (une action qui est continue dans le temps), imperfectif (une action qui a pris une certaine période dans le temps mais on ne sait pas si elle est complète)/perfectif, itératif (une action qui se répète), prospectif (une action qui devrait avoir lieu à un moment qui suit un point de référence), etc.
- **Voix (diathèse)** : décrit comment s'organisent les rôles sémantiques par rapport au verbe. active (le sujet est l'agent ; celui qui a fait l'action), moyenne (le sujet est l'agent et le patient au même temps ; faire une action sur lui-même), passive (le sujet est le patient ; celui qui reçut l'action), réciproque (des agents et des patients qui font des actions l'un sur l'autre), etc.
- **Polarité** : considère un mot comme affirmatif ou négatif.
- **Politesse** : représente le niveau de respect. informelle, formelle, etc.

Contrairement à la morphologie flexionnelle, la morphologie dérivationnelle vise à former des mots en changeant de catégorie (jouer, joueur) ou en créant de nouveaux lexèmes (connecter, déconnecter). Parmi les méthodes utilisées par la dérivation, on peut citer :

- **Affixation** : en utilisant des préfixes, infixes et/ou suffixes. Exemple, happy (ADJ), unhappy (ADJ), unhapyness (N) ; جهد /jahada/ (V), اِجتهد /ijtahada/ (V)
- **Composition** : en fusionnant des mots dans un seul. Exemple, porter (V) + manteau (N) = porte-manteau (N) ; wind (N), mill (N), windmill (N)
- **Conversion** : en changeant la catégorie grammaticale d'un mot sans aucune modification. Exemple, orange (fruit, N), orange (couleur, ADJ) ; visit-er (V), visite (N) ; fish (N), to fish (V)
- **Troncation** : Exemple, bibliographie, biblio, information, info
- **Redoublement** : Exemple, كر (V), كركر (V)

2.5 Syntaxe

La syntaxe est l'ensemble des règles et principes qui contrôlent la structure d'une phrase. Cette structure est en relation avec les catégories grammaticales (Verb, Nom, Adjectif, etc) des mots composant la phrase. Selon la position du mot, il peut avoir une fonction grammaticale (Sujet, COD, COI, etc.). Les langues peuvent être classifiées selon l'ordre des trois composants : **Sujet (S)**, **Verb (V)** et **Objet (O)**. Exemple SOV [japonais] : カリムさん[S] は日本語[O] を勉強します[V]。. Exemple SVO [français] : Karim [S] apprend [V] le français [O].. Exemple VSO [arabe] :. [O] العربية [S] كريم [V] يتعلّم. Le tableau 1.1 représente une étude sur 402 langues qui nous donne une idée sur les proportions de chaque catégorie des ordres.

Il existe plusieurs approches théoriques de la syntaxe, notamment : la grammaire des constituants et la grammaire des dépendances. Dans une grammaire des constituants, une phrase est constituée de plusieurs **syntagmes** qui sont constitués des mots et d'autres **syntagmes**. Un syntagme contient un noyau qui est l'élément central et des éléments satellites. Selon le noyau, le **syntagme** peut être : nominal (NP), adjectival (AP), verbal (VP) ou prépositionnel (PP). Le système formel le plus utilisé pour modéliser la structure des constituants d'une phrase est la grammaire hors-contexte (Niveau 2 dans la hiérarchie de Chomsky). Un exemple d'une grammaire ainsi que l'arbre syntaxique d'une phrase est illustré dans la figure 1.5. La deuxième règle n'ai pas écrite (NP → DET NP) sinon on peut avoir plusieurs déterminants à un nom.

Ordre	Proportion	Exemples
SOV	44.78%	japonais, latin, tamoul, basque, ourdou, grec ancien, bengali, hindi, sanskrit, persan, coréen
SVO	41.79%	français, mandarin, russe, anglais, haoussa, italien, malais (langue), espagnol, thaï
VSO	9.20%	irlandais, arabe, hébreu biblique, philippin, langues touarègues, gallois
VOS	2.99%	malgache, baure, car (langue)
OVS	1.24 %	apalai, hixkaryana, klingon (langue)

Tableau 1.1 : *Ordres syntaxique et leurs proportions d'après l'étude de 402 langues (Blake, 1988)*

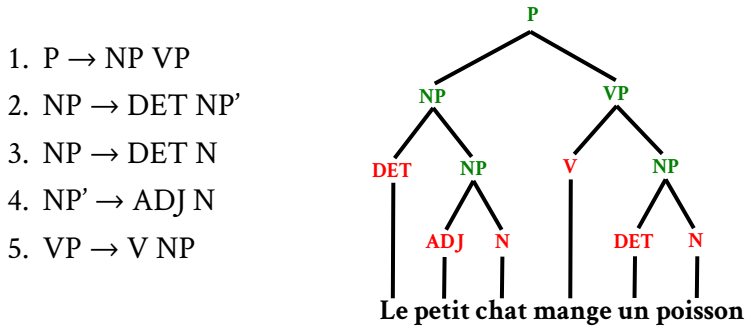


Figure 1.5 : *Exemple d'une grammaire hors-contexte et un arbre syntaxique de la phrase "Le petit chat mange un poisson"*

Dans une grammaire de dépendances, la structure syntaxique est décrite en terme des mots et pas des syntagmes. Les mots de la phrases sont liés entre eux par des relations binaires. Ces relations [2] peuvent être : un sujet nominal (**nsubj**), un objet (**obj**), un modificateur d'adjectif (**amod**), déterminant (**det**), etc. Un exemple d'une grammaire de dépendance est donné dans la figure 1.6.

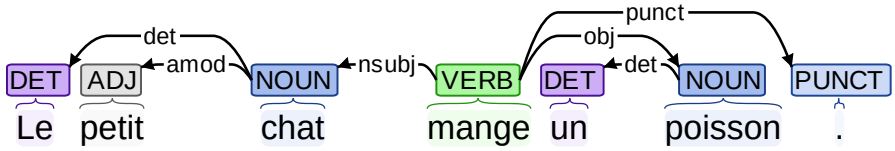


Figure 1.6 : *Exemple de dépendances, généré par https://corenlp.run/ [visité le 2021-09-08]*

2.6 Sémantique

La sémantique est l'étude du sens dans les langues. Il y a deux types de sémantique : la sémantique lexicale pour étudier le sens des mots et la sémantique propositionnelle qui concerne le sens des phrases. Plusieurs langues introduisent la notion de polysémie où un mot possède plusieurs sens. Par exemple, le terme "poulet" possède un sens différent selon la phrase où il est utilisé. Voici quelques exemples de l'utilisation de ce mot :

— J'écoute les piaulements des poulets. "*Petit du coq et de la poule, plus âgé que le poussin, avant d'être adulte.*"
— Je mange du poulet. "*Viande de jeune poule ou jeune coq.*"
— Mon petit poulet. "*Terme d'affection, que l'on adresse généralement aux enfants.*"

2. Pour plus de relations vous pouvez consulter : https://universaldependencies.org/u/dep/index.html [visité le 2021-09-08]

Dans la sémantique lexicale le but est d'encoder le sens d'un mot. Il faut mentionner que dans les systèmes logographiques, un graphème peut être décrit par sa forme ; par exemple, 川 (rivière), 山 (montagne). Donc, le sens d'un mot est dérivé des sens des graphèmes qui le composent ; par exemple, 音 /on, in, oto, .../ (son, bruit) + 楽 /gaku, tano, raku, .../ (musique, confort, facilité) = 音楽 /ongaku/ (musique). D'une manière générale, le sens peut être représenté en utilisant : un ensemble de "primitives sémantiques", à l'aide de relations entre morphèmes (un réseau de sens), comme un vecteur de nombres réels, etc. Une des théories pour représenter un sens en utilisant les primitives sémantiques est l'analyse sémique. Dans cette dernière, un sème est une unité minimale de signification (trait sémantique minimal). Les sèmes sont définis manuellement. Un sémème est un faisceau de sèmes correspondant à une unité lexicale. Les mots ayant un ou plusieurs sèmes positifs en commun appartiennent au même champ sémantique. La tableau 1.2 représente un exemple de l'analyse sémique des mots "chat", "chien" et "lion" en se basant sur les sèmes : "animé", "domestique" et "félin". La représentation vectorielle peut être vue comme une analyse sémique, où chaque valeur représente le poids d'un sème, sauf que les sèmes ici sont anonymes. On apprend cette représentation, en général, par ce qu'on appelle word **embedding**. Une autre représentation est d'utiliser un réseau de sens où les morphèmes sont liés avec des relations sémantiques. Ces dernières peuvent être :

- **Synonyme** si on substitue un mot par un autre dans une phrase sans changer le sens, donc les mots sont des synonymes
- **Antonyme** le sens opposé. Les deux mots doivent exprimer deux valeurs d'une même propriété. Exemple, grand et petit expriment la propriété taille
- **Hyponyme** un mot ayant un sens plus spécifique qu'un autre. Exemple, chat est l'hyponyme de félin hyponyme de animal.
- **Hyperonyme** un mot avec un sens plus générique. Exemple, félin est l'hyperonyme de chat
- **Méronyme** un mot qui désigne une partie d'un autre. Exemple, roue est un méronyme de voiture

Mot/Sème	animé	domestique	félin
Chat	+	+	+
Lion	+	-	+
Chien	+	+	-

Tableau 1.2 : *Exemple de l'analyse sémique*

Une proposition prend son sens des sens des mots qui la composent. Il existe plusieurs façons pour représenter le sens d'une phrase ; parmi ces méthodes : la logique du premier ordre, la représentation par graphes et la représentation vectorielle. Dans cette dernière, on peut générer le **embedding** comme un vecteur représentant. Une représentation sémantique doit être indépendante de la structure syntaxique. Par exemple, les phrases "*Quelques étudiants possèdent deux ordinateurs*" et "*Deux ordinateurs sont possédés par quelques étudiants*" doivent avoir la même représentation sémantique. Un exemple de quelques représentations sémantiques justes et fausses de ces deux phrases en logique du premier ordre peuvent être :

- $E = Etudiant, O = Ordinateur, P = Possede$
- $\exists x(E(x) \land \forall y(O(y) \Rightarrow P(x,y)))$ ☒
- $\exists x(E(x) \land \exists y,z(((O(y) \land P(x,y)) \land (O(z) \land P(x,z))))$ ☒
- $\exists x(E(x) \land \exists y,z(\neg y = z \land ((O(y) \land P(x,y)) \land (O(z) \land P(x,z))))$ ☑
- $\exists x(E(x) \land \exists y((O(y) \land P(x,twoOf(y))))$ ☑

Parmi les applications de la représentation sémantique d'une proposition : l'inférence. On peut déduire une nouvelle connaissance à partir d'autres existantes. Un exemple de l'inférence dans la logique du premier ordre est donné dans la figure 1.7.

$$\frac{\begin{array}{ll}\text{Chaque homme est mortel.} & \text{Socrate est un homme.}\\ \forall x(Homme(x) \Rightarrow Motel(x)) & Homme(Socrate)\end{array}}{Motel(Socrate)}$$

Figure 1.7 : *Exemple d'inférence dans la logique du premier ordre.*

2.7 Pragmatique et discours

Le niveau pragmatique est responsable de l'étude du sens lié au contexte ; celui difficile à comprendre seulement avec le niveau sémantique. Prenons un exemple : imaginons qu'on est dans un anniversaire et une personne a créé "Les bougie !". On peut directement comprendre qu'il n'y a pas des bougies sur la tarte. Un autre exemple : supposons que vous avez fixé un rendez-vous avec une autre personne qui n'a pas arrivée à temps. Une des réactions que vous pouvez faire est de demander la raison pour laquelle elle a fait de retard. Une façon de le faire est de demander : "A votre avis, quelle heure est-il ?". Si cette personne comprend seulement le niveau sémantique, elle va annoncer l'heure. Sinon, elle va expliquer la raison du retard. Parmi les champs d'intérêt de cette branche, on peut citer :

- **Implicature conversationnelle** : réfère à ce que le locuteur veut dire d'une façon implicite. D'après (Grice, 1979), les interlocuteurs doivent respecter certaines normes (maximes) conversationnelles : quantité, qualité, pertinence et manière. Par exemple, à partir de la phrase "Karim, qui est paresseux, a arrêté de rédiger.", on peut extraire les informations suivantes : "Karim rédigeait.", "Karim est paresseux.", "Karim ne rédige plus." et "Il existe un homme qui s'appelle Karim.".
- **Présupposition** : réfère aux suppositions faites par les interlocuteurs lors de la communication.
- **Acte de langage** : réfère à l'interaction linguistique du locuteur afin d'agir sur son environnement. Une liste des actes de langage est fournie par (Austin, 1962) : déclaration, ordre, question, interdiction, salutation, invitation, félicitation, excuses.

Le niveau du discours s'intéresse aux aspects d'un texte entier comme : la coréférence et la cohérence. Le but de la coréférence est de détecter les références et de les lier avec leurs entités. Cette tâche est vraiment difficile vu qu'elle nécessite une connaissance de l'environnement. Par exemple, dans la phrase "The cat doesn't fit in the box because it is too big.", on sait que "it" référence "The cat". Ceci est dû à la connaissance préalable qu'une chose rentre dans une autre si cette dernière est plus large. En utilisant cette connaissance, on sait que "it" référence "the box" dans la phrase "The cat doesn't fit in the box because it is too small.". Une référence peut se manifester en plusieurs formes :

- **Pronoms** : le chat a chassé une souris et il joue avec elle.
- **Syntagmes nominaux** : Apple est un fabricant d'ordinateur. La firme est mondialement connue.
- **Zero Anaphora** : dans des langues comme le japonais, parfois, la référence est omise.
- **Noms** : International Business Machine est une société américaine. Comme Apple, IBM fabrique des ordinateurs.

Un autre sujet traité au niveau de discours est la cohérence des phrases ; est-ce que l'ordre des phrases est logique ? Afin d'analyser le discours, on peut soit se baser sur sa structure ou sur les entités qui lui composent. Parmi les modèles de discours les plus utilisés, on peut mentionner **Rhetorical Structure Theory (RST)**. C'est un modèle basé sur la structure où on veut trouver les relations entre deux phrases. Parmi les relations de cohérence, on peut citer : raison, élaboration, évidence, attribution, narration, etc. Ce sont des relations binaires entre **noyau** et **satellite**. Prenons la phrase "L'étudiant s'est absenté hier. Il a été malade.". La première phrase est un noyau puisqu'elle décrit l'évènement principal. La deuxième phrase est un satellite puisqu'elle dépend de la première. Voici quelques exemples des relations de cohérence :

- **la raison** : [$_{\text{NOY}}$ L'étudiant s'est absenté hier.] [$_{\text{SAT}}$ Il a été malade.]

— **l'élaboration** : [$_{NOY}$ L'examen est facile.] [$_{SAT}$ Il ne prend qu'une heur.]
— **l'évidence** : [$_{NOY}$ Kevin doit être ici.] [$_{SAT}$ Sa voiture est garée à l'extérieur.]

3 Applications du TALN

Les applications du TALN peuvent être vu selon trois volets : tâches, systèmes complets ou affaires. Comme tâches, on s'intéresse aux applications élémentaires qui peuvent aider d'autres applications. Comme systèmes, on s'intéresse aux différentes applications destinées au utilisateurs. Comme affaires, on s'intéresse comment appliquer le TALN pour résoudre des problèmes de la santé, éducation, etc.

3.1 Tâches

On commence par énumérer quelques tâches de la morphosyntaxe. Ces tâches sont utiles pour les applications de recherche d'information et d'extraction d'information. En général, elles n'ont pas besoin d'une grande quantité de ressources : puissance de calcul et données.

— **Délimitation de la phrase et séparation des mots** : diviser le texte en unités plus petites pour faciliter le traitement. La plupart des applications du TALN essayent de traiter un texte sous forme des unités plus petites : paragraphes, phrases, mots, caractères. D'où la nécessité de cette tâche.
— **Lemmatisation et racinisation** : chercher une forme standard d'un mot. Dans la recherche d'information, si on cherche le terme "formation" il se peut qu'il existe des pages contenant des résultats intéressants mais avec le mot "former". Donc, l'intérêt de cette tâche est de diminuer les variations d'un même concept qui résultent de la formation des mots dans les langues synthétiques.
— **Étiquetage morpho-syntaxique** : trouver les catégories grammaticales des mots d'une phrase. Plusieurs applications peuvent s'améliorer en introduisant les catégories grammaticales des mots. Par exemple, pour comprendre une phrase contenant le mot "fish", il faut savoir si ce mot est un verbe ou un nom.
— **Analyse syntaxique** : trouver la structure syntaxique d'une phrase (comment elle a été formée). Afin de comprendre les relations entre les parties d'une phrase, il faut savoir comment elle sont structurées.
— **Extraction terminologique** : chercher les terminologies existantes dans un texte. Ceci est une application directe de l'extraction de données. Parmi ses applications : la création automatique des bases de données.

Les tâches sémantiques sont plus difficiles et elles ont besoin de plus de ressources. Elles ont comme but la compréhension et la génération du texte.

— **Désambiguïsation lexicale** : trouver le sens d'un mot et sa fonction grammaticale. A cause de la polysémie, on peut trouver un mot avec plusieurs sens et même avec plusieurs fonctions grammaticales. Afin de comprendre une phrase, il faut trouver le sens de ses mots.
— **Étiquetage des rôles sémantiques** : chercher le rôle sémantique (agent, thème, etc.) des mots et syntagmes. Afin de comprendre une phrase, il faut savoir qui a fait l'action, qui a subit l'action, etc. Ceci est utile, aussi, pour l'application de question/réponse où la machine cherche et répondre aux questions des utilisateurs.
— **Reconnaissance d'entités nommées** : extraire les noms de personnes, noms d'organisations ou d'entreprises, noms de lieux, quantités, distances, valeurs, dates, etc. Cette tâche est utile pour la compréhension d'une phrase en essayant d'ajouter plus de contexte. Aussi, elle est utile pour l'extraction de données et la création des bases de données.
— **Analyse sémantique** : trouver la représentation sémantique du texte. Elle est utilisée pour com-

prendre le sens d'une phrase.
- **Paraphrase** : formuler un texte différemment. Ceci est utile lors de la génération du texte ; générer un texte de plusieurs façons.
- **Implication textuelle** : vérifier si un segment du texte est vrai implique qu'un autre est vrai aussi. Ceci est utile pour comprendre l'intention du locuteur.
- **Génération automatique de textes** : générer du texte automatiquement. Elle est utile dans plusieurs applications comme les chatbots, la traduction automatique, etc.

Le discours est un niveau plus avancé que celui de la sémantique ; il a besoin de plus de données. Il est utile pour la compréhension du texte entier et pas seulement phrase par phrase.
- **Résolution de coréférence** : trouver les différentes références dans le texte. Elle est utile dans plusieurs applications ; par exemple lorsqu'on extrait une partie du texte, on veut savoir quelle est la référence d'un pronom personnel.
- **Analyse du discours** : chercher les relations entre les phrases. On peut utiliser cette tâche pour tester si un texte est cohérent, et aussi pour générer un texte cohérent.
- **Résolution d'ellipse** : trouver les éléments omis du texte. Exemple d'ellipse : "Pierre mange des cerises, Paul des fraises". Dans cet exemple, la phrase complète est "Pierre mange des cerises, Paul mange des fraises". Ici, on a omis le verbe puisqu'il est compréhensible d'après le contexte.

3.2 Systèmes

Ici, on va présenter quelques applications du TALN comme un système de bout en bout. Ces systèmes utilisent un ensemble des tâches précédemment présentées afin de satisfaire un besoin. Ils essayent d'automatiser des tâches complexes qui nécessitent un être humain pour être complétées.
- **Traduction automatique** : traduction d'une langue vers une autre. La langue est un outil pour transmettre de la connaissance entre les être humains. Il existe une richesse de langues dans le monde qui nous permet de ressentir la grandeur des êtres humains. Malheureusement, cela conduit aussi à séparer les nations. La traduction automatique peut aider les peoples à communiquer leurs idées et à combler le vide créé par la langue.
- **Résumé automatique** : produire une version compacte du texte. Beaucoup de données sont générées chaque minute sur le web. Prendre compte des informations et des nouvelles est vraiment difficile voir impossible. La quantité d'information n'est pas seulement le seul obstacle, mais aussi sa qualité. Il existe beaucoup de redondance sur le web ; un exemple peut être ressentit dans les réseaux sociaux et la mentalité de copier-coller. Maintenant, imaginons un système qui scanne le web (en ciblant quelque sites) et qui récupère un résumé de tous ce qui est publié. Imaginons qu'il peut trouver les nouvelles et filtrer ce qui est redondant.
- **Questions-réponses** : chercher des réponses aux questions formulées en langage naturel. Souvenons-nous du temps perdu pour trouver une petite information : chercher dans les sites web et les livres, lire des milliers de lignes, comparer les informations trouvées entre deux sites, etc. Dans notre vie, il existe une forte probabilité qu'on ait rencontré quelqu'un qui nous a donné une réponse à une question et nous a épargné la peine de chercher. Si on se dispose d'un système qui peut répondre à des questions génériques ou spécifiques à un domaine (médecine par exemple), ça va sauver beaucoup de temps perdu.
- **Agents conversationnels (chatbots)** : dialoguer avec un utilisateur. Les chatbots sont utiles pour aider l'utilisateur en répondant à ces questions afin d'accomplir une tâche. Par exemple, la réservation des vols en fournissant des informations sur les aéroports, le temps, les pays, etc.
- **Extraction d'information** : extraire des informations ciblées sur le web fin de faire des analyses ou de peupler une base de données.

— **Fouille des textes** : extraction de connaissances à partir d'un texte.
— **Analyse des sentiments** : trouver les sentiments existants dans un texte. Par exemple, vérification si des utilisateurs sont satisfaits par un produit ou non.
— **Recommandation automatique des documents** : présenter des éléments qui sont susceptibles d'intéresser un utilisateur. Ex., recommander des livres.

3.3 Affaires (business)

Cette siècle est connue par l'énorme quantité des données accessibles sur le web. Une grande proportion de ces données est sous format textuelle non structurée. Elle contient des informations et des opinions de tout genre. Beaucoup d'entreprises ont ressenti la nécessité d'exploiter ces ressources afin d'améliorer leur commerce. Parmi les applications du TALN dans le monde de la commerce, on peut citer :

— **Publicité** : identifier de nouveaux publics potentiellement intéressés par certains produits.
— **Service clientèle** : utiliser des chatbots pour répondre aux questions potentielles des clients. Aussi, utiliser l'analyse de sentiments pour avoir une idée sur l'opinion des clients sur les produits de l'entreprise.
— **Intelligence de marché** : surveiller les blogs, les sites web et les réseaux sociaux pour analyser les tendances du marché et pour avoir une idée sur la compétition.
— **Recrutement** : filtrer les CVs pour trouver des candidats plus rapidement et sans biais.

Afin d'améliorer la communication entre citoyens et gouvernement, le TALN peut être utilisé dans le domaine de E-Gouvernance. Des cas d'utilisation du TALN dans ce domaine peuvent être résumés dans ces points :

— **Communication gouvernement/citoyens** : les citoyens analphabètes peuvent partager leurs opinions en utilisant des audio/vidéo, qui peuvent être traduites en texte. De même, un message aux citoyens peut être transformé à un message vocal.
— **Fouille d'opinions** : extraction des commentaires, des plaintes et des critiques sur une politique particulière, déterminant ainsi le consensus général à ce sujet.

Un des domaines qui s'intéressent de plus en plus aux applications du TALN est le domaine de la santé. En effet, le TALN peut vraiment améliorer plusieurs tâches dans ce domaine puisque ce dernier génère et nécessite beaucoup de documents y compris ceux textuels. Améliorer l'accès à l'information peut sauver pas seulement le temps, mais aussi beaucoup de vies. Parmi les tâches où le TALN peut jouer un grand rôle, on peut citer :

— Structurer les documents médicaux afin de faciliter leur exploitation.
— Rechercher, analyser et interpréter des quantités gigantesques d'ensembles de données de patients.
— Prédire les maladies en se basant sur les symptômes et les documents médicaux.
— Générer des rapports.
— Utiliser des assistants virtuels pour monitorer et aider les patients.

La langue est un outil pour communication, d'où l'intérêt d'apprendre au moins une. Les élèves apprennent une langue en interagissant avec un éducateur et en passant des évaluations afin de juger les lacunes et de les améliorer. Le TALN, vu qu'il fourni des tâches liées aux langues, peut aider dans le domaine de l'éducation. Parmi les tâches d'un éducateur qu'on peut automatiser, on peut énumérer :

— Évaluation de la langue : lire, écrire et parler.
— Correction des erreurs d'orthographe
— Évaluation automatique des travaux des élèves, tels que les essais et les réponses

— Apprentissage en ligne en intégrant des chatbots avec des jeux, ce qui favorise un environnement d'apprentissage actif

4 Défis du TALN

Chaque domaine a ces propres défis ; le TALN n'est pas une exception. Les variétés des langues, la non standardisation, l'évolution des langues et tous les aspects npn prédictible des être humains rendent ce domaine vraiment difficile. Ici, on va discuter quelques défis qui ne prouvent pas seulement que ce domaine est difficile, mais aussi qui motivent son importance.

4.1 Ressources

Les ressources que se soit en terme de la puissance de calcul ou en terme de données posent un problème surtout avec les tâches qui ne peuvent être résolues qu'avec l'apprentissage automatique. La plupart de ces tâches ont besoin d'un apprentissage supervisé, d'où la nécessité d'annotation manuelle des corpus d'entraînement et de test. Plusieurs chercheurs et développeurs évitent l'annotation en cherchant des datasets déjà annotées et en modifiant le problème initial (résoudre un problème en résolvant un autre). Cela peut affecter la qualité de l'outil final. Si on parle de la taille des documents, lorsqu'on doit traiter des documents plus larges ça va complexer la tâche. En optant d'une solution par apprentissage, des fois un modèle va avoir du mal à représenter des contextes longs. Jusqu'à maintenant, j'ai discuté les défis en terme de ressources dans le cas d'une langue bien visible. Lorsqu'on veut traiter une langue moins parlée, on ne va pas tomber seulement dans le problème des données absents, mais aussi le manque des outils. Supposant qu'on veuille concevoir un système de traduction vers une langue rare or nous ne disposons même pas d'un outil pour la segmentation des mots.

4.2 Compréhension de la langue

L'ambigüité est le plus grand défis du traitement d'une langue. Même les être humains ont du mal à comprendre certains passages dû à l'ambigüité. Ce problème est originaire des propriétés des langages naturels, parmi lesquelles on peut citer :

— **polysémie** : un mot ayant plusieurs sens. Ex. "Indien : de l'Inde, indigène des Amériques".
— **énantiosémie** : un terme polysémique ayant deux sens antonymes. Par exemple, le mot "Regret" désigne la plainte ou la nostalgie : "Je regrette mon enfance".
— **Trope** (langage figuratif) : métaphores (Ex. "It's raining cats and dogs"), métonomie (Ex. "La salle a applaudi" [les gens dans la salle]), ironie (Ex. "Quelle belle journée !" [pour signifier qu'il pleut des cordes.])
— Les coréférences dans les textes longs (ambigüité dans le choix de référence). Ex. "Table data is dumped into a delimited text file, which is sent to the remote site where it is loaded into the destination database.".

Le défis précédent suppose que le locuteur utilise un langage standard, simple, directe et objectif. Si une de ces propriétés sera absente dans le langage, ce dernier sera plus difficile à comprendre même par des être humains. Parmi les aspects humains qui peuvent aggraver la difficulté du traitement d'un discours, on peut énumérer :

— **Personnalité** : comment modéliser une personnalité ? Comment la personnalité affect le discours ?
— **Variations et communication non standard** : les utilisateurs ne respectent pas les standards d'écriture d'une langue. Ex., langue de chat, arabizi, franglais, etc.

— **Intention** : il existe des phrases qui ne veulent pas dire ce qu'on comprenne directement ; elles veulent une autres chose
— **Émotions** : les phrases peuvent changer de sens selon les émotions accompagnées

4.3 Évaluation

L'évaluation automatique est vraiment difficile surtout pour certains aspects de la langue, comme par exemple la cohérence. Si on arrive à évaluer un aspect automatiquement, on peut utiliser le même algorithme d'évaluation pour l'améliorer. La solution la plus idéale pour tester qu'un système est conforme à une tâche humaine est de le faire passer par un humain. Ceci prend du temps (coût élevé en terme de temps). En plus, l'évaluation nécessite des experts (coût élevé en terme de dépenses). Une évaluation manuelle peut causer le problème de subjectivité de l'évaluateur et de l'agrément inter-évaluateurs.

4.4 Éthique

Comme tout outil, le TALN peut être utilisé d'une mauvaise façon. C'est un outil très puissant si on arrive à le maitriser ; ça sera d'une grande utilité à l'humanité. Dans l'autre coté, il peut être utilisé contre un groupe d'humains que se soit avec intention ou non. Même si on ne veut pas l'utiliser pour nuire à un groupe, cela peut être fait indirectement. Donc, avant de concevoir un système il faut vérifier quelques points comme :

— **Vie privée** : les données collectées peuvent contenir des informations privées des individus. Des entreprises peuvent stocker des informations sur leurs utilisateurs.
— **Biais et discrimination** : les corpus utilisés pour entraîner un système peuvent causer du biais. Par exemple, on a remarqué que les **embeddings** (représentation des mots) attribuent des jobs comme "docteur" aux hommes et "infirmière" aux femmes. Le système ne peut pas comprendre qu'il existe des docteures (femelles) ou des infirmiers (mâles).
— **Utilisation** : espionnage et ingénierie sociale, perte d'emplois à cause de l'automatisation, etc.

Discussion

Pouvez-vous comprendre cette phrase ? Quelque soit votre réponse (oui ou non), vous avez certainement compris la question ; sinon, vous n'aurez pas à répondre en premier lieu. Imaginez une machine qui peut raisonner comme ça. Imaginez qu'elle peut comprendre votre langue et qu'elle peut répondre d'une façon similaire à la vôtre. Cela était considéré de la science fiction dans les années 50. Il l'est toujours, mais de nos jours on est plus proche de ce rêve qu'avant. Le domaine du TALN, et de l'IA en général, a passé par des belles périodes et d'autres difficiles. Au long de son évolution, même au milieu de l'hiver, il y avait toujours des savants qui ont combattus afin d'avancer ce domaine et afin d'atteindre ce rêve. Le combat continue ...

Cette machine doit absolument percevoir la parole. Si elle peut entendre sans parler, ça sera drôle ! Pour lui accorder cette capacité, on doit utiliser la phonétique. Puis, on doit lier les sons à une langue donnée ; d'où la nécessité de la phonologie. Afin qu'elle puisse écrire, elle doit maitriser l'orthographe. Pour former les mots et les utiliser d'une façon plus naturelle, elle a besoin de la morphologie. Elle doit comprendre la structure d'une phrase en se basant sur la syntaxe. Elle doit comprendre le sens en utilisant la sémantique. Dans des cas, elle doit utiliser le contexte pour comprendre en impliquant la pragmatique. Enfin, elle doit pouvoir comprendre un texte entier ; d'où la nécessité du discours.

Une fois complète, cette machine peut vous aider afin d'accomplir plusieurs tâches. Elle aura la capacité de traduire tous ce que vous voulez dans une instance. Elle peut vous sauver le temps en résumant les documents et en répondant aux différentes questions. Elle peut mener des conversations avec les gens pour les guider ou pour passer du temps. Si vous avez une entreprise, vous pouvez compter sur elle pour gérer la publicité, le service clientèle, le recrutement et l'étude du marché. Elle peut même diminuer le vide entre les gouvernements et leurs citoyens en améliorant la communication. Surtout, elle peut améliorer les systèmes de santé et de l'éducation.

Une telle machine vient avec un coût, et son coût est de perdre votre temps afin qu'elle apprenne. Il faut que vous choisissiez les meilleures données et de les annoter avec grande quantité. Il faut lui apprendre à faire face à l'ambigüité, jusqu'à ce que l'élève dépassera son maitre. Il faut évaluer ses tâches et l'améliorer jusqu'à ce qu'elle devient comme un expert. Enfin, il faut l'utiliser seulement pour faire du bien. Sinon, une fois était un rêve, cette machine pourrait devenir un cauchemar.

CHAPITRE 2

TRAITEMENTS BASIQUES DU TEXTE

LES LANGUES

En traitant un langage, on commence par sa plus petite unité : le graphème. En informatique, les graphèmes plus les ponctuations et les espacements sont appelés caractères. Il existe des opérations qui nous permet de chercher et de comparer les chaînes de caractères. En combinant les caractères, on va avoir un texte avec des phrases et des mots. Étant donné un texte, on doit pouvoir le segmenter en phrases et une phrase en mots. Ces derniers peuvent être inutiles pour certains tâches; comme les prépositions dans la recherche d'information. Donc, il faut les filtrer avant d'appliquer cette tâche. Aussi, les mots peuvent prendre plusieurs variations morphologiques. Si on veut traiter une seule variation, il faut les normaliser. En plus, on doit pouvoir générer ces variations ou passer d'une variation à une autre étant donné une forme d'un mot. Ce chapitre résume quelques tâches du niveau morphologique des langages (analyse lexicale).

D'après Larousse, un caractère est définie dans l'informatique et télécommunications comme : "*Tout symbole (chiffre, lettre de l'alphabet, signe de ponctuation, etc.) employé pour représenter des données en vue de leur traitement ou de leur transmission.*". Un mot est composé de plusieurs caractères suivant un langage régulier. Une phrase, à son tour, se compose de plusieurs mots suivant un langage hors contexte (dans la plupart des langues). Cette dernière proposition concerne le niveau syntaxique. Pour l'instant, on s'intéresse au niveau morphologique ou ce qu'on appelle l'analyse lexicale. Les points traités dans ce chapitre sont les suivants :

- La recherche dans le texte en utilisant les expressions régulières. Parmi les applications : l'extraction des données (dates, numéros de téléphones, adresses émail, etc.).
- La comparaison entre les chaines de caractères en utilisant la distance d'édition. Parmi ses applications : la correction d'orthographe et la recherche approximative.
- La segmentation du texte en phrases et les phrase en mots.
- La normalisation du texte afin de diminuer les variations d'un mots.
- La formation des mots dans les langues synthétiques et l'opération inverse.

1 Traitements sur les caractères

Ici, on va considérer un texte comme une séquence de caractères. D'une manière plus formelle, un texte peut être composé en utilisant un automate à états finis où le vocabulaire est l'ensemble des caractères. Pour chercher des sous-chaines de caractères dans un texte, on peut utiliser les expressions régulières reconnaissant des langages de types 3 (langages réguliers) dans la hiérarchie de Chomsky. La recherche des séquences dans un texte nous permet de les remplacer ou de les séparer (ex. séparation des mots). Un autre type de recherche est la recherche approximative : chercher des parties presque similaires à une chaîne donnée. Pour ce faire, on doit pouvoir comparer deux chaînes de caractères en mesurant la

différence entre les deux. Une des techniques utilisées est la distance d'édition.

1.1 Expressions régulières

Une expression régulière, appelée **RegEx**, est une séquence de caractères spécifiant un motif (pattern) de recherche. Plusieurs langages de programmation fournissent la capacité de recherche et de remplacement en utilisant les expressions régulières. Afin de l'utiliser pour la recherche, une expression régulière est transformée à un automate à états finis (AEF) qui est transformé à son tour à un AEF déterministe. Un point dans une expression régulière représente n'importe quel caractère. Si on veut rechercher un point et pas n'importe quel caractère, on ajoute un backslash avant le point. On peut concevoir une expression régulière complexe en composant plusieurs expressions régulières simples. Ces dernières sont décrites dans le tableau 2.1 avec leurs sens et des exemples.

ER	Sens	Exemple
.	n'importe quel caractère	/beg.n/ : I begun at the beginning
[aeuio]	caractères spécifiques	/[Ll][ae]/ : Le chat mange la sourie
[a-e]	plage de caractères	/[A-Z]../ : J'ai vu Karim
[^aeuio]	exclure des caractères	/[^A-Z]a./ : J'ai vu Karim
c?	un ou zéro	/colou?r/ : It is colour or color
c*	zéro ou plus	/No*n/ : Nn ! Non ! Noooooooon !
c+	un ou plus	/No+n/ : Nn ! Non ! Noooooooon !
c{n}	n occurrences	/No{3}n/ : Nn ! Non ! Noon ! Nooon !
c{n,m}	de n à m occurrences	/No{1,2}n/ : Nn ! Non ! Noon ! Nooon !
c{n,}	au moins n occurrences	/No{2,}n/ : Nn ! Non ! Noon ! Nooon !
c{,m}	au plus m occurrences	/No{,2}n/ : Nn ! Non ! Noon ! Nooon !
\d	[0-9]	/\d{2,}/ : L'annee 1962
\D	[^0-9]	/\D{2,}/ : L'annee 1962
\w	[a-zA-Z0-9_]	/\w{2,}/ : L'annee 1962
\W	[^\w]	/\W{2,}/ : L'annee 1962
\s	[\r\t\n\f]	/\s+/ : L'annee 1962
\S	[^\s]	/\S+/ : L'annee 1962
( )	le groupement	/(bla)+/ : Ceci est du blabla
|	la disjonction	/continu(er|ation)/ : Je continue la continuation
^	début du texte	/^K/ : Kill Karim
$	fin du texte	/\.[^.]+$/ : fichier.tar.gz

Tableau 2.1 : *Expressions régulières*

La plus fréquente utilisation des expressions régulières est la recherche des sous-chaînes de caractères dans un grand texte. La plupart des éditeurs de textes et des langages de programmation fournissent des mécanismes pour utiliser les expressions régulières. Utiliser des motifs (patterns) pour chercher des chaînes de caractères rend les expressions régulières un outil très puissant pour l'extraction de données. Par exemple, on peut utiliser les expressions régulières pour extraire les émails à partir des blogs et des réseaux sociaux. Prenons un exemple d'une expression régulière /[a-zA-Z]\w*(@| at)[a-zA-Z]\w+\.[a-zA-Z]{2,3}/ pour chercher des adresses mail. La partie /[a-zA-Z]/ garantie que le nom d'utilisateur et le nom du domaine commencent par une lettre. Le nom d'utilisateur accepte au moins un caractère puisque la première lettre

est suivie par /\w*/ qui veut dire zéro à plusieurs caractères de type lettre, chiffre ou souligné. Le nom du domaine dans cette expression régulière accepte au moins deux caractères puisque la première lettre est suivie par /\w+/ qui veut dire un à plusieurs caractères de type lettre, chiffre ou souligné. Le nom d'utilisateur et le nom du domaine sont séparés soit par un "@" ou par "at "; ce qui est indiqué par /(@| at)/. L'extension du domaine doit toujours être précédée par un point ; ici, on utilise /\./ pour indiquer qu'il s'agit du caractère point. Sinon, si on utilise /./, ceci représente n'importe quel caractère. L'extension est composée de deux à trois lettres. Voici quelques émails qu'on puisse extraire en utilisant cette expression régulière : "ab_aries@esi.dz", "ab_ARIES@eSi.dz", "ab_aries at esi.dz", "a@es.edu", "a1@e2s.edu".

Afin de capturer une chaîne de caractère, on utilise le regroupement avec le numéro du groupe. Le numéro du groupe "N" doit être précédé par un caractère spécial , souvent "$" ou "\". Un exemple de remplacement de texte dans un éditeur de texte ("kate" dans notre cas) est illustré dans la figure 2.1. Ici, on cherche le mot "chat" pour le remplacer par le mot "chien". Mais, on veut garder "ch" comme c'est partagé entre les deux mots. Dans ce cas, on capture la chaîne "ch" pour la garder et remplacer le reste : "at" par "ien".

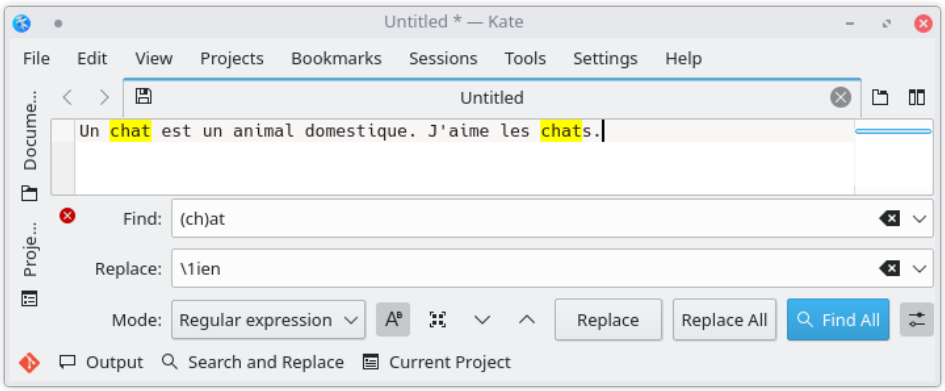


Figure 2.1 : *Exemple de remplacement par expressions régulières dans l'éditeur de texte "kate"*

Peut-être l'utilité du regroupement n'est pas claire avec l'exemple précédent puisque nous avons une seule chaîne à capturer. Si nous avons une disjonction, ça sera plus intéressant. Prenons l'exemple : "J'aime les chats. Dans la cuisine, il y a un rat.". Si on remplace la chaîne "at" de "chat" et de "rat" par "ien", on aura le texte suivant : "J'aime les chiens. Dans la cuisine, il y a un rien.". Dans ce cas, on doit tester les mots "chat" et "rat" pour remplacer seulement la dernière partie et garder la première intacte. Plusieurs langages de programmation fournissent le remplacement des chaînes de caractères en utilisant les **RegEx**. Dans Javascript, les chaînes de caractères sont attribuées une méthode "replace" pour remplacer une de leurs parties par une autre chaîne. Si on veut remplacer toutes les occurrences, on doit utiliser le flag "g" après l'expression régulière. Pour récupérer le groupe capturé, on utilise "$".

```
1 let orig = "J'aime les chats. Dans la cuisine, il y a un rat.";
2 let remp = orig.replace(/(ch|r)at/g, "$1ien");
```

Pour toute utilisation des expressions régulières en Python, on doit faire appel au module "re". Une expression régulière est une chaîne de caractères précédée par l'indicateur "r". Pour récupérer le groupe capturé, on utilise "\".

```
1 import re
2 orig = "J'aime les chats. Dans la cuisine, il y a un rat."
3 remp = re.sub(r'(ch|r)at', r'\1ien', orig)
```

1.2 Distance d'édition

Des fois, lorsqu'on écrit en utilisant nos ordinateurs, on fait des erreurs de frappe ; on peut insérer un caractère de plus, dupliquer un caractère, etc. Une telle opération est appelée "opération d'édition". On peut comparer entre deux chaînes de caractères en comptant le nombre des opérations d'édition pour passer d'une chaîne originale à une autre modifiée. Les différentes opérations d'édition sont les suivantes (où X est l'ensemble des caractères du langage) :

- **Insertion** : insertion d'un caractère dans une chaîne ($uv \rightarrow uxv \,/\, u, v \in X^*;\ uv \in X^+;\ x \in X$). Par exemple, courir → courrir, entraînement → entraînnement.
- **Suppression** : suppression d'un caractère d'une chaîne ($uxv \rightarrow uv \,/\, u, v \in X^*;\ uv \in X^+;\ x \in X$). Par exemple, héros → héro, meilleur → meileur.
- **Substitution** : substitution d'un caractère par un autre ($uxv \rightarrow uyv \,/\, u, v \in X^*;\ x, y \in X;\ x \neq y$). Par exemple, cela → celà, croient → croyent
- **Transposition** : changement de l'ordre de deux caractères ($uxwyv \rightarrow uywxv \,/\, u, v, w \in X^*;\ x, y \in X;\ x \neq y$). Par exemple, cueillir → ceuillir.

Savoir le nombre des opérations d'édition nous permet de comparer deux chaînes de caractères. Il existe plusieurs exemples de l'utilisation de la distance d'édition (nombre des modifications) :

- **Révision des fichiers** : par exemple, la commande Unix "diff" qui compare deux fichiers.
- **Correction d'orthographe** : suggérer des corrections possibles d'une faute (Ex. Hunspell).
- **Détection du plagiat** : ici, on utilise des mots à la place des caractères.
- **Filtrage de spam** : parfois, les spammeurs commettent des fautes d'orthographe intentionnellement pour tromper l'outil de détection de spam.
- **Bio-informatique** : quantification de la similarité entre deux séquences d'ADN.

Distance de Hamming

Cette distance permet seulement la substitution. Les chaînes doivent être de la même longueur. Parmi ses utilisations, on peut citer la détection des erreurs de transmission de données.
Par exemple, $D(0100101001\underline{1}0, 0101101\underline{1}0\underline{1}0\underline{0}) = 3$. Pour calculer la distance, on compare les deux chaînes caractère par caractère pour avoir le nombre des caractères différents (voir l'algorithme 2.1).

Algorithme 2.1 : *Calcul de la distance de Hamming*

```
Data : orig, modif
Result : distance : entier
distance ← 0 ;
pour pos ∈ 1 . . . |orig| faire
    si orig[pos] ≠ modif[pos] alors
        incrémenter distance ;
    fin
fin
```

Plus longue sous-séquence commune

Cette distance permet l'insertion et la suppression. Parmi ses utilisations, la détection des modifications dans le programme "diff" (utilisé aussi dans "Git"). En anglais, elle s'appelle "Longest common subsequence" (LCS). Étant donné deux chaînes X et Y avec les longueurs n et m respectivement, on définit un tableau $C[m, n]$ à deux dimensions contenant la longueur de la LCS. L'équation 2.1 peut être utilisée afin

de calculer la longueur de la plus longue sous-séquence.

$$C[i,j] = \begin{cases} 0 & \text{Si } i = 0 \text{ ou } j = 0 \\ C[i-1, j-1] + 1 & \text{Si } i, j > 0 \text{ et } x_i = y_j \\ max(C[i, j-1], C[i-1, j]) & \text{Si } i, j > 0 \text{ et } x_i \neq y_j \end{cases} \tag{2.1}$$

Dans ce cas, la longueur de la plus longue sous-séquence $|LCS(X,Y)| = C[m,n]$. La distance sera calculée en utilisant l'équation 2.2.

$$D(X,Y) = m + n - 2|LCS(X,Y)| \tag{2.2}$$

Distance de Levenshtein

Cette distance permet l'insertion, la suppression et la substitution. En général, elle est utilisée dans la recherche approximative et la vérification d'orthographe. Étant donné deux chaînes X et Y avec les longueurs n et m respectivement, on définit un tableau $D[m,n]$ à deux dimensions contenant la distance d'édition entre les sous-chaînes X[1..i] et Y[1..j]. Dans ce cas $D[0,0] = 0$ et la distance finale sera stockée dans $D[m,n]$. L'équation 2.3 formule comment on calcule cette distance (en utilisant la programmation dynamique). Il faut savoir que dans cette version, le coût de la suppression et l'insertion est 1 et le coût de la substitution est 2.

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 \text{ //Suppression} \\ D[i,j-1] + 1 \text{ //Insertion} \\ D[i-1,j-1] + \begin{cases} 2 & \text{si } x_i \neq y_j \\ 0 & \text{sinon} \end{cases} \end{cases} \tag{2.3}$$

Par exemple, D(intention, execution) = 8. Dans ce cas, "i" est supprimé (1), "n" est substitué par "e" (2), "t" est substitué par "x" (2), "e" c'est le même, "c" est inséré après "e" (1), "n" est substitué par "u" (2) et le reste c'est le même. Le calcul est illustré dans la figure 2.2 où les flèches représentent d'où vient la valeur et les cellules colorées représentent le chemin choisi. Bien sûr, on peut choisir un autre chemin qui donne une autre interprétation. Pour aider le choix du chemin, on peut utiliser des probabilités des opérations d'édition. Par exemple, dans la correction d'orthographe, des lettres sont plus probables d'être remplacées par d'autres (celles adjacentes dans le clavier).

	#	e	x	e	c	u	t	i	o	n
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ **1**	↖↑ 2	↖↑ 3	↖↑ 4	↖↑ 5	↖↑ 6	↖↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖↑ **3**	↖↑ 4	↖↑ 5	↖↑ 6	↖↑ 7	↖↑ 8	↑ 7	↖↑ 8	↖ 7
t	↑ 3	↖↑ 4	↖↑ **5**	↖↑ 6	↖↑ 7	↖↑ 8	↖ 7	↑ 8	↖↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← **5**	← **6**	← 7	↑ 8	↖↑ 9	↖↑ 10	↑ 9
n	↑ 5	↑ 4	↖↑ 5	↖↑ 6	↖↑ 7	↖↑ **8**	↖↑ 9	↖↑ 10	↖↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖↑ 6	↖↑ 7	↖↑ 8	↖↑ 9	↖ **8**	← 9	← 10	↑ 11
i	↑ 7	↑ 6	↖↑ 7	↖↑ 8	↖↑ 9	↖↑ 10	↑ 9	↖ **8**	← 9	← 10
o	↑ 8	↑ 7	↖↑ 8	↖↑ 9	↖↑ 10	↖↑ 11	↑ 10	↑ 9	↖ **8**	← 9
n	↑ 9	↑ 8	↖↑ 9	↖↑ 10	↖↑ 11	↖↑ 12	↑ 11	↑ 10	↑ 9	↖ **8**

Figure 2.2 : *Exemple de calcul de distance de Levenshtein (Jurafsky et Martin, 2019)*

Distance de Damerau - Levenshtein

Cette distance permet l'insertion, la suppression, la substitution et la transposition entre deux caractères adjacents. En général, elle est utilisée dans la vérification d'orthographe. Cette distance est calculée comme

celle de Levenstein, en ajoutant la transposition entre deux caractères adjacents. L'équation 2.4 représente comment calculer cette distance. Dans cette version, on a essayé d'attribuer le poids 1 pour toutes les opérations d'édition.

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 \text{ //Suppression} \\ D[i,j-1] + 1 \text{ //Insertion} \\ D[i-1,j-1] + \begin{cases} 1 & \text{si } x_i \neq y_j \text{ //Substitution} \\ 0 & \text{sinon} \end{cases} \\ D[i-2,j-2] + 1 \text{ si } x_i = y_{j-1} \text{ et } x_{i-1} = y_j \text{ //Transposition} \end{cases} \tag{2.4}$$

Distance de Jaro

Cette distance permet seulement la transposition. En réalité c'est une mesure de similarité qui retourne une valeur entre 0 (pas de similarité) et 1 (similaires). Elle est utilisée pour le calcul de la similarité entre les entités nommées, etc. Étant donné deux chaînes X et Y avec les longueurs n et m respectivement, on calcule le nombre des caractères correspondants c et le nombre des transpositions t. La distance de Jaro est calculée selon l'équation 2.5.

$$D(X,Y) = \begin{cases} 0 & \text{si } c = 0 \\ \frac{1}{3}(\frac{c}{m} + \frac{c}{n} + \frac{c-t}{c}) & \text{sinon} \end{cases} \tag{2.5}$$

Le nombre des caractères correspondants c est le nombre des caractères identiques de X et de Y avec un éloignement $e = \max(m,n)/2 - 1$. Dans ce cas, si on est dans le caractère i (allant de 0 jusqu'à $n-1$) du mot X, on commence la comparaison par le caractère $j = \max(0, i-e)$ du mot Y et on termine par le caractère $j = \min(i+e, m-1)$. Dans cette opération, on peut préparer deux vecteurs des booléens qui indiquent la position du caractère ayant un correspondant dans l'autre mot.

Le nombre des transpositions t est calculé en comparant le i$^{\text{ème}}$ caractère correspondant de X avec le i$^{\text{ème}}$ caractère correspondant de Y. On compte le nombre des fois $x_i \neq y_i$ et on le divise par deux.

Algorithme 2.2 : *Calcul du nombre de transpositions entre deux mots X et Y dans la distance de Jaro*

```
Data : Xmatch, Ymatch : vecteurs de booléens
Result : t : entier
t ← 0 ;
j ← 0 ;
pour i ∈ 0 . . . m faire
    si Xmatch[i] = Vrai alors
        tant que Ymatch[j] ≠ Vrai faire
            incrémenter j ;
        fin
        si x_i ≠ y_j alors
            incrémenter t ;
        fin
        incrémenter j ;
    fin
fin
t ← t/2 ;
```

Prenons l'exemple de "amibe" et "immature" ayant les tailles 5 et 8 respectivement. Donc, l'éloignement $e = \max(5,8)/2 - 1 = 3$. La figure 2.3 représente le calcul de la distance entre ces deux mots en utilisant les deux ordres possibles. Dans la matrice, 0 veut dire "Faux" et 1 veut dire "Vrai" (correspondance).

Les cases en gras représentent la plage de recherche en utilisant l'éloignement de 3. Les cases soulignées représentent l'intersection entre la ième correspondance de X et la ième correspondance de Y. Les correspondances soulignées des vecteurs des deux mots représentent des transpositions.

	a	m	i	b	e	
i	**0**	**0**	**1**	**0**	**0**	1
m	**0**	**1**	**0**	**0**	**0**	1
m	**0**	**0**	**0**	**0**	**0**	0
a	**1**	**0**	**0**	**0**	**0**	1
t	0	**0**	**0**	**0**	**0**	0
u	0	0	**0**	**0**	**0**	0
r	0	0	0	**0**	**0**	0
e	0	0	0	0	**1**	1

1	1	1	0	1

	i	m	m	a	t	u	r	e	
a	**0**	**0**	**0**	**1**	0	0	0	0	1
m	**0**	**1**	**0**	**0**	**0**	0	0	0	1
i	**1**	**0**	**0**	**0**	**0**	**0**	0	0	1
b	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0
e	0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	1

1	1	0	1	0	0	0	1

Figure 2.3 : *Exemple de calcul de la distance de Jaro des mots "amibe" et "immature"*

2 Segmentation du texte

Un texte peut être traité lorsqu'il est segmenté en passages de petites tailles ; elles peuvent êtres des chapitres, des paragraphes, des phrases ou des mots selon la granularité voulue. Dans plusieurs langues, les phrases sont délimitées par un point ou une marque spécifique, et les mots sont délimités par des espaces. Même dans ces langues, le délimiteur peut être utilisé pour d'autres fins. Il existe des langues où il n'y a pas de délimiteur des phrases, des mots ou des deux.

2.1 Délimitation de la phrase

Afin de séparer les phrases dans un format semi-structuré, comme HTML, on peut utiliser des marqueurs comme la balise <P>. Mais, lorsqu'il s'agit d'un texte non structuré, plusieurs langues utilisent des marqueurs comme le point, point d'exclamation et point d'interrogation pour marquer la fin d'une phrase. Dans ce cas, on peut utiliser une expression régulière simple /[.?!]/ pour délimiter les phrases dans des langues comme français, anglais, etc. Des fois, on veut séparer les phrases longues avec des clauses séparées par des virgules. Aussi, les guillemets peuvent pauser un problème : est-ce que la phrase dedans est séparée ou elle est une continuation de la clause principale ? Par exemple, Il a dit : "Je suis fatigué." en retournant à son lit.. Le point dans ces langues n'est pas toujours utilisé pour séparer les phrases. Il peut être utilisé dans les nombres : 123,456.78 (style américain) 123.456,78 (style européen). Les abréviations comme "Mr.", "Dr." et "etc." contiennent des points et peuvent se trouver au milieu de la phrase. Ils peuvent, aussi, se trouver à la fin ; donc, il faut vraiment être capable de détecter les abréviations et s'ils marquent la fin de la phrase ou non. Si tous ça ne parait pas problématique, on peut toujours essayer de séparer les phrase du thaï. Cette langue n'utilise pas des marqueurs pour séparer les phrases.

Lorsque le marqueur de phrase est réservé pour d'autres utilisations, il y a toujours des facteurs qui aident le lecteur à décider si c'est un délimiteur de phrase ou non. Ces facteurs peuvent être extraits en observant comment la langage définit la fin de la phrase. Quelques facteurs contextuelles ont été proposés et utilisés pour la segmentation des phrases (Palmer, 2010) :

— **La casse** : les phrases commencent toujours par un majuscule. Mais, ce n'est pas toujours le cas ; on peut trouver une abréviation suivie par un nom propre (Ex. "M. Aries"). En plus, ce n'est pas

garantie que l'écrivain respecte les règles d'écriture. On peut voir ça, par exemple, dans les réseaux sociaux où plusieurs règles sont abandonnées.

— **Noms propres** : les noms propres se commencent par un majuscule ; ils peuvent ne pas être le début. Dans l'exemple précédent ("M. Aries"), on peut déduire que le point ne représente pas une séparation vu que les deux mots commencent par un majuscule et le deuxième est un nom propre. Dans ce cas, la probabilité que le premier soit une abréviation est grande surtout si le mot est au milieu de la phrase.

— **Catégorie grammaticale** : les catégories des mots qui entourent le point peuvent aider la décision (limite ou non). En fait, Palmer et Hearst (1997) ont été capables d'améliorer la détection des limites des phrases en utilisant les catégories grammaticales des deux mots avant et après le point avec un algorithme d'apprentissage automatique.

— **Longueur du mot** : les abréviations sont moins longues. Revenons toujours à l'exemple précédent, il est clair que "M" n'est pas vraiment un mot.

— **Préfixes et suffixes** : les mots avec des affixes sont moins probables d'être des abréviations.

— **Classes des abréviations** : les abréviations peuvent se figurer à la fin de la phrase. Mais, il y a un ensemble d'abréviations qui sont toujours suivies par un autre mot, comme par exemple "Mr.". Riley (1989) ; Reynar et Ratnaparkhi (1997) divisent les abréviations en deux catégories : les titres (qui ne peuvent pas être à la fin de phrase (Ex. "Mr.", "Dr.", etc.) et les indicatifs corporatifs (qui peuvent être à la fin de phrase (Ex. "Corp.", "S.P.A.", etc.).

La détection automatique des limites d'une phrase peut être accomplie en utilisant des règles manuelles ou en utilisant l'apprentissage automatique. Dans la première approche, on peut utiliser des expressions régulières pour détecter les délimiteurs. Ensuite, on peut utiliser une liste des abréviations pour améliorer la décision. Les règles peuvent être enrichies en introduisant les facteurs discutés précédemment. Afin d'éviter l'écriture manuelle de ces règles, on peut utiliser un algorithme d'apprentissage automatique qui classe le point comme délimiteur/non-délimiteur en se basant sur ces même règles.

2.2 Séparation des mots

Plusieurs langues (arabe, français, anglais, etc.) utilisent l'espace comme délimiteur des mots. Une expression régulière simple comme /[ ]+/ peut être utilisée pour séparer les mots. Mais, des fois on veut récupérer une expression avec plusieurs mots ; comme le cas des dates, des nombres, etc. Un exemple des nombres est "neuf cent quarante" ; cette expression peut être considérée comme un seul token dans certains traitements. Dans des langues, l'apostrophe peut être une source d'ambigüité. Dans l'anglais, l'apostrophe peut être utilisée avec un "*s*" dans la forme possessive (Karim's thesis), dans les contractions (she's, it's, I'm, we've) ou dans le pluriel de certains mots (I.D.'s, 1980's). Dans le français, il y a pas mal d'exemples de contractions : la contraction des articles (l'homme, c'était), la contraction des pronoms (j'ai, je l'ai), et autres formes (n'y, qu'ils, d'ailleurs). Certaines langues utilisent des mots composés, soit par composition ou par trait d'union. Dans l'allemand, il est commun d'utiliser la composition des mots : nom-nom (Lebensversicherung : assurance vie), adverbe-nom (Nichtraucher : non-fumeur), et préposition-nom (Nachkriegszeit : période d'après-guerre). Autres langues utilisent le trait d'union, comme l'anglais (end-of-file, classification-based) et le français (va-t-il, c'est-à-dire, celui-ci). Il existe des langues, comme le japonais, qui n'utilisent pas de marqueurs pour séparer les mots (今年は本当に忙しかったです。).

Il existe deux approches pour la séparation des mots : par règles et statistique. L'approche par règles utilise principalement les expressions régulières en se basant sur les règles morphologiques. Elle peut aussi utiliser des listes des mots. Par exemple, dans une langue de type **Scriptio Continua** comme le japonais et le chinois, on peut utiliser un dictionnaire et comparer les mots en commençant par la fin en prenant la séquence la plus longue. L'approche statistique utilise un modèle de langage pour calculer la probabilité qu'un caractère marque la fin d'un mot. Les modèles de langages seront présentés dans le

chapitre suivant.

3 Normalisation du texte

Un texte peut contenir des variations du même terme. Dans des tâches qui se basent sur les statistiques sur des mots (comme la recherche d'information), on doit traiter ces variations comme un seul token. Même dans des tâches comme la compréhension de la langue, on a besoin de trouver des formes standards des dialectes (comme "ain't"). La transformation du texte à une forme canonique, en général, se fait en utilisant des expressions régulières pour chercher les variations et un dictionnaire pour chercher leurs formes canoniques. Parmi les variations qui ont besoin d'être normalisées, on peut citer :

- **Acronymes et les abréviations** : des fois, on trouve plusieurs variations d'une même abréviation. Dans ce cas, il faut choisir une seule variation pour les représenter. Par exemple, US → USA, U.S.A. → USA. Dans les tâches où on a besoin d'une compréhension plus approfondie, on doit chercher la version longue. Par exemple, M. → Monsieur.
- **Valeurs numériques** (dates et nombres) : selon la tâche, on doit unifier le format des valeurs numériques. Des fois, on a besoin de trouver la forme textuelle (1205 DZD → Mille deux cents cinq dinars algériens). Ceci est utile, par exemple, dans le cas de la synthèse de parole où l'appareil doit prononcer ce qui est écrit. Lorsque nous avons plusieurs sources de textes, on tombe sur plusieurs variations des dates. Dans ce cas, on peut utiliser le standard **ISO 8601** (12 Janvier 1986, 12.01.86 → 1986-01-12). Dans les tâches qui s'intéressent aux statistiques (Ex. nombre des dates dans un texte), on n'a pas besoin de garder les formes numériques. Dans ce cas, on peut garder seulement le type de ces formes (12 Janvier 1986 → DATE, kariminfo0@gmail.com → EMAIL).
- **Majuscules et Minuscules** : dans la plupart des cas, on n'a pas besoin de garder les mots en majuscules (Texte → texte). Mais, il faut faire attention lorsqu'on a besoin de garder cette information pour des tâches comme la segmentation du texte, ou pour différencier les nom propres (Will).
- **Diacritiques** : comme le point précédent, les diacritiques peuvent être supprimées lorsque la tâche ne dépend pas sur elles. Par exemple, dans le français, on appelle ça désaccentuation (système → systeme). Dans l'arable, on l'appelle dé-vocalisation (يَدرُسُ → يدرس). Dans des tâches qui ont besoin de vocalisation, comme le traitement des poèmes, on doit garder les diacritiques. En fait, on doit appliquer l'opération inverse (vocalisation du texte).
- **Contractions** : en général, on trouve ces formes beaucoup plus sur les réseaux sociaux. Mais, elles peuvent aussi être une règle standard de la langue. Par exemple, y'll → you all, s'il → si il. Dans le cas du dernier exemple, ça va aider dans la tâche de séparation des mots (tokenization).
- **Encodage** : il faut utiliser le même encodage supporté dans le traitement. L'encodage le plus célèbre est **UTF-8**.

4 Filtrage du texte

Le texte peut contenir des caractères, des mots et des expressions qui peuvent entraver son traitement. Pour faciliter ce dernier, il faut supprimer le bruit. Un exemple très commun est la présence des caractères spéciaux, comme les caractères non imprimables, dans le texte. Les mots contenant ces caractères ne sont pas considérés les mêmes que les mots sans ces caractères. En général, ces caractères sont d'origine des pages web ou des PDFs (extraction du texte à partir des PDFs ou autres formes d'images). Les mots clés des formats textuelles est un exemple des mots à filtrer. On peut trouver ça dans les balises de certaines formats semi-structurées comme HTML, XML, etc.

Les **mots vides** représentent les mots non significatifs comme les prépositions, articles et les pronoms.

La suppression des mots vides peut être utilisée si plusieurs mots dans le document ne contribuent pas particulièrement dans la description de son contenu. Afin d'augmenter la performance du système, plusieurs tâches (comme la recherche d'information et de résumé automatique) font appel à cette technique. Mais, on peut avoir des cas particuliers où cela peut causer des problèmes. Par exemple, la phrase "To be or not to be" sera totalement ignorée vu que tous ses mots sont des mots vides. Le nom propre "Will" peut être ignoré par confusion avec le verbe "to be" en future.

5 Morphologie

Nous avons vu dans le chapitre précédent qu'il existe pas mal de langues qui permettent la formation des mots en utilisant la flexion (ex. conjugaison) et la dérivation (ex. nominalisation). La formation des mots la plus utilisée est l'affixation en suivant certains règles. Par exemple, pour conjuguer le verbe "étudier" avec le pronom "nous" en présent, on supprime le suffixe "er" pour avoir le radical "étudu" et on ajoute le suffixe "ons". L'automatisation de cette tâche peut aider plusieurs applications, comme la génération du langage naturel (anglais : NLG). La tâche inverse consiste à trouver une forme standard des différentes variations ; c'est une technique de normalisation. Elle peut aider dans des tâches comme la recherche d'information et la compréhension du langage naturel (anglais : NLU).

5.1 Formation des mots

Dans les langues synthétiques, on peut former les mots en utilisant la flexion ou la dérivation. La flexion génère des variations morphologiques d'un mot selon les traits grammaticaux (nombre, genre, etc.). Les deux types de flexion sont la conjugaison des verbes et la déclinaison des noms, pronoms, adjectifs et déterminants. Quant à la dérivation, les mots créés forment un nouveau lexèmes (un sens différent du mot original) ou ils appartiennent à une autre catégorie grammaticale. Un exemple d'un nouveau lexème, couper → découper, عمل → إستعمل. Le changement de catégorie peut être dû à la nominalisation (classer → classement, classeur ; دَرَسَ → دَارِسٌ ، مُدَرِّسٌ ، مَدرَسَةٌ ، دَرسٌ) ou l'adjectif (fatiguer → fatigant), etc.

La formation des mots suit des règles bien définies, donc le problème peut être résolu en utilisant un automate à état fini. Bien sûr, il existe des cas spéciaux qui peuvent simplement être stockés dans un fichier. L'approche par règles est beaucoup plus utilisée dans ce cas, mais on peut trouver des recherches qui utilisent le niveau caractère pour appendre à générer des formes d'un mot donné, comme le projet MLConjug [1]. Personnellement, je voix que l'apprentissage automatique doit être utilisé pour résoudre les problèmes vraiment difficiles (en général, niveau syntaxique, sémantique, et pragmatique). Autre que l'approche statistique, plusieurs méthodes traditionnelles ont été utilisées pour la formation des mots. Dans le contexte du conjugaison automatique des verbes, on peut citer :

- **Base de données** : dans cette solution, on stocke tous les verbes dans une base de donnée avec les différentes variations possibles. Le point fort de cette solution est qu'on peut vérifier si un verbe appartient à la langue. Aussi, en arabe, les verbes peuvent avoir les mêmes lettres ; la seule différence est en vocalisation (diacritiques). Malgré ces points forts, cette solution est vraiment difficile à être implémentée ; on doit chercher tous les verbes et toutes les variations possibles.
- **Modèles (*template*)** : dans cette solution, on stocke les conjugaisons de certains verbes comme modèles et une autre liste de tous les verbes de la langue avec leurs modèles respectifs. C'est la forme la plus utilisée en français (par exemple, le verbe "sourire a comme modèle le verbe "rire"). Elle est similaire à la solution précédente, mais avec moins d'espace de stockage.
- **Règles** : dans cette solution, on utilise des règles SI-SINON et des expressions régulières. Un exemple

1. MLConjug : https://github.com/SekouD/mlconjug [visité le 2021-09-08]

de cette méthode peut être trouvé dans le projet Qutrub [2] pour la conjugaison automatique de l'arabe. Un autre exemple de ce type est celui de JsLingua [3] pour la conjugaison des verbes en arabe, anglais, français et japonais. L'avantage est qu'on n'a pas besoin de créer une base de données ; la solution est plus rapide. Mais, on doit gérer beaucoup de règles et aussi on a besoin d'une base de données si on veut vérifier le type du verbe (il peut y avoir des verbes confondus).

5.2 Réduction des formes

Il y a deux types réduits d'une forme d'un mot : radical et lemme. Le radical est un morphème qui peut ne pas être un mot du vocabulaire, or le lemme est un mot qui représente le lexème. La radicalisation (racinisation, en anglais : stemming) est l'opération de supprimer les affixes afin d'obtenir un radical (racine, en anglais : stem). Par exemple, chercher → cherch. Cette tâche est rapide et préférée dans des tâches comme la recherche d'information où on a besoin de plus de vitesse d'exécution même au détriment de l'exactitude. Quelques techniques pour la radicalisation sont les suivantes :

— **Base de données** : Stocker tous les termes et leurs racines dans une table.
— **Statistiques** : Utiliser un modèle de langage (comme les **N-Grammes**) pour estimer la position de troncation.
— **Règles** : Utiliser un ensemble de règles condition/action afin de détecter les affixes et les tronquer. L'algorithme le plus connu est celui de Porter (Porter et al., 1980) pour l'anglais. Il existe un framework très connu, appelé SnowBall [4], pour réaliser des racinateurs de ce genre. Dans ce framework, plusieurs conditions sont utilisées : sur la racine, sur l'affixe ou sur la règle. Une condition sur la racine peut être la longueur, la fin, si elle contient des voyelles, etc. Exemple, (*v*) Y → I : happy → happi, sky → sky. Une condition sur l'affixe peut être "il n'y a que le suffixe". Exemple, SSES → SS, ATIONAL → ATE. Une condition sur la règle peut être : la désactivation de certaines règles si une a été exécutée.

La lemmatisation (en anglais : lemmatization) cherche la forme canonique d'un mot appelée "lemme" (en anglais : lemma). Exemple, comprennent → comprendre, better → good Cette tâche est plus difficile que celle de radicalisation, puisqu'on a besoin du contexte du mot (Ex. saw → (V) see ou (N) saw).

— **Bases lexicales** : Ici, on utilise la radicalisation avec d'autres règles afin de chercher une forme dans une liste des lemmes possibles (un dictionnaire). Un exemple de ce type de lemmatisation est la lemmatisation morphy de Wordnet (voir l'algorithme 2.3).
— **Apprentissage automatique** : On essaye d'apprendre le lemme des mots en se basant sur des critères comme leurs catégories grammaticales. L'outil OpenNLP [5] implémente une version statistique de lemmatisation.

2. Qutrub : https://github.com/linuxscout/qutrub [visité le 2021-09-08]
3. JsLingua : https://github.com/kariminf/jslingua [visité le 2021-09-08]
4. SnowBall : https://snowballstem.org/ [visité le 2021-09-08]
5. OpenNLP lemmatisation : https://opennlp.apache.org/docs/1.8.0/manual/opennlp.html#tools.lemmatizer [visité le 2021-09-08]

Algorithme 2.3 : *Lemmatisation "morphy" de Wordnet*

Data : mot, catégorie
Result : liste des lemmes possibles
si *mot* ∈ *list_exceptions[catégorie]* **alors**
    return chercher_dans_dictionnaire({mot} ∪ list_exceptions[catégorie]) ;
fin
formes = {mot}
tant que *formes* ≠ ∅ **faire**
    formes = supprimer_les_affixes(formes, catégorie) ;
    résultats = chercher_dans_dictionnaire({mot} ∪ formes) ;
    si *résultats* ≠ ∅ **alors**
        return résultats ;
    fin
fin
return ∅ ;

Discussion

La morphologie est le niveau le plus simple dans le traitement d'un langage. Les tâches dans ce niveau sont beaucoup plus basées sur le caractère (graphème) qui est l'unité la plus basique dans le système d'écriture. La plupart des problèmes peuvent être résolus avec un automate à état fini ; d'où l'utilisation des expressions régulières. Ces dernières peuvent être utilisées pour rechercher des mots, extraire les phrases et les mots, extraire des parties du mot (racine), etc. Toutes ces tâches reviennent à la recherche d'un segment dans le texte. Pour une recherche approximative, des méthodes de comparaison dans le niveau caractère (distance d'édition) sont utilisées.

Le texte se compose des unités qui peuvent être des chapitres, des phrases, des mots ou des caractères. Un mot c'est l'unité la plus petite ayant un sens. Donc, afin de traiter un texte, il faut le décomposer en petites unités afin de traiter les unités plus grandes, etc. D'où l'importance de la segmentation du texte. Ces unités peuvent avoir le même sens, mais plusieurs variations. Dans la compréhension du texte, on essaye de représenter le texte d'une manière plus abstraite. Donc, une variation peut être représentée par un mot représentant plus des caractéristiques comme le genre, le nombre, etc. Quelques mots doivent être filtrés puisqu'ils sont considérés comme du bruit.

Certes, les tâches de ce niveau sont vraiment simples. Mais, elles sont vraiment primordiales pour la réussite des tâches plus évoluées. Ces tâches sont vraiment dépendantes à la langue traitée. Tellement elles sont simples, on peut implémenter des outils pour n'importe quelle langue en sachant les règles morphologiques. Malheureusement, pas toutes les langues fournissent de tels outils. C'est la responsabilité des gens qui parlent ces langues à les développer dans un monde de plus en plus numérique.

CHAPITRE 3

MODÈLES DE LANGAGE

LES LANGUES

Si on veut tester qu'un langage soit bien écrit, on doit revenir aux règles de composition de ses phrases. Un langage se compose d'un vocabulaire et une grammaire pour construire les phrases. Par exemple, on peut trouver un syntagme nominale suivi par un verbe suivi par un syntagme nominale si le verbe est transitif. En statistique, ces règles peuvent être vues comme des probabilités d'apparition d'un mot sachant qu'un ou plusieurs mots soient apparus avant. Les probabilités estimées à partir d'un corpus d'entrainement sont appelées un modèle de langage. Ce dernier est utile pour estimer le mot suivant sachant une liste de mots passés et aussi pour calculer la probabilité qu'une phrase soit juste. Dans ce chapitre, on va présenter les modèles de langage traditionnels (N-Grammes), ainsi que ceux basés sur les réseaux de neurones.

Une phrase est bien définie si sa probabilité $P(S) = P(w_1, w_2, ..., w_n)$ égale à 1. Une phrase avec une probabilité plus grande qu'une autre a plus de chance de se produire étant donné le contexte qu'on ait appris. Cette hypothèse a plusieurs applications :

- Traduction automatique : en traduisant un texte d'une langue vers une autre, on peut trouver des mots avec plusieurs traductions/sens. En plus, l'ordre des mots n'est pas toujours symétrique. En utilisant un modèle de langage de la langue destinataire, on peut vérifier le choix et l'ordre les plus probables. Exemple, My tall brother → P(Mon grand frère) > P(Mon haut frère).
- Correction des fautes grammaticales : du même, en utilisant un modèle de langage, on peut vérifier qu'un mot n'ait pas de chance de se produire après une séquence de mots. En calculant, par exemple, les différentes probabilités des phrases avec les mots similaires (en terme de distance d'édition) avec le mot erroné, on peut suggérer des corrections. Exemple, P(Un objet qu'on puisse emporter) > P(Un objet qu'ont puisse emporter).
- Reconnaissance de paroles : lorsqu'on transforme les paroles en texte, le programme peut mélanger entre les mots ou les expressions proches en prononciation. On peut aider le choix des mots en utilisant un modèle de langage. Exemple, P(Jeudi matin) > P(Je dis matin)

En plus de la probabilité d'une phrase, on peut aussi estimer la probabilité d'occurrence d'un mot en se basant sur les mots précédents $P(w_n|w_1, \ldots, w_{n-1})$. Estimer cette probabilité a plusieurs applications :

- Auto-complétion : en se basant sur les mots déjà introduits par l'utilisateur, le programme estime la probabilité de chaque mot du vocabulaire et affiche ceux avec une grande probabilité conditionnelle. Exemple, P(traitement automatique de l'information) > P(traitement automatique de l'eau).
- Génération automatique de textes : en utilisant une représentation interne, on essaye de générer le texte mot par mot. Le programme utilise cette représentation et un modèle de langage pour apprendre la génération.

1 Modèle N-gramme

La probabilité d'occurrence d'une phrase est calculée en utilisant la formule des probabilités composées exprimée dans l'équation 3.1. Par exemple, P(*je travaille à l'ESI*) = P(je) P(travaille | je) P(à | je travaille) ... P(ESI | je travaille à l'). Afin d'estimer les probabilités conditionnelles, on utilise le maximum de vraisemblance (qui va être présenté juste après).

$$P(w_1 \dots w_m) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \dots P(w_m|w_1, \dots, w_{m-1}) \tag{3.1}$$

Dans un langage naturel, on peut trouver plusieurs phrases possibles ; on peut même générer une infinité de phrases. Pour estimer une probabilité conditionnelle avec un historique long (mots précédents), on doit utiliser un corpus (dataset textuel) très grand. Vu que les phrases sont infinies, on ne peut pas représenter toutes les combinaisons possibles. Dans ce cas, une solution est de limiter la taille de l'historique. Donc, on essaye d'estimer $P(w_i|w_{i-n+1}, \dots, w_{i-1})$ avec un historique de $n-1$ mots ; ce modèle est appelé **N-gramme**.

1.1 Formulation

Étant donné un processus stochastique, ceci vérifie la propriété de Markov si l'état futur ne dépend que sur l'état présent. Donc, la probabilité d'occurrence d'un mot w_i sachant l'occurrence de plusieurs mots $w_1, \dots, w_{i-1}$ ne dépend que sur le mot précédent w_{i-1} suivant la propriété de Markov (voir l'équation 3.2). Ce modèle est appelé un modèle **Bi-gramme**. Ce modèle peut être représenté graphiquement comme un automate à état fini où les mots sont représentés comme des états et les transitions sont représentées par des probabilités.

$$P(w_i|w_1, \dots, w_{i-1}) = P(w_i|w_{i-1}) \tag{3.2}$$

Ce modèle peut être généralisé en considérant $n-1$ mots précédents. Dans ce cas, l'équation 3.2 sera formulée comme l'équation 3.3. Le modèle généralisé est appelé **N-gramme**. Les modèles les plus utilisés sont : Uni-gramme ($n=1$), Bi-gramme ($n=2$) et Tri-gramme ($n=3$). Une compilation des différents N-grammes préparée par Google et distribuée sous une licence open source sous le nom : Google Books Ngram [1]

$$P(w_i|w_1, \dots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \dots, w_{i-1}) \tag{3.3}$$

Donc, l'équation 3.1 qui calcule la probabilité d'une phrase sera reformulée comme indiqué dans l'équation 3.4 en utilisant un modèle **N-gramme**.

$$P(w_1 \dots w_m) \approx \prod_{i=1}^{m} P(w_i|w_{i-n+1}, \dots, w_{i-1}) \tag{3.4}$$

Étant donné une fonction $C(S)$ qui compte le nombre d'occurrences d'une séquence S dans un corpus, la probabilité conditionnelle est calculée selon le maximum de vraisemblance (voir l'équation 3.5). Le corpus d'entraînement doit être suffisant afin de capturer toutes les combinaisons possibles. En fait, plus n est grand, plus on aura besoin de données. En observant la formule, on se demande : comment peut-on calculer la probabilité conditionnelle des mots du début et de fin ? On marque le début et la fin des phrases

1. Google Books Ngram : https://storage.googleapis.com/books/ngrams/books/datasetsv3.html [visité le 2021-09-25]

par <s> et </s> respectivement (une fois pour les bi-grammes, 2 fois pour les tri-grammes, etc.). Comme ça, on peut exprimer la probabilité qu'un mot soit au début ou à la fin de la phrase.

$$P(w_i|w_{i-n+1},\ldots,w_{i-1}) = \frac{C(w_{i-n+1}\ldots w_{i-1}w_i)}{\sum_j C(w_{j-n+1}\ldots w_{j-1}w_j)} = \frac{C(w_{i-n+1}\ldots w_{i-1}w_i)}{C(w_{i-n+1}\ldots w_{i-1})} \tag{3.5}$$

Supposons, nous avons un corpus d'entraînement avec 4 phrases. Si on veut utiliser un modèle **Bi-gramme**, on doit encercler chaque phrase par "<s>" et "</s>". Voici l'ensemble des phrases :

- <s>un ordianteur peut vous aider </s>
- <s>il veut vous aider </s>
- <s>il veut un ordinateur </s>
- <s>il peut nager </s>

Dans ce modèle, on calcule la probabilité d'occurrence de chaque mot sachant le mot qui le précède. Par exemple, $P(peut|il) = \frac{C(il\ peut)}{C(il)} = \frac{1}{3}$. Dans ce cas, la probabilité d'occurrence de la phrase "il peut vous aider" peut être estimée comme suit :

$$P(\text{<s>}il\ peut\ vous\ aider\text{</s>}) = \underbrace{P(il|\text{<s>})}_{3/4}\underbrace{P(peut|il)}_{1/3}\underbrace{P(vous|peut)}_{1/2}\underbrace{P(aider|vous)}_{2/2}\underbrace{P(\text{</s>}|aider)}_{2/2} = \frac{1}{8}$$

Si, on essaye d'estimer la probabilité d'occurrence de la phrase "il peut aider", on aura 0. Même si la phrase est juste, nous n'avons pas suffisamment de données pour entraîner notre modèle à capturer cette forme. La probabilité sera calculée comme suit :

$$P(\text{<s>}il\ peut\ aider\text{</s>}) = \underbrace{P(il|\text{<s>})}_{3/4}\underbrace{P(peut|il)}_{1/3}\underbrace{P(aider|peut)}_{0/2}\underbrace{P(\text{</s>}|aider)}_{2/2} = 0$$

Maintenant, on essaye d'estimer la probabilité d'occurrence de la phrase : "il peut nous aider" sachant que le mot "nous" n'existe pas dans le vocabulaire. La probabilité peut être estimée comme suit :

$$P(\text{<s>}il\ peut\ vous\ aider\text{</s>}) = \underbrace{P(il|\text{<s>})}_{3/4}\underbrace{P(peut|il)}_{1/3}\underbrace{P(nous|peut)}_{0/0}\underbrace{P(aider|nous)}_{0/2}\underbrace{P(\text{</s>}|aider)}_{2/2} = NaN$$

Les mots qui ne figurent pas dans le vocabulaire sont appelés "Out-of-vocabulary words". Afin de régler ce problème, on peut ajouter un autre mot "<UNK>" au vocabulaire. Pour incorporer ce mot dans le corpus d'entraînement, on peut fixer un vocabulaire et remplacer le reste des mots par ce mot clé, ou en peut remplacer les mots les moins fréquents par "<UNK>".

1.2 Lissage (Smoothing)

Dans l'exemple précédent, nous avons vu le problème des mots qui n'appartiennent pas au vocabulaire (Ex. "nous") qui résultent à une division par 0. Ce dernier problème peut être résolu en réservant un mot clé pour les mots absents (comme présenté précédemment). Le problème des n-grammes absents (Ex. "peut aider"), qui résultent à une probabilité de 0, ne peut pas être résolu par cette technique. Une solution de ce problème est d'ajouter plus de données. Même sans ce problème, si on utilise des petits N-grammes, on peut ne pas capturer les dépendances à long terme. Exemple, "L'ordinateur que j'ai utilisé hier à l'ESI pendant la séance du cours a planté". Donc, nous sommes obligés à utiliser des N-grammes plus grands. Pour entraîner un tel modèle, il nous faut une grande quantité de données. Pour une vocabulaire de taille V et N-gramme de taille N, on aura V^N n-grammes possibles. Une solution de tous ça est d'utiliser la technique de lissage. L'intuition est d'emprunter une petite portion des probabilités des N-grammes existants pour former une probabilité aux N-grammes absents.

Lissage de Lidstone/Laplace

Supposons que notre modèle de langage supporte un vocabulaire de taille V et n grammes. Afin de soustraire une petite probabilité de chaque N-gramme existant et créer une probabilité pour les N-grammes non existants, on peut modifier la formule du maximum de vraisemblance. On utilise un paramètre de lissage α comme indiqué par l'équation 3.6. Dans le cas général, ceci est appelé "lissage de Lidstone". Lorsque $\alpha = 1$, il est appelé "lissage de Laplace". Si $\alpha = 0.5$, il est appelé "loi de Jeffreys-Perks".

$$P(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{C(w_{i-n+1} \ldots w_{i-1} w_i) + \alpha}{C(w_{i-n+1} \ldots w_{i-1}) + \alpha V} \tag{3.6}$$

Prenons l'exemple précédent, le vocabulaire est de taille 8 ($8^2 = 64$ bi-grammes possibles). Si, on essaye d'estimer la probabilité d'occurrence de la phrase "il peut aider", on n'aura pas 0 même si la séquence "peut aider" ne figure pas dans le corpus. La probabilité de cette phrase en utilisant un modèle Bi-gramme et un lissage de Laplace sera calculée comme suit :

$$P(\text{<s>}il\ peut\ aider\text{</s>}) = \underbrace{P(il|\text{<s>})}_{(3+1)/(4+8)}\ \underbrace{P(peut|il)}_{(1+1)/(3+8)}\ \underbrace{P(aider|peut)}_{(0+1)/(2+8)}\ \underbrace{P(\text{</s>}|aider)}_{(2+1)/(2+8)}$$

On revient à la phrase : "il peut nous aider" où le mot "nous" n'existe pas dans le vocabulaire. On teste si on peut résoudre le problème avec le lissage de Laplace ; sans utiliser la solution du mot clé réservé pour les mots inconnus. La probabilité en utilisant un modèle Bi-gramme peut être estimée comme suit :

$$P(\text{<s>}il\ peut\ vous\ aider\text{</s>}) = \underbrace{P(il|\text{<s>})}_{(3+1)/(4+8)}\ \underbrace{P(peut|il)}_{(1+1)/(3+8)}\ \underbrace{P(nous|peut)}_{(0+1)/(0+8)}\ \underbrace{P(aider|nous)}_{(0+1)/(2+8)}\ \underbrace{P(\text{</s>}|aider)}_{(2+1)/(2+8)}$$

Interpolation

L'idée de l'interpolation est d'entraîner n modèles de langage : de n-grammes jusqu'à uni-gramme. La probabilité de l'interpolation P_I sera calculée en utilisant une composition linéaire entre les probabilités des différents modèles. Les paramètres de composition λ_j seront estimés en utilisant un corpus de réglage avec la condition $\sum_j \lambda_j = 1$. Dans l'entraînement, on essaye de maximiser la probabilité d'interpolation sur ce corpus. La probabilité d'interpolation pour un modèle Tri-gramme peut être calculée en utilisant l'équation 3.7.

$$P_I(w_i|w_{i-2}w_{i-1}) = \lambda_3 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_1 P(w_i) \tag{3.7}$$

Toujours avec l'exemple précédent, on essaye d'estimer la probabilité d'occurrence de la phrase "il peut aider" en utilisant l'interpolation Tri-grammes. Tout d'abord, on calcul la probabilité en utilisant chaque modèle n-gramme.

- Uni-gramme : $P_1(\text{<s>}il\ peut\ aider\text{</s>}) = \underbrace{P(il)}_{3/16}\ \underbrace{P(peut)}_{2/16}\ \underbrace{P(aider)}_{2/16} \approx 0.003$
- Bi-gramme : déjà calculé ; $P_2(\text{<s>}il\ peut\ aider\text{</s>}) = 0$.
- Tri-gramme : Si la probabilité de cette phrase en utilisant le Bi-gramme est nulle, donc elle l'est aussi avec des modèles de l'ordre supérieure. $P_3(\text{<s><s>}il\ peut\ aider\text{</s></s>}) = 0$. Ici, on peut tomber sur la division par zéro, mais on va considérer la probabilité comme nulle.

Si on utilise les paramètres $\lambda_3 = 0.7$, $\lambda_2 = 0.2$, $\lambda_1 = 0.1$, la probabilité de l'interpolation sera $P_I = 0.7P_3 + 0.2P_2 + 0.1P_1 = 0.1P_1 \approx 0.0003$. Bien sûr, cette solution ne résout pas le problème des mots absents.

Back-off de Katz

L'idée est d'utiliser la probabilité de l'ordre supérieure n si le n-gramme existe dans le corpus d'entraînement. Sinon, on passe à l'ordre suivant $n-1$. Pour maintenir une distribution correcte des probabilités, on doit réduire la probabilité de l'ordre supérieure pour avoir une probabilité réduite P^*. La réduction sera distribuée sur les probabilités des N-grammes de l'ordre inférieur en utilisant une fonction α à base du contexte. La formule pour calculer la probabilité P_{BO} du Back-off de Katz est indiquée dans l'équation 3.8.

$$P_{BO}(w_i|w_{i-n+1},\ldots,w_{i-1}) = \begin{cases} P^*(w_i|w_{i-n+1},\ldots,w_{i-1}) & \text{si } C(w_{i-n+1},\ldots,w_{i-1}w_i) > 0 \\ \alpha(w_{i-n+1},\ldots,w_{i-1})P_{BO}(w_{i-n+2},\ldots,w_{i-1}) & \text{sinon} \end{cases} \tag{3.8}$$

2 Modèles neuronaux

Comme nous avons vu, les **N-grammes** ont besoin d'utiliser le lissage afin de prendre en considération des combinaisons absentes dans le corpus d'entrainement. Ce modèle ne peut pas capturer les mots similaires (utilisés dans le même contexte ; Ex. synonymes). En plus, il ne supporte pas des contextes avec un historique long. Les réseaux de neurones semblent avantageux considérant ces problèmes. Ils n'ont pas besoin de lissage puisqu'ils généralisent mieux ; ils peuvent donner une petite probabilité aux n-grammes absents. Aussi, ils peuvent apprendre des représentations proches pour les mots similaires (qui sont dans le même contexte). A cause de leur capacité à généraliser, ils peuvent supporter un contexte plus grand. Cependant, ces modèles sont plus lents à entraîner.

2.1 Réseau de neurones à propagation avant

Un réseau de neurones à propagation avant est la forme traditionnelle : une couche d'entrée, des couches cachées et une couche de sortie. L'idée est de choisir le nombre des n-grammes n. La couche d'entrée est alimentée par les $n-1$ mots précédents afin d'estimer le mot n dans la couche de sortie. Chaque mot est représenté sous forme d'un vecteur de taille V (taille de vocabulaire) où toutes les positions ont un zéro sauf la position réservée pour ce mot. Cette représentation est appelée "**One-Hot** représentation". En sortie, on aura un vecteur avec des probabilités ; le mot dont la position ait la plus grande probabilité est celui choisi.

Ici, on va présenter le modèle de Bengio et al. (2003) illustré par la figure 3.1. Après choisir la taille n du modèle, les mots $w_{i-n+1},\ldots,w_{i-1}$ sont représentés sous forme des vecteurs **One-Hot** $h_{i-n+1},\ldots,h_{i-1}$. Chacun de ces vecteur est passé par une couche cachée de taille d ; donc on aura un vecteur de taille d pour chaque mots de ces $n-1$. Ce vecteur est appelé **embedding** (sera discuté en détail dans le chapitre de sémantique lexicale). En fusionnant les vecteurs, on aura un seul vecteur de contexte $m \in \mathbb{R}^{(n-1)d}$. Ce vecteur va être passé par deux blocs en parallèle :

- Une couche cachée avec les paramètres $A \in \mathbb{R}^{(n-1)\times d\times V}$ (poids) et $b \in \mathbb{R}^V$. La somme pondérée résulte à un vecteur de taille V.
- Une couche cachée avec les paramètres $T \in \mathbb{R}^{(n-1)\times d\times H}$ suivie par une fonction "Tanh", ce qui va générer un vecteur de taille H. Ce dernier passe par une autre couche avec les paramètres $W \in \mathbb{R}^{V\times H}$, ce qui génère un vecteur de taille V.

Les deux vecteurs sont additionnés élément par élément pour avoir un seule vecteur de taille V. Pour avoir des probabilités avec une somme égale à 1, on passe ce vecteur par une fonction "softmax". L'architecture

de ce modèle peut être exprimé par l'équation 3.9.

$$P(.|h_1, \ldots, h_{n-1}) = Softmax((b + mA) + W \tanh(u + mT)) \tag{3.9}$$

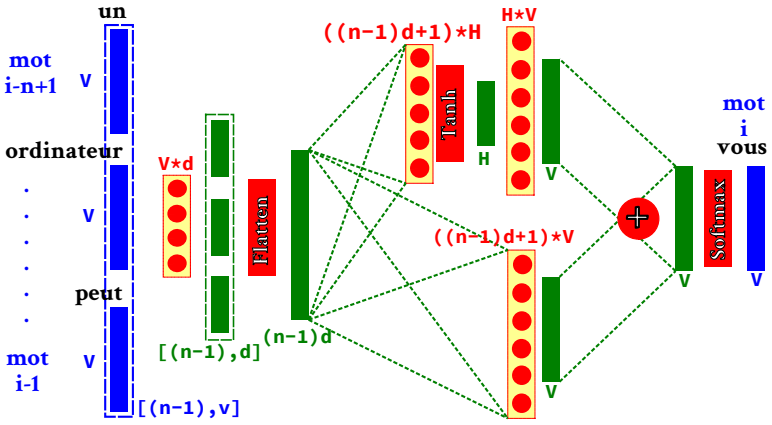


Figure 3.1 : *Représentation du modèle proposé par Bengio et al. (2003)*

2.2 Réseaux de neurones récurrents

Certes, les modèles basés sur les réseaux de neurones à propagation avant sont avantageux par rapport aux **N-grammes**. Mais, ils ne supportent pas des longueurs variables des contextes. Par contre, les réseaux de neurones récurrents ont la capacité d'estimer un mot dans la position i sachant tous les mots passés. Ici, on va présenter le modèle de Mikolov et al. (2010) qui se base sur le réseau de Elman.

La figure 3.2 représente un exemple d'exécution du modèle proposé par Mikolov et al. (2010). Le réseau récurrent a une couche d'entrée x, une couche cachée s (l'état) avec la fonction "sigmoid" et une couche de sortie y avec une fonction "softmax". Dans un instant t, l'entrée x est composée du mot $w_t \in \mathbb{N}^V$ encodé en utilisant One-Hot et le contexte précédent $s_{t-1} \in \mathbb{R}^H$ (équation 3.10). L'entrée x_t est passée par la couche cachée ayant des paramètres $W \in \mathbb{R}^{(H+V)\times H}$ pour avoir un nouveau contexte s_t (équation 3.11). Le contexte s_t est passé vers l'état suivant $t+1$ et il est utilisé pour estimer le mot suivant dans l'état actuel. Pour ce faire, il passe par une couche de sortie ayant des paramètres $U \in \mathbb{R}^{H\times V}$, ce qui génère un vecteur de taille V qui est passé par une fonction "softmax" (équation 3.12).

$$x_t = s_{t-1} \bullet m_t \tag{3.10}$$

$$s_t = \sigma(x_t W) \tag{3.11}$$

$$y_t = softmax(s_t U) \tag{3.12}$$

Il ne faut pas oublier d'utiliser le mot clé "<UNK>" afin d'entraîner le modèle à prendre en considération les mots inconnus. Le problème avec cette architecture est que le modèle puisse arrêter à apprendre avec un contexte à long terme (problème de disparition des gradients). Une solution est d'utiliser des architectures plus avancées comme **LSTM** et **GRU**. Ceci est dit, le problème de disparition des gradients existe toujours. Une solution technique est de fixer un nombre maximum des états.

3 Évaluation

Qu'est ce que rend un modèle de langage plus bon qu'un autre ? Il faut avoir une méthode pour comparer les deux. Pour ce faire, il existe deux approches :

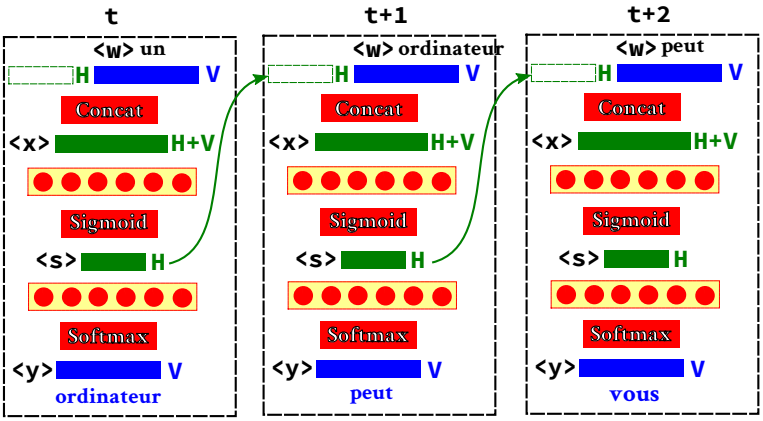


Figure 3.2 : *Modèle de langage à base des réseaux de neurones récurrents proposé par Mikolov et al. (2010)*

— **Évaluation extrinsèque** : ici, on veut tester l'effet d'un modèle de langage sur une autre tâche. On évalue la tâche en utilisant plusieurs modèles de langue pour choisir celui qui donne des meilleurs résultats. Exemple, "La qualité de traduction automatique en utilisant ce modèle". Dans cet exemple, on essaye de choisir le modèle de langue le plus adéquat pour la tâche de traduction automatique. Bien sûr, l'évaluation dans ce cas est vraiment couteuse vu qu'on va entraîner le système de traduction à chaque fois qu'on modifie le modèle de langage.

— **Évaluation intrinsèque** : ici, on veut tester la capacité du modèle à représenter le langage. Étant donné deux modèles de langage entraînés sur le même corpus d'entraînement, on utilise un autre de test pour tester la capacité de représentation. La méthode la plus utilisée dans ce cas est **la perplexité**. Il faut savoir que le fait d'être un modèle représentatif ne garantit pas d'avoir une bonne performance dans une tâche donnée.

La perplexité est une mesure intrinsèque qui vise à tester la qualité de prédiction d'un modèle sur un corpus de test (non vu par ce modèle). Étant donné un corpus de test avec la taille N, on ajoute les marqueurs de début et de fin pour toutes les phrases ; La perplexité traite le corpus comme étant une seule chaîne. Le but est de calculer la probabilité d'occurrence du texte (la totalité) en utilisant le modèle de langage. Cette probabilité est inversée puis passée par une racine d'ordre N comme indiqué dans l'équation 3.13. Dans ce cas, un modèle ayant une perplexité minimale est le meilleur.

$$\begin{aligned} PP(w) &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \dots w_{i-1})}} \end{aligned} \tag{3.13}$$

Discussion

Un langage se définit un vocabulaire et une grammaire afin de composer des phrases. Les règles de la grammaire sont définies par des linguistes. Des fois, il est vraiment difficile de définir ces règles, surtout si elles se basent sur des exceptions des mots. Par exemple, un verbe transitif qui n'accepte pas des mots comme complément d'objet directe puisque la phrase n'aura aucun sens (J'ai mangé le ciel). Ce dernier exemple peut être réglé si nous avons des statistiques sur le contexte des mots dans un langage : quel mot peut se produire en voisinage d'un autre ? et à quel fréquence ? Cette représentation est appelée : un modèle de langage ; déjà présenté dans ce chapitre.

Nous avons vu que le modèle de langage se base sur le vocabulaire appris à partir du corpus d'entraî-

nement. Ce que nous n'avons pas discuté est le sens du vocabulaire : que veut-on dire par vocabulaire ? Nous avons vu dans le chapitre précédent qu'on puisse générer des mots à partir d'autres (par exemple, la conjugaison). En réalité, on peut entraîner le modèle avec toutes les formes possibles. Comme ça, on peut représenter le fait que le mot "étudiez" ne peut pas venir après le mot "je". En quelque sorte, on essaye d'apprendre la syntaxe et la lexique en parallèle. Mais, ceci pose un problème dans les langues fortement flexionnelles ; on aura un vocabulaire gigantesque. Théoriquement, ceci ne pose pas de problème puisque le vocabulaire reste toujours un ensemble fini (mis à part l'évolution de la langue : ajout de mots à la lexique). Mais pratiquement, plus le vocabulaire est grand, plus la tâche est couteuse. Dans ce cas, le traitement prend plus de temps supposant qu'on ait suffisamment de mémoire pour représenter tous les mots. Aussi, on doit avoir un grand corpus afin de capturer toutes les variations morphologiques de tous les mots. Une des solutions est d'appliquer une sorte de radicalisation, en séparant la racine et le suffixe. Tous les deux seront considérés comme des mots à part. Comme ça, on réduit la taille du vocabulaire et on apprend la formation des mots au même temps.

Dans ce chapitre, nous avons présenté les modèles de langage en prenant les mots comme unité. En réalité, les modèles de langage peuvent être entraînés sur des caractères. Parmi les applications de ce genre de modèles est la détection des langues, surtout pour celles qui utilisent le même système d'écriture. Supposons qu'on veut détecter les langues : français, anglais et espagnol. Dans ce cas, on entraîne trois modèles de langage pour chacune de ces langues. Étant donné une phrase, on essaye d'estimer les trois probabilités (niveau caractère) et considérer le modèle qui maximise la probabilité. Un modèle de langage peut être entrainé sur des séquences d'ADN. Dans ce cas, le vocabulaire sera : "A", "T", "C", "G" et "U". Parmi ses applications : la détection de l'espèce.

CHAPITRE 4

ÉTIQUETAGE MORPHO-SYNTAXIQUE

LES LANGUES

Les catégories grammaticales des mots représentent une caractéristique nécessaire pour comprendre un texte. Des fois, des mots peuvent avoir plusieurs catégories grammaticales (natures). Par exemple, le mot "nouveau" dans la phrase "C'est l'anniversaire de mon nouveau[NOM]" n'a pas la même nature que celui dans la phrase "Mon nouveau[ADJ] cours est presque terminé". Cette propriété peut être aperçue beaucoup plus dans l'anglais où les noms peuvent être transformés en verbes sans changement (Ex. "fish"). Cette tâche est utile si on veut appliquer des statistiques sur les catégories grammaticales afin d'améliorer une autre tâche comme celle de classification des textes. Dans ce chapitre, on va présenter la tâche d'étiquetage des séquences en général et l'étiquetage morpho-syntaxique en particulier.

Savoir les catégories grammaticales des mots est un pas vers la compréhension d'une phrase. Prenons un exemple d'une phrase en anglais : "We can can the can". Le mot "can" a plusieurs sens : (1-verbe) pouvoir (2-verbe) mettre en boîte (3-nom) boîte. Après étiquetage morpho-syntaxique, on aura "We[pronom] can[verbe] can[verbe] the[déterminant] can[nom]". Ceci n'est pas seulement utile pour savoir le sens du mot, mais aussi on peut voir la structure de la phrase. Essayez d'étiqueter la phrase suivante : "Will Will will the will to Will?". L'étiquetage morpho-syntaxique est bénéfique pour plusieurs tâches :

- c'est une étape préliminaire pour l'analyse syntaxique (chapitre suivant).
- les catégories grammaticales peuvent être considérée comme une caractéristique importante dans des tâche de classification des textes. Par exemple, la tâche de détection de l'auteur peut bénéficier de cette caractéristiques ; certains auteurs utilisent plus de noms dans les phrases que d'autres.

1 Étiquetage de séquences

Un langage est une séquence continue des mots ; il est considéré comme un phénomène temporel. Chaque mot ou ensemble de mots possèdent des caractéristiques linguistiques, chacune définit plusieurs catégories. Par exemple, les mots ont des catégories lexicales : nom, verbe, adjectif, etc. En général, la catégorie d'un mot est dépendante aux catégories des mots avant. L'étiquetage d'une séquence des mots $w = w_1, \ldots, w_n$ sert à trouver une séquence des étiquettes équivalente $t = t_1, \ldots, t_n$.

L'étiquetage de séquences peut être réalisé en utilisant des règles définies manuellement. Une des méthodes consiste à assigner à chaque mot un ensemble des étiquettes possibles. Ensuite, on utilise des règles afin de réduire les étiquettes jusqu'à atteindre une étiquette par mot. Parmi les règles qu'on peut utiliser : Un déterminant est suivi par un nom. L'approche statistique vise à apprendre l'annotation en se basant sur un corpus annoté manuellement. Le problème de l'étiquetage revient à maximiser la probabilité condi-

tionnelle d'une séquence des étiquettes sachant une séquence des mots (voire l'équation 4.1).

$$\hat{t} = \arg\max_{t} P(t|w) \tag{4.1}$$

Dans les statistiques, il existe deux types de modèles : génératif et discriminatif. Un modèle génératif essaye d'apprendre la génération de l'entrée à partir de la sortie ; étant donné une étiquette, quel est la probabilité de chaque mot du vocabulaire ? En utilisant la théorème de Bayes, l'équation 4.1 revient à résoudre l'équation 4.2. Le modèle de Markov caché, en anglais Hidden Markov Model (HMM), est un exemple d'un modèle génératif.

$$\hat{t} = \arg\max_{t} P(t)P(w|t) \tag{4.2}$$

Contrairement aux modèles génératifs, un modèle discriminatif estime la probabilité $P(t|w)$ directement. Ceci revient à apprendre l'étiquette d'un mot en se basant sur ces caractéristiques en plus des caractéristiques d'un nombre limité des mots avant. Un modèle qui suit cette méthode est Maximum Entropy Markov Model (MEMM). Une autre méthode est d'utiliser les réseaux de neurones récurrents qui supportent l'aspect temporel.

L'étiquetage morpho-syntaxique est parmi les tâches appartenant à l'étiquetage des séquences. Son but est de trouver les catégories grammaticales des mots (nom, verbe, etc.) d'une phrase. Des fois, on veut détailler les catégories afin d'avoir plus de précision. Par exemple, un nom peut être un nom propre ou un nom impropre. Un exemple d'une phrase en anglais étiquetée morpho-syntaxiquement est illustré dans la figure 4.1.

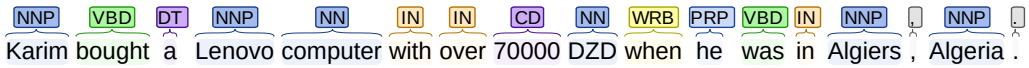


Figure 4.1 : *Exemple d'étiquetage morpho-syntaxique par https://corenlp.run/ [visité le 2021-09-25]*

Une autre tâche de ce type est la reconnaissance d'entités nommées. Elle vise à trouver les personnes, les organisations, les places, les nombres, etc. dans une phrase. Contrairement à l'étiquetage morpho-syntaxique, une catégorie peut être affectée à une séquence de mots et pas un seul. Par exemple, l'expression "école nationale supérieure d'informatique" a la catégorie "organisation". Mais, comment exprimer que tous ces mots forment une seule catégorie et pas chaque mot à part ? La méthode utilisée pour ça est le format **IOB** (Inside, Outside, Beginning). Les mots qui n'appartiennent à aucune catégorie sont annotés par "O". Les mots qui marquent le commencement d'une catégorie sont annotés par "B-" suivi par le nom de la catégorie. Les mots du milieu sont annotés par "I-" suivi par le nom de la catégorie. Donc, l'expression précédente sera annotée comme "école[B-ORG] nationale[I-ORG] supérieure[I-ORG] d'informatique[I-ORG]". La figure 4.2 représente un exemple d'une phrase en anglais contenant des entités nommées.

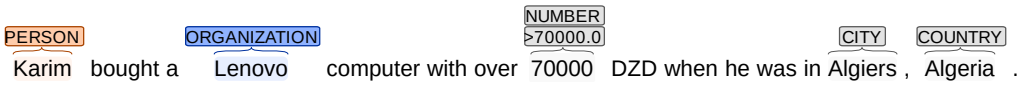


Figure 4.2 : *Exemple de la reconnaissance d'entités nommées par https://corenlp.run/ [visité le 2021-09-25]*

L'analyse syntaxique de surface (Chunking) est un autre exemple d'étiquetage des séquences. Elle sert à trouver les syntagmes d'une phrase sans structure d'arbre. Donc, la catégorie est affectée à une séquence et pas seulement un seule mot.

```
Battle-tested/JJ Japanese/JJ industrial/JJ managers/NNS
here/RB always/RB buck/VBP up/RP nervous/JJ newcomers/NNS
with/IN the/DT tale/NN of/IN the/DT first/JJ of/IN
their/PP$ countrymen/NNS to/TO visit/VB Mexico/NNP ,/,
a/DT boatload/NN of/IN samurai/FW warriors/NNS blown/VBN
ashore/RB 375/CD years/NNS ago/RB ./.
```

Figure 4.3 : *Exemple d'un texte annoté de Penn TreeBank (Taylor et al., 2003)*

2 Ressources pour l'étiquetage morpho-syntaxique

Comme mentionné avant, cette tâche sert à trouver la catégorie grammaticale de chaque mot dans une phrase. Une catégorie grammaticale peut avoir plusieurs noms (selon l'ouvrage) : nature, catégorie lexicale, classe grammaticale, espèce grammaticale, partie du discours (en anglais, Part of speech). Afin de réaliser cette tâche, il faut tout d'abord définir les étiquettes (tags). Est-ce qu'on détaille dans les catégories ou on se contente des catégories principales ? Les étiquettes les plus fameuses sont celles de Penn **TreeBank** pour l'anglais : conjonction de coordination (CC), nombre cardinal (CD), déterminant (DT), etc. Les catégories grammaticales sont différentes d'une langue à une autre. Mais, on peut trouver certaines catégories qui sont partagées. Le projet "Universal dependencies" cherche à normaliser ces catégories comme indiqué dans le tableau 4.1.

Classe ouverte	Classe fermée	Autres
ADJ : adjectif	**ADP** : adposition	**PUNCT** : ponctuation
ADV : adverbe	**AUX** : auxiliaire	**SYM** : symbole
INTJ : interjection	**CCONJ** : conjonction de coordination	**X** : Autres
NOUN : nom	**DET** : déterminant	
PROPN : nom propre	**NUM** : numérique	
VERB : verbe	**PART** : particule	
	PRON : pronom	
	SCONJ : conjonction de subordination	

Tableau 4.1 : *Catégories grammaticales universelles selon https://universaldependencies.org/u/pos/ [visité le 2021-09-25]*

L'étape suivante est d'annoter un corpus manuellement. Ce type de corpus est appelé **TreeBank**. Parmi les corpus annotés, on peut citer Penn Treebank pour l'anglais (Exemple dans la figure 4.3). Un autre projet riche en terme de langues traitées est Universal TreeBank [1] (Petrov et al., 2012). C'est un projet à communauté ouverte qui vise à utiliser les mêmes classes grammaticales afin d'annoter plusieurs langues. Mais, il faut faire attention lorsqu'on fusionne deux treebanks différents. Lorsqu'il y a plusieurs tags, on peut trouver des inconsistances dans l'annotation. Par exemple, dans les corpus "Brown" et "WSJ", le mot "to" a le tag "TO". Dans le corpus "Switchboard", il a le tag "TO" lorsqu'il indique l'infinitif, sinon "IN" lorsque c'est une préposition.

1. Universal TreeBank : https://universaldependencies.org/ [visité le 2021-09-09]

Voici quelques applications qui fournissent la tâche d'annotation morpho-syntaxique :

— https://nlp.stanford.edu/software/tagger.shtml [visité le 2021-09-26]
— https://github.com/sloria/textblob [visité le 2021-09-26]
— https://www.nltk.org/api/nltk.tag.html [visité le 2021-09-26]
— https://spacy.io/ [visité le 2021-09-26]
— https://github.com/clips/pattern [visité le 2021-09-26]

3 Approches d'étiquetage morpho-syntaxique

Ici, on va seulement présenter les approches : statistique et neuronale. Dans l'approche statistique, on va présenter le modèle de Markov caché qui est un modèle génératif, et le modèle de Markov basé sur l'entropie maximale qui est un modèle discriminatif. Dans l'approche basée sur les réseaux de neurones, on va présenter quelques architectures.

3.1 Modèle de Markov caché

Étant donné une séquence des variables d'état $q_1, \ldots, q_i$, l'hypothèse de Markov (comme vu dans le chapitre précédent) affirme que la probabilité d'être dans un état ne dépend que de l'état précédent. Donc, $P(q_i|q_1, \ldots, q_{i-1}) \approx P(q_i|q_{i-1})$. Une chaîne de Markov peut être formalisée comme un ensemble d'états $Q = \{q_1, q_2, \ldots, q_n\}$, une distribution initiale des probabilités $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$, $\sum_i \pi_i = 1$ et une matrice des probabilités de transition $A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}$

La figure 4.4 représente un exemple d'une chaîne de Markov pour la prédiction du temps. Il est claire que les états sont $Q = [HOT, COLD, WARM]$. Si on suppose que la distribution initiale est $\pi = [0.1, 0.7, 0.2]$, la probabilité de la séquence $HOT\ WARM\ COLD\ COLD$ sera calculée comme suit

$$\begin{aligned} P(H\,W\,C\,C) &= P(H)P(W|H)P(C|H\,W)P(C|H\,W\,C) \\ &= \pi(H)P(W|H)P(C|W)P(C|C) \\ &= 0.1 * 0.3 * 0.1 * 0.8 \end{aligned}$$

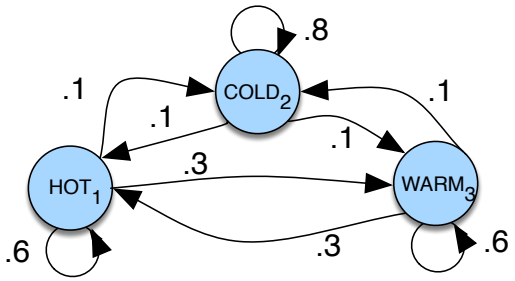


Figure 4.4 : *Exemple d'un chaîne de Markov pour la prédiction du temps (Jurafsky et Martin, 2019)*

Ceci est un modèle de langage qui ne représente que les probabilités de transition. Dans ce cas, on peut seulement représenter la probabilité de transition d'une catégorie à une autre. Si on estime la probabilité

d'émission d'un mot dans chaque état (étiquette), on peut estimer les étiquettes en utilisant la théorème de Bayes. Ce modèle est appelé : modèle de Markov caché (**HMM**) ; les mots d'une séquence sont observés contrairement aux catégories qui sont cachées et doivent être estimées. Chaque étiquette (catégorie) t_i est représentée par un un état q_i dans l'ensemble des états $Q = \{q_1, q_2, \ldots, q_n\}$. Les transitions d'un état vers un autre suivant un modèle Bi-gramme sont représentées par la matrice de transitions A. La distribution initiale des probabilités de ces états est annotée $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$ où $\sum_i \pi_i = 1$. Un mot w_i est représenté comme une observation dans la séquences des évènements observés $O = o_1 o_2 \ldots o_T$. Une observation o_t peut être générée dans un état q_i avec une probabilités d'observation (probabilités d'émission) $B = b_i(o_t)$. Un exemple d'un **HMM** est illustré dans la figure 4.5.

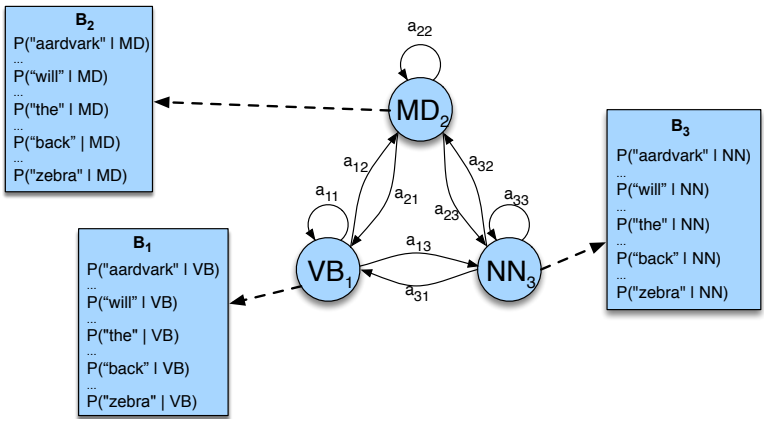


Figure 4.5 : *Exemple d'une représentation HMM (Jurafsky et Martin, 2019)*

La séquence estimée $\hat{t}$ est celle qui maximise la probabilité $P(t)$ et la probabilité $P(w|t)$ selon l'équation 4.3. En utilisant un modèle **Bi-gramme**, la probabilité $P(t)$ est calculée en utilisant les transitions de l'état initial t_1 jusqu'à le dernier t_n. Il faut savoir que la probabilité $P(t_1|t_0)$ est en réalité la probabilité initiale π_1 de l'état t_1. La probabilité $P(w_i|t_i)$ est une probabilité d'émission dans le modèle de Markov caché.

$$\begin{aligned} \hat{t} &= \arg\max_t P(t)P(w|t) \\ &= \arg\max_t \prod_{i=1}^{n} \underbrace{P(t_i|t_{i-1})}_{a_{i-1,i}} \underbrace{P(w_i|t_i)}_{b_i(w_i)} \end{aligned} \tag{4.3}$$

Avant de discuter comment une séquence $w = w_1 \ldots w_n$ est décodée selon l'équation 4.3, on commence par décrire l'entrainement. Dans un **HMM**, les états sont les tags (catégories) t_i et les observations sont les mots w_i. Notons la fonction qui compte le nombre des occurrence d'une séquence comme C. La probabilité de transition $a_{i-1,i}$ de l'état t_{i-1} vers t_i est estimée à partir du corpus d'entraînement selon l'équation 4.4. On note "<s>" comme l'étiquette qui précède la première catégorie de la phrase. Dans ce cas, la probabilité π_i de commencer par l'état t_i est estimée selon l'équation 4.5. La probabilité d'émission d'un mot w_i dans un état t_i est estimée selon l'équation 4.6.

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \tag{4.4}$$

$$\pi_i = P(t_i| < s >) = \frac{C(< s >, t_i)}{C(< s >)} \tag{4.5}$$

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \tag{4.6}$$

Prenons un exemple d'un corpus d'entraînement avec quatre phrases. Nous avons 4 catégories : DET, PRON, VERB et NOUN.

— un/DET ordianteur/NOUN peut/VERB vous/PRON aider/VERB
— il/PRON veut/VERB vous/PRON aider/VERB
— il/PRON veut/VERB un/DET ordinateur/NOUN
— il/PRON peut/VERB nager/VERB

Un exemple de calcul d'une transition est le suivant : $P(VERB|NOUN) = \frac{C(NOUN, VERB)}{C(NOUN)} = \frac{1}{2}$. La probabilité d'émission du mot "veut" lorsque le tag est "VERB" est calculée comme suit : $P(veut|VERB) = \frac{C(VERB, veut)}{C(VERB)} = \frac{2}{7}$. La probabilité q'un pronom soit au début de la phrase est calculée comme : $\pi_{PRON} = \frac{C(<s>, PRON)}{C(<s>)} = \frac{3}{4}$. La matrice des transitions ainsi que la distribution initiale sont illustrés dans le tableau 4.2. Portant ce n'est pas utilisé dans le calcul, nous avons ajouté le passage d'un état vers la fin afin de compléter la distribution (avoir une somme de 1 sur les lignes). Les probabilités d'émission sont illustrées dans le tableau 4.3.

	DET	PRON	VERB	NOUN	</s>
DET	0	0	0	1	0
PRON	0	0	1	0	0
VERB	1/7	2/7	1/7	0	3/7
NOUN	0	0	1/2	0	1/2
<s>(π)	1/4	3/4	0	0	0

Tableau 4.2 : *Exemple des probabilités de transition et une distribution initiale*

	un	ordinateur	peut	vous	aider	il	veut	nager
DET	1	0	0	0	0	0	0	0
PRON	0	0	0	1/4	0	3/4	0	0
VERB	0	0	2/7	0	2/7	0	2/7	1/7
NOUN	0	1	0	0	0	0	0	0

Tableau 4.3 : *Exemple des probabilités d'émission*

Revenons maintenant au décodage des tags (étiquetage). Après l'entraînement d'un modèle de Markov caché $\lambda = (A, B)$, on veut l'utiliser pour estimer la séquence des étiquettes $\hat{t} = \hat{t}_1\hat{t}_2 \dots \hat{t}_n$ à partir d'une séquence des observations (mots) $w = w_1w_2 \dots w_n$. Dans chaque état s de la séquence, on se base sur l'estimation précédente pour calculer les probabilités de tous les tags afin de choisir celui qui maximise la probabilité comme estimation courante. Il faut toujours sauvegarder l'état précédent pour revenir en fin d'algorithme. Ceci est appelé la recherche **Viterbi** (voir l'algorithme 4.1).

En utilisant le modèle entraîné dans l'exemple précédent, on veut trouver les tags de cette phrase : il peut aider. Suivant l'algorithme de Viterbi, on aura le tableau 4.4 où dans chaque cellule on précise les calculs séparés par des points virgules (les états précédent) et le pointeur de retour entre deux crochets. Dans la dernière séquence ("aider"), l'état qui maximise la probabilité est "VERB". Le retour est l'état 3 ("VERB") ayant un retour vers l'état 2 ("PRON"). Donc, le texte annoté est : il/PRON peut/VERB aider/VERB.

3.2 Maximum entropy

Les estimations par **HMM** se basent seulement sur des statistiques des mots. Il est difficile à introduire des caractéristiques comme les suffixes, le majuscule, etc. La régression logistique, qui est un modèle discri-

	il	peut	aider
DET	$\frac{1}{4} * 0$ [0]	$0 * 0 * 0$; $\frac{3}{4} * 0 * 0$; $0 * \frac{1}{7} * 0$; $0 * 0 * 0$ [2]	$0 * 0 * 0$; $0 * 0 * 0$; $\frac{6}{28} * \frac{1}{7} * 0$; $0 * 0 * 0$ [3]
PRON	$\frac{3}{4} * 1$ [0]	$0 * 0 * 0$; $\frac{3}{4} * 0 * 0$; $0 * \frac{2}{7} * 0$; $0 * 0 * 0$ [2]	$0 * 0 * 0$; $0 * 0 * 0$; $\frac{6}{28} * \frac{2}{7} * 0$; $0 * 0 * 0$ [3]
VERB	$0 * 0$ [0]	$0 * 0 * \frac{2}{7}$; $\frac{3}{4} * 1 * \frac{2}{7}$; $0 * \frac{1}{7} * \frac{2}{7}$; $0 * \frac{1}{2} * \frac{2}{7}$ [2]	$0 * 0 * \frac{2}{7}$; $0 * 1 * \frac{2}{7}$; $\frac{6}{28} * \frac{1}{7} * \frac{2}{7}$; $0 * \frac{1}{7} * \frac{2}{7}$ [3]
NOUN	$0 * 0$ [0]	$0 * 1 * 0$; $\frac{3}{4} * 0 * 0$; $0 * 0 * 0$; $0 * 0 * 0$ [1]	$0 * 1 * 0$; $\frac{6}{28} * 0 * 0$; $0 * 0 * 0$; $0 * 0 * 0$ [1]

Tableau 4.4 : *Exemple d'exécution de l'algorithme de Viterbi*

Algorithme 4.1 : *Algorithme de Viterbi pour encoder une séquence selon un modèle de Markov caché.*

Data : $w = w_1 \dots w_T$, HMM $\lambda = (A, B)$ avec N états
Result : *meilleur_chemin*, *prob_chemin*
Créer une matrice $viterbi[N, T]$;
pour *état* $s = 1 \dots N$ **faire**
  $viterbi[s, 1] = \pi_s * b_s(w_1)$; $backpointer[s, 1] = 0$;
fin
pour $t = 1 \dots T$ **faire**
  pour *état* $s = 1 \dots N$ **faire**
    $viterbi[s, t] = \max_{s'=1}^{N} viterbi[s', t-1] * a_{s',s} * b_s(w_t)$;
    $backpointer[s, t] = \arg\max_{s'=1}^{N} viterbi[s', t-1] * a_{s',s} * b_s(w_t)$;
  fin
fin
$prob\_chemin = \max_{s=1}^{N} viterbi[s, T]$; $pointeur\_chemin = \arg\max_{s=1}^{N} viterbi[s, T]$;
meilleur_chemin est le chemin qui commence par *pointeur_chemin* et qui suit *backpointer*
return *meilleur_chemin*, *prob_chemin* ;

minatif, peut combiner plusieurs caractéristiques afin d'estimer une probabilité de sortie. Malheureusement, cette méthode ne prend pas les séquences en considération. On peut utiliser des caractéristiques sur le mot actuel et les mots avant afin d'estimer l'étiquette ; comme dans les **HMM**. Dans ce cas le modèle sera appelé Maximum Entropy Markov Model (MEMM). Le problème revient à maximiser les probabilités individuelles de chaque étiquette $\hat{t}_i$ (voir l'équation 4.7).

$$\hat{t} = \arg\max_t P(t|w) = \arg\max_t \prod_i P(t_i|w_i, t_{i-1}) \tag{4.7}$$

Notons s_i^j la séquence $s_i \dots s_j$. Afin d'estimer l'étiquette t_i d'un mot w_i, on prend l mots précédents et l mots suivants, en plus de k tags précédents. On utilise un ensemble des caractéristiques f où f_j peut être la présence du majuscule dans le mot, etc. En utilisant la régression logistique, le problème sera formulé comme dans l'équation 4.8. L'apprentissage vise à minimiser la somme des erreurs d'étiquetage multi-classes de chaque mot (pour chaque mot, on aura plusieurs probabilités ; une pour chaque étiquette).

$$\hat{t} = \arg\max_t \prod_i \frac{exp\left(\sum_j \theta_j f_j(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}{\sum_{t' \in tags} exp\left(\sum_j \theta_j f_j(t'_i, w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)} \tag{4.8}$$

Voici une liste des caractéristiques f qu'on peut utiliser dans cette méthode :
— Le mot w_i (en considérant chaque mot comme caractéristique)
— Les étiquettes précédentes

— w_i contient un préfixe à partir d'une liste ($len(prefixe) \leq 4$)
— w_i contient un suffixe à partir d'une liste ($len(suffixe) \leq 4$)
— w_i contient un nombre
— w_i contient une lettre en majuscule
— w_i contient un trait d'union
— w_i est complètement en majuscule
— La forme du mot w_i (Ex. X.X.X)
— w_i est en majuscule avec un trait et un nombre (Ex. CFC-12)
— w_i est en majuscule suivi des mots Co., Inc., etc. dans 3 mots maximum

3.3 Modèle neuronal

Les réseaux de neurones récurrents sont conçus pour traiter les séquences. Nous avons vu qu'un Recurrent Neural Network (RNN) peut être utilisé comme modèle de langage en passant le mot encodé avec **One-Hot** en entrée et en estimant le mot suivant sachant le contexte précédent. Si on modifie la sortie : à la place des probabilités des mots du vocabulaire, on essaye d'estimer les probabilités des catégories possibles. On peut, bien sûr, utiliser une couche cachée pour apprendre une représentation plus compacte des mots en entrée (**embedding**). La figure 4.6 représente un exemple d'un réseaux de neurones récurrent pour la détection des étiquettes morpho-syntaxique suivant cette modification.

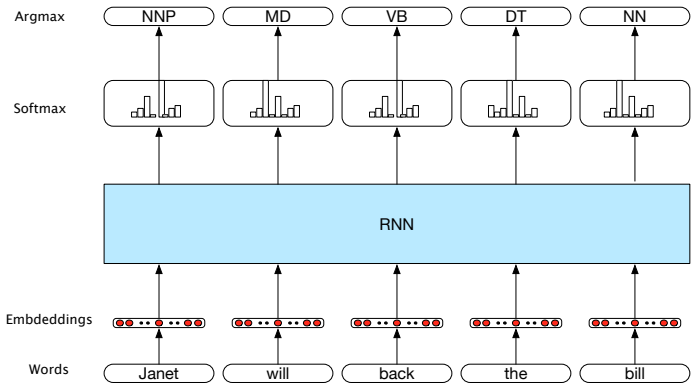


Figure 4.6 : *RNN simple pour l'étiquetage morpho-syntaxique (Jurafsky et Martin, 2019)*

Si on veut avoir des réseaux plus profonds et plus complexes, on peut empiler plusieurs RNN ensembles. Cela permet au modèle d'apprendre des traitements plus complexes améliorant la prédiction. Pour entraîner ce modèle, on aura besoin d'un grand dataset ; la taille de ce dernier augmente avec le nombre des réseaux empilés. La figure 4.7 est une illustration d'un modèle avec trois RNN empilés.

Dans les modèles neuronaux précédents, l'étiquette courante t_i dépend sur le mot courant w_i et le contexte passé. Des fois, savoir le contexte futur peut améliorer la décision courante ; on peut trouver une dépendance entre le mot courant et les mots après. Afin d'estimer l'étiquette courante en se basant sur le contexte futur, on utilise un RNN avec une séquence inversée ; c-à-d. l'étiquette t_i sera dépendante à la séquence $w_n, w_{n-1}, \ldots, w_{i+1}, w_i$. Afin d'exprimer la bidirectionnalité, on utilise deux RNN : en avant $RNN_{forward}(w_1^i)$ et en arrière $RNN_{backword}(w_i^n)$. On fait la somme entre les deux états cachés $h_i^f \oplus h_i^b$ résultant à un vecteur qui doit passer par une fonction "softmax" afin d'estimer les probabilités des tags. Cette architecture est illustrée dans la figure 4.8.

Les langues sont toujours en amélioration : ajout des nouveaux mots, des nouvelles formes, etc. En plus,

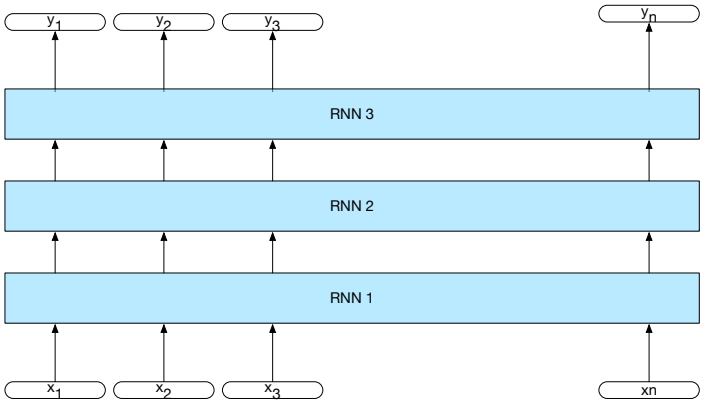


Figure 4.7 : *RNNs empilés pour l'étiquetage morpho-syntaxique (Jurafsky et Martin, 2019)*

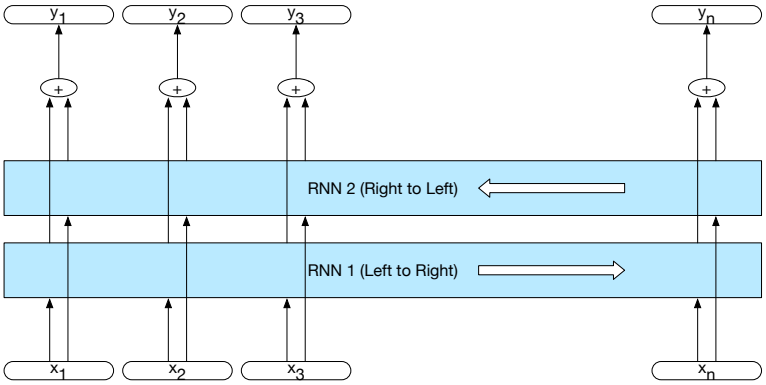


Figure 4.8 : *RNN bidirectionnel pour l'étiquetage morpho-syntaxique (Jurafsky et Martin, 2019)*

on ne peut pas capturer toutes les variations de tous les mots en utilisant un corpus limité surtout dans les langues fortement flexionnelles. Afin de représenter les variations morphologiques d'un mot, on doit descendre au niveau caractère. Une idée (voir la figure 4.9) est d'utiliser un modèle de langage au niveau caractère afin d'encoder un mot et combiner ce code avec le code lexicale (**One-Hot** ou **embedding**) du mot. Le code résultat sera pris comme l'entrée du réseau récurrent (simple, empilé ou bidirectionnel).

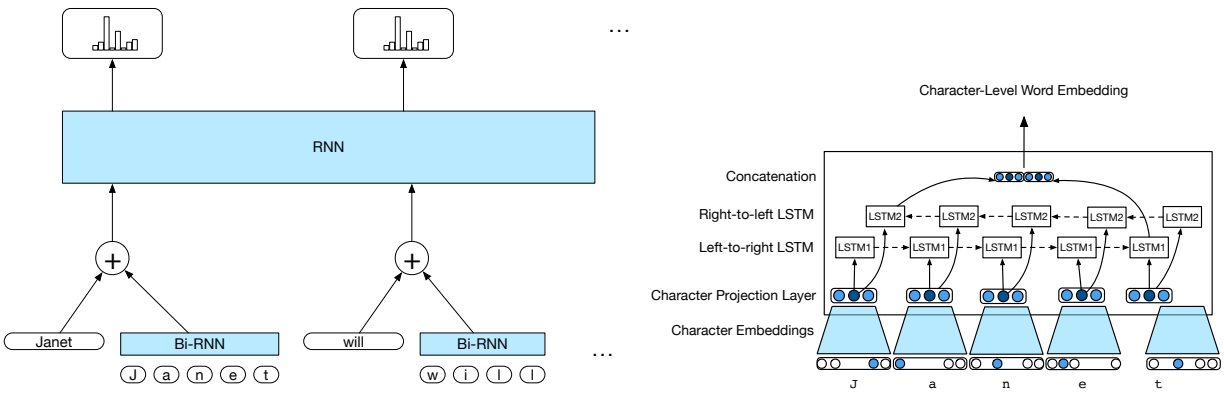


Figure 4.9 : *RNN avec des embedding de caractères pour l'étiquetage morpho-syntaxique (Jurafsky et Martin, 2019)*

Discussion

Chaque langue doit avoir une méthode pour identifier les entités du monde et les actions d'une entité. Les linguistes font la différence entre la fonction d'un mot dans la phrase en utilisant des catégories. Par exemple un nom c'est la référence à un objet (concret ou abstrait), or un adjectif est un mot qui représente une propriété d'un nom. Donc, le fait de savoir la catégorie d'un mot aide la compréhension d'une phrase. Automatiser la tâche de détection des catégories d'une séquence de mots est une étape vers la compréhension de la phrase. Même dans les tâches statistiques (qui ne sont pas besoin de la compréhension), les catégories des mots peuvent être un bon indicateur.

Étiqueter les mots par leurs catégories grammaticales est une tâche appartenant aux tâches de l'étiquetage des séquences. Plusieurs approches ont été utilisées pour résoudre ce problème : par règles, statistique et par réseaux de neurones. Les règles sont difficiles à mettre en œuvre ; il faut avoir une connaissance profonde de la langue traitée. Cette approche a été remplacée par l'approche statistique qui vise à estimer l'étiquette en se basant sur des statistiques sur le mot actuel ainsi que les mots précédents et leurs étiquettes. Avec l'arrivée des RNN, cette tâche a devenue plus facile ; il faut juste avoir un corpus étiqueté avec une taille suffisante. Dans les architectures présentées dans ce chapitre, l'entrée d'un RNN est une représentation vectorielle du mot courant. En réalité, on peut définir des architectures plus complexes en introduisant les caractéristiques d'un mot comme par exemple la présence des suffixes. On peut même encoder tous les suffixes avec l'encodage One-Hot et fusionner ce code avec celui du mot.

Dans ce chapitre, nous avons présenté l'étiquetage des séquences et plus précisément l'étiquetage morpho-syntaxique. Nous avons vu les approches et quelques méthodes utilisées pour résoudre cette tâche. Mais la question qui se pose : comment peut-on évaluer un système d'étiquetage des séquences. Tout simplement, cette tâche est un cas spécial de classification : à la place de classer une phrase, on classe chaque mot (ou partie) à part en tenant compte de leurs interactions (temporelles). Donc, les métriques d'évaluation de la classification comme la justesse (accuracy), le rappel (recall), la précision (precision), etc. peuvent être utilisées. Ceci concerne l'évaluation intrinsèque, sinon on peut évaluer l'effet de cette tâche sur une autre (évaluation extrinsèque).