

La compression de phrases pour le résumé

Approches automatiques pour des résumés plus qualitatifs

Mémoire réalisé par
Jeremy Chelala

Promoteur
Thomas François

Année académique 2017-2018
Master en Linguistique, finalité en traitement automatique du langage

Remerciements

Je tiens à remercier les quelques personnes sans lesquelles je n'aurais sans doute pas pu terminer ce mémoire.

Je souhaite remercier, en premier lieu, le promoteur de ce mémoire, M. François, pour sa disponibilité, sa méticulosité et ses conseils avisés lors de toutes nos entrevues.

Je voudrais aussi remercier M. Torres-Moreno, chercheur français en résumé automatique de texte, d'abord, pour certains de ses écrits scientifiques qui m'ont permis d'y voir plus clair dans le vaste champ de recherche en résumé, et ensuite, pour m'avoir donné accès à un corpus annoté pour la compression.

Mes derniers remerciements s'adressent à mes parents. Je les remercie pour leur soutien précieux, dans la rédaction de ce mémoire comme durant tout mon parcours universitaire. Je les remercie de m'avoir donné la chance de suivre des études supérieures. Ce mémoire, qui en marque la fin, leur est dédié.

Table des matières

Table des matières	3
1 Introduction	5
1.1 Problématique	5
1.2 Plan de l'exposé	7
I État de l'art	9
2 Résumé automatique de texte	13
2.1 Définitions et enjeux	14
2.2 Méthodologies	16
2.3 Approche abstractive	17
2.4 Approche extractive	18
2.5 Évaluation	27
2.6 Limites de l'extraction de phrases	31
3 Compression automatique de phrases	35
3.1 Définitions et enjeux	36
3.2 Méthodologies	37
3.3 Approche symbolique	37
3.4 Approche statistique	38
3.5 Évaluation	41
4 Compression de phrases pour le résumé	45
4.1 Intérêt de la compression pour le résumé	46
4.2 Méthodologies	47
II Implémentation de compresseurs automatiques	53
5 Compression symbolique	57
5.1 Méthodologie	58
5.2 Évaluation	61
5.3 Discussion	63
6 Compression probabiliste	67
6.1 Méthodologie	68
6.2 Champs conditionnels aléatoires	72
6.3 Facteurs et outils d'annotation	75
6.4 Conception du modèle	77
6.5 Évaluation	79
7 Conclusion générale	89
7.1 Résultats	89
7.2 Perspectives	90
Bibliographie	91

Chapitre 1

Introduction

1.1 Problématique

Grâce au développement et à la démocratisation de l'internet au niveau mondial, l'accès à l'information devient, de nos jours, de plus en plus rapide et facile. Tous les jours ou presque, nous effectuons des recherches sur la toile pour obtenir des réponses à toutes sortes de questions, la plupart du temps avec succès. En effet, la quantité de données disponibles, tellement large et variée, garantit souvent un résultat satisfaisant. Mais cette masse d'information, si utile pour la connaissance, provoque paradoxalement une surcharge d'information : il y a tellement de contenu disponible qu'il devient impossible d'en traiter la majeure partie. Ce phénomène semble d'autant plus inquiétant que le nombre de données produites ne cesse de croître, proportionnellement à l'augmentation du nombre d'ordinateurs et d'ordiphones (ou *smartphones*) en circulation (Villars *et al.* 2011 : 1).

Or, cette surcharge informative pose plusieurs problèmes. Le sujet a été étudié par une série de sociologues et de médecins, notamment dans son influence sur l'efficacité et le bien-être dans le milieu professionnel. Ils ont montré que le trop-plein d'information augmentait le stress et la fatigue chez le travailleur, le menant parfois à un mal-être au travail ou des maladies physiques (Lewis *et al.* 1996 : 2). Par ailleurs, le fait que nous recevions trop d'information nous empêche de traiter les données correctement. D'abord, parce que leur nombre s'avère trop important : parmi toute l'information disponible sur un sujet de recherche, nous sommes presque toujours contraints de sélectionner une petite partie des documents proposés, rendant nos recherches particulièrement incomplètes. Ensuite, parce que la surcharge d'information rend plus difficile l'obtention d'informations pertinentes (Edmunds & Morris 2000 : 22) : plus un volume d'information est grand, plus il est susceptible de contenir des éléments redondants ou sans intérêt pour nos recherches, et plus nous prenons de temps pour y trouver ce qui nous est utile. Ce qui nous conduit au constat suivant : la *surinformation* est, en fait, une forme de *désinformation*.

On connaît la rapidité et l'efficacité des ordinateurs à effectuer, pour l'homme, des tâches nombreuses et de plus en plus complexes. Dès lors, la machine ne pourrait-elle pas nous aider à mieux disposer de l'information, en la filtrant d'une quelconque façon ? C'est l'enjeu principal du résumé automatique de texte, un domaine du traitement automatique des langues et du traitement de l'information. Son objectif est d'arriver à produire un système capable de résumer un texte écrit en langue naturelle, c'est-à-dire de le condenser en reprenant l'essentiel de son contenu. Ce système permettrait de se procurer des résumés à la demande, et d'une façon bien plus rapide que si l'humain s'en chargeait. L'intérêt d'un résumeur automatique se situe dans l'utilité du résumé en tant que genre textuel. Le résumé est un outil qui nous aide beaucoup lorsqu'on n'a pas le temps ni les moyens de lire toute l'information à notre disposition. Nous nous servons d'ailleurs quotidiennement d'applications du résumé, en lisant, par exemple, des synopsis, des critiques d'œuvres

ou les chapeaux d'articles de presse pour nous faire une idée convenable du contenu d'un texte avant de le lire en entier. En d'autres termes, le résumé est un moyen efficace, mais aussi naturel et habituel pour l'homme de s'orienter dans ses recherches, qu'elles soient culturelles, journalistiques ou scientifiques. Certains algorithmes de recherche et d'extraction de l'information profiteraient également de l'accès automatique aux résumés, qui ont des propriétés plus intéressantes en matière d'informativité.

Très tôt déjà, la recherche s'était penchée sur le problème du résumé automatique. Les premiers travaux datent en effet des années 1950-60 (Luhn 1958, Edmundson 1969). Plus de soixante ans plus tard, aucune étude n'a, à ce jour, présenté de solution définitive au problème. Avec le temps, les chercheurs semblent avoir abandonné l'idée de produire des résumés entièrement paraphrastiques, cette méthode présentant trop de contraintes. Ils se sont plutôt attardés sur une sous-tâche du résumé : l'extraction de phrases. Pour ce problème particulier, plusieurs solutions convaincantes ont bien été proposées (Radev *et al.* 2004, Svore *et al.* 2007), et l'on trouve aujourd'hui des services de « résumé extractif », tels que *Essential Summarizer*¹ et *Intellexer Summarizer*².

Toutefois, malgré leur utilité indéniable, ce genre d'applications simplifient outre mesure la tâche complexe que constitue le résumé. Il ne s'agit pas de simplement extraire des parties du texte, mais d'appréhender le contenu informationnel d'un texte et de le retranscrire de la façon la plus condensée et claire possible. Or, pour être cohérente, cette retranscription exige presque toujours de retoucher le texte original. Il faut donc dépasser le résumé par extraction, et dès lors associer l'extraction de phrases à d'autres techniques de réduction ou de remaniement de l'information. Parmi ces techniques, la compression automatique de phrases apparaît essentielle. *Compresser* une phrase revient à la débarrasser de tous ses segments jugés inutiles dans un contexte donné, tout en conservant sa grammaticalité. C'est une sorte de résumé, mais appliqué au niveau de la phrase. Un compresseur automatique appliqué au résumé extractif ferait gagner le résultat final en pertinence et en fluidité. En pertinence, parce que la compression garantit un rendement informatif maximal au sein de la phrase, et en fluidité, parce que les phrases courtes se rapprochent plus du style concis et direct propre au résumé (Vanderwende *et al.* 2006, Silveira & Branco 2012).

Ce travail analyse, d'un point de vue à la fois théorique et expérimental, les différentes méthodologies en compression de phrases, spécifiquement pensées pour l'amélioration de résumés extractifs. Nous cherchons tout d'abord à comprendre de manière approfondie les besoins du résumé en matière de compression, en identifiant les limites théoriques et algorithmiques des systèmes d'extraction de phrases. Nous voulons également présenter les différentes façons d'implémenter l'interaction entre un compresseur et un résumeur automatiques. Nous souhaitons enfin montrer expérimentalement les résultats produits par les deux approches principales en compression automatique, et ainsi mieux cerner les difficultés et avantages de chaque méthode. Cela nous permettra de faire le point sur l'avancement des recherches dans les deux domaines du résumé et de la compression automatiques, et, finalement, d'envisager des solutions nouvelles, qui, à terme, rendront possible la production automatique de résumés exploitables par l'homme.

1. https://play.google.com/store/apps/Essential_Summarizer.

2. <http://summarizer.intellexer.com>.

1.2 Plan de l'exposé

L'exposé se divise en deux grandes parties : d'abord l'état de l'art, ensuite les expérimentations.

Nous commençons, dans la partie I, par établir une somme des connaissances dans les domaines du résumé et de la compression, organisée chronologiquement et thématiquement (chapitres 2 et 3). Cette première étape nous semble essentielle si l'on veut saisir pleinement l'intérêt du résumé et de la compression pour le résumé. En particulier pour le domaine du résumé automatique, le champ méthodologique est tellement vaste qu'il exige une structure intelligente du propos. Cette première partie comprend également un chapitre décrivant les approches actuelles pour l'application de la compression de phrases au résumé automatique (chapitre 4). À notre connaissance, notre état de l'art est le seul travail de synthèse en français qui réunisse ces trois sujets de recherche et les détaille de façon aussi approfondie, en particulier en ce qui concerne la compression appliquée au résumé.

Nous implémentons ensuite, dans la partie II, deux compresseurs originaux afin de proposer de nouvelles méthodologies pour la compression de phrases et de tirer des conclusions expérimentales sur ce domaine. Les deux compresseurs suivent des méthodologies très différentes : l'une s'appuie sur l'approche symbolique (chapitre 5), l'autre sur l'approche statistique (chapitre 6). Cette ambivalence permet d'envisager plus de facettes de la tâche de compression de phrases, nous confrontant à des problèmes variés et qu'on n'aurait pas rencontrés autrement. Elle autorise aussi une comparaison plus détaillée entre deux approches courantes en traitement automatique des langues, nécessaire pour déterminer si l'une est plus adéquate que l'autre dans le cas de la compression. Nos différentes expérimentations nous en apprennent plus sur les particularités de chaque méthode et sur leur impact envers le résumé.

Le travail se termine sur une conclusion générale, dans laquelle nous revenons sur les grands éléments de conclusion dégagés par l'état de l'art et par nos deux expériences originales. Nous mettons ces éléments en relation pour répondre aux problématiques qui viennent d'être posées et proposer de nouvelles solutions vers l'amélioration des résumés produits automatiquement.

Première partie

État de l'art

Introduction

Nous rendons compte, dans cette première partie, des divers travaux rédigés dans les domaines du résumé automatique et de la compression de texte, deux domaines de recherche en traitement automatique du langage (TAL). Les résumés des travaux de recherche se veulent à la fois complets et critiques. On prêterait attention aux avantages et défauts tant théoriques que pratiques de l'usage des algorithmes présentés.

Nous consacrons un chapitre par domaine envisagé. Il nous semble en effet plus pertinent, dans un état de l'art, de d'abord les décrire séparément pour mieux comprendre les enjeux des différents travaux présentés. Cela permettra également de mieux discerner les apports de la compression de phrases au résumé de texte, lorsque nous aborderons les travaux en compression pour le résumé.

Notre état de l'art sert tout d'abord à contextualiser la recherche expérimentale que nous menons en partie II. Si l'on veut pouvoir justifier l'originalité de nos méthodes, l'on doit au préalable parler de ce qui a déjà été publié dans la littérature. Le développement de la compression de phrases étant fortement lié au domaine du résumé, il apparaît intéressant, dans un esprit d'exhaustivité, d'évoquer ces deux tâches et d'expliquer en quoi leurs objectifs se rejoignent. L'état de l'art nous donne également l'occasion de justifier pleinement l'intérêt de la compression de phrases pour le résumé et donc de notre travail dans son ensemble. C'est pourquoi nous identifierons, d'une part, les enjeux du résumé automatique de texte, et, d'autre part, ses limites actuelles. Cela nous permettra de mieux cerner les besoins de ce domaine et de mieux comprendre quels avantages la compression a à offrir au résumé.

Cet état de l'art nous paraît d'autant plus intéressant qu'il n'existe pas, à notre connaissance, de travail analogue au nôtre dans les publications académiques. Ce travail est, premièrement, un des seuls à réunir un historique des recherches de deux domaines distincts aussi approfondi. Le fait qu'il soit rédigé en français constitue une seconde singularité, puisqu'une grande majorité des écrits scientifiques abordant nos domaines d'intérêt sont rédigés en anglais. Enfin, il présente une première typologie des méthodologies existantes pour l'inclusion d'un module de compression dans un résumeur automatique.

La structure de cette partie sur l'état de l'art est la suivante. Dans le chapitre 2, nous nous attardons sur la tâche de résumé automatique de texte, en commençant par évoquer les notions-clés et enjeux de ce domaine du TAL. Le chapitre 3 suit en tous points la structure du chapitre 2, mais traite de la compression automatique de phrases. Le chapitre 4 est dédié aux approches combinant des techniques issues de ces deux domaines.

Chapitre 2

Résumé automatique de texte

Introduction

Ce chapitre définit la tâche de résumé automatique de texte (RAT à partir de ce point), ses enjeux et les méthodes utilisées dans ce domaine du TAL. Ces dernières sont nombreuses, tant l'on étudie le RAT depuis longtemps. Les premières études voient en effet le jour dans les années 1950. S'il ne nous a pas été possible de toutes les décrire, nous avons voulu présenter les principales techniques dont l'influence a été cruciale pour l'avancement de la recherche en RAT.

Autant que faire se peut, nous avons respecté la chronologie des travaux dans notre présentation. Toutefois, il nous a aussi paru nécessaire d'organiser l'état de l'art d'un point de vue méthodologique, à cause du grand nombre d'articles publiés en RAT. C'est pourquoi, comme nous l'expliquons au point 2.4.3, il nous a fallu abandonner la composante historique dans la description des techniques les plus récentes.

Avant l'état de l'art proprement dit, nous évoquons, dans la première section (2.1), les enjeux de l'automatisation de la production de résumés et discutons la définition du résumé. Les sections 2.2 à 2.4 décrivent les différentes approches conçues pour le RAT. Nous consacrons une section (2.5) à la question de l'évaluation de résumeurs automatiques. La dernière section offre une réflexion critique sur les approches récentes en RAT et les limites de celles-ci.

2.1 Définitions et enjeux

2.1.1 Enjeux

La masse de données dans le monde ne cesse croître d'année en année, et ce de façon exponentielle. On estime que la quantité de données numériques atteindra 44 trillions gigaoctets d'ici 2020 (Turner *et al.* 2014). Or, la plupart de ces données prennent la forme de textes en langue naturelle, écrits par et pour l'homme, dans un nombre de langues de plus en plus grand. Confronté à cette masse de données, l'homme n'arrive pas les traiter correctement, tandis que la machine ne peut pas l'aider dans cette tâche à cause de leur nature non structurée.

Disposer de résumés des textes pourrait constituer une solution élégante à cette situation problématique. Et ce, pour les raisons suivantes (Torres-Moreno 2014 : 4-5) :

- Le résumé, de par sa taille réduite et son style plus direct, prend moins de temps à lire.
- Il aide à mieux identifier les informations importantes du texte.
- Il évite au lecteur de devoir lire l'entièreté du texte, lorsque l'information qu'il cherche est déjà contenue dans le résumé ou lorsque le résumé lui indique qu'elle ne se trouve pas dans le texte complet.
- En présentant l'essentiel d'un texte, il offre également un moyen plus pertinent et plus rapide de rechercher des documents et d'extraire de l'information.

Pour le dire simplement, le résumé peut servir de source d'accès primaire à l'information, bien plus pratique à manipuler que le texte original. Cela vaut pour l'homme, mais aussi pour la machine.

Pour les processeurs, il y a des enjeux d'efficacité et d'optimisation de l'espace de stockage. S'il existe déjà des techniques de compression d'octets efficaces (compression de Huffman, *gunzip* et traitements similaires), celles-ci conservent, malgré tout, l'entièreté d'un texte donné en mémoire, sans tenir compte de ce qu'il contient. Elles ne sont d'aucune utilité pour des tâches impliquant du traitement de l'information. Avec un résumé pour chaque texte résumable, les systèmes gagneraient tout d'abord en temps de lecture et de fouille en analysant des textes plus courts. Les tâches de recherche et d'extraction d'information gagneraient également en rapidité, puisque le résumé réduit drastiquement la taille des textes à fouiller, et aussi parce que le système ne perd pas de temps à lire l'inutile ou le redondant dans les textes. Le cas échéant, le programme mettra à disposition le résumé en lieu et place du texte original si l'information demandée s'y retrouve.

Si l'on comprend l'intérêt de disposer d'un résumé pour un texte ou un ensemble de textes, il nous reste à motiver celui de l'automatisation de cette tâche. « Pourquoi ne pas se contenter d'un résumé rédigé par un résumeur professionnel ? » se demandent Minel *et al.* (2001 : 2). La réponse à cette question est multiple. Tout d'abord, parce que l'homme prend beaucoup de temps pour effectuer cette tâche. Rien que cet argument a suffi à justifier l'automatisation de nombreuses tâches dans le passé plus ou moins récent de l'industrie informatique. Ensuite, parce que la masse de données à traiter s'avère trop grande pour envisager un traitement humain. En effet, la plupart des textes ne viennent pas avec leur résumé (Minel *et al.* 2001 : 3). Le dernier argument en faveur de l'automatisation du résumé automatique est son caractère reproductible et sa plus grande fiabilité. Si l'homme

produit des résumés de très bonne qualité (pour autant qu'il connaisse bien le sujet traité), il reste subjectif dans sa démarche. Il n'est dès lors pas souhaitable qu'il se charge de résumer un grand ensemble de textes desquels on veut extraire de l'information.

2.1.2 Définitions

Avant d'évoquer les méthodes automatiques de résumé, il convient de définir ce que les chercheurs et nous-même entendons par « résumé », ainsi que de décrire la tâche de RAT et ses objectifs.

Qu'est-ce qu'un résumé ?

Un résumé se définit comme l'action d'abrégier un texte ou un discours sous la forme d'un texte continu et cohérent, tout en garantissant le maintien de l'essentiel de l'information du texte ou discours de départ. On repère dans cette définition trois composantes importantes de tout résumé : une taille considérablement réduite, une forme semblable au discours naturel et la conservation de l'information. Si une écrasante majorité de chercheurs s'accordent sur cette définition primaire du résumé, on observe en revanche des divergences dans la littérature quant à sa fonction principale. Ainsi, chez van Dijck, « la fonction première des résumés est d'indiquer et prédire la structure et le contenu du texte »¹, tandis que Cleveland le considère comme un « véritable substitut » du texte², et que l'ANSI (*American National Standards Institute*) situe l'utilité du résumé dans la gestion de base de données informatisées³ (tous cités dans Torres-Moreno 2014 : 5). Cette diversité d'opinions rend avant tout compte de la grande variété d'usages du résumé, tant il paraît utile.

La production de résumés résulte d'un processus cognitif complexe qui nécessite au préalable la compréhension de l'ensemble du texte ainsi qu'un certain nombre de connaissances externes au texte (Torres-Moreno 2014 : 8). Suivant la division de la tâche de résumé proposée par Saggion & Lapalme (2002 : 497), on peut identifier les trois étapes suivantes au résumé⁴ :

1. Extraire les informations qui seront reprises dans le résultat final, c'est-à-dire les informations essentielles ;
2. Condenser cette information et déterminer une structure logique et cohérente au résumé ;
3. Mettre en œuvre cette structure et présenter le résumé en utilisant un langage naturel.

Il existe par ailleurs une typologie fournie du concept de résumé, en particulier dans la littérature anglo-saxonne, qui dispose de plus de termes qu'en français pour désigner les différents types de résumés en fonction de leurs objectifs et usages. Ainsi, en anglais, le concept de résumé peut être désigné par le générique *summary*, mais aussi par *extract*, *abstract*, *abridgement*, *precis*, *digest*, *highlight* ou encore *synopsis*, termes dont on peut trouver les définitions succinctes chez Saggion & Lapalme (2002 : 498). Le résumé se différencie

1. *The primary function of abstracts is to indicate and predict the structure and content of the text.*

2. *An abstract summarizes the essential contents of a particular knowledge record, and it is a true surrogate of the document.*

3. *[A]bstracts are useful in access publications and machine-readable databases.*

4. Saggion & Lapalme (2002 : 497) identifient une première étape d'interprétation du texte, que nous considérons plutôt ici comme une condition préalable au processus.

également selon le genre du texte qu'il abrège : on ne résume pas de la même façon un roman qu'un article scientifique ou un article de presse. En ce qui nous concerne, nous ne limiterons pas notre définition du résumé à un sous-ensemble d'objectifs ou d'usages, mais nous nous contenterons d'une définition large et englobante du résumé, profitant du sens générique du mot français et considérant qu'un bon résumé doit être capable de remplir toutes les fonctions désignées par les termes anglo-saxons.

Production automatique de résumés

La tâche de RAT consiste à automatiser l'ensemble du processus de résumé, afin qu'un système soit capable de produire en sortie, pour un ou plusieurs texte(s) donné(s) en entrée, un résumé de ce(s) texte(s) dont la qualité vaut celle d'un résumé écrit par l'homme. Certains auteurs incluent également dans leur définition du RAT un « taux de compression », noté τ , qui indique le pourcentage de phrases ou de mots conservés dans le résumé. Sa valeur minimale varie selon les auteurs entre un tiers et un demi (Torres-Moreno 2014 : 8).

Actuellement, la qualité des résumés produits automatiquement ne rivalise pas avec le savoir-faire humain, tant la tâche s'avère complexe. En particulier, trois causes de difficultés majeures surgissent lorsqu'on envisage la voie informatique : l'accès au contenu du texte à abréger, la maîtrise de compétences productives (ou génératives) en langue naturelle et la subjectivité de la tâche. Appréhender le sens et les idées d'un texte représente une tâche complexe, à cause des nombreuses ambiguïtés de la langue : ses défis majeurs sont le traitement des unités polysémiques et de l'idiomaticité, la résolution d'anaphores et le recours à des connaissances extratextuelles, pour ne pas évoquer la gestion de phénomènes comme l'ironie et l'implicite. Les recherches en génération automatique de texte traitent des problèmes tout aussi compliqués, pour lesquels les dernières avancées n'ont pas encore trouvé de solution évidente. Enfin, la subjectivité dans le choix des informations à conserver dans le résumé soulève des ambiguïtés méthodologiques (quelles sont les informations essentielles ? sur quels résumés baser un modèle ? quelles sont les meilleures reformulations ?), et pose des soucis tout aussi dérangeants concernant l'évaluation des résultats, une question que nous abordons plus loin.

Compte tenu de ces difficultés, on peut légitimement douter de la réussite d'un projet aussi ambitieux que le RAT. Il est même très probable que les ordinateurs ne pourront jamais égaler l'homme dans cette matière, comme le suggère Torres-Moreno (2014 : 10), ou du moins pas avant un temps considérable. Dans cette optique, l'objectif du RAT serait de produire des résumés non plus équivalents, mais aussi utiles que ceux des humains, les systèmes de RAT devenant alors une alternative au résumé humain plutôt qu'un moyen de remplacer l'être humain.

2.2 Méthodologies

On définit généralement deux types de méthodes pour le résumé : l'approche abstractive, et l'approche extractive. La première produit le résumé en imitant le processus humain : on cherche à dégager le sens du texte puis à le reformuler en langage naturel. Les deux actions-clés sont donc la compréhension de document et la paraphrase. La seconde approche produit des résumés en extrayant les phrases les plus importantes, sans autre modification.

Ces deux approches ont été inégalement explorées par les chercheurs. Bien que l'extraction ait été envisagée bien avant l'abstraction, nous commencerons notre état de l'art du domaine du RAT par la description des tentatives faites dans l'approche abstraitive, car ce sont celles qui, de prime abord, semblent les plus adéquates pour la tâche étant donné leur similarité avec la méthode humaine. Nous verrons pourquoi les systèmes abstraectifs actuels paraissent en réalité inadaptés et peu efficaces, raison pour laquelle la recherche s'est focalisée sur l'extraction.

Une deuxième division de types de résumés est en général opérée dans la littérature. Elle sépare les résumés produits à partir d'un seul document (*single-document summarization* en anglais) de ceux générés pour plusieurs documents différents, mais portant sur le même sujet (*multi-document summarization*). Dans la mesure où les techniques de ces deux sous-ensembles s'avèrent relativement similaires et que nous nous focalisons sur le résumé uni-document dans ce travail, nous avons choisi de ne pas diviser l'état de l'art en fonction du nombre de documents sources, préférant envisager chaque algorithme que nous avons trouvé intéressant. Nous ne manquerons toutefois pas de signaler, en citant un système, s'il s'agit d'un résumeur uni-document ou multi-document.

2.3 Approche abstraitive

Le premier travail dans l'approche abstraitive ayant manifesté un intérêt important lors de sa parution en 1982 est celui de Dejong, dans lequel est présenté le système FRUMP. Dejong modélise la sémantique d'un texte et les connaissances extralinguistiques nécessaires pour le comprendre par des « scripts rudimentaires » (*sketchy scripts*), que l'on peut voir comme des scénarios qui préétabliissent les événements et les rôles potentiels d'une situation donnée⁵ (Dejong 1982 : 150). Il s'agit concrètement de 60 scripts décrivant des événements généralement relatés dans la presse, comme des catastrophes naturelles, le récit de manifestations, de guerres... Un premier module, appelé « prédicteur », s'occupe de sélectionner un scénario adéquat et de prédire, en fonction du contexte courant, les événements suivants dans le texte (par exemple, si l'on parle de l'arrivée de manifestants, on s'attend à ce que le texte renseigne leur nombre ou la réaction de la police dans les phrases suivantes). Un second, nommé « substantiateur », instancie les propositions du prédicteur en fouillant le texte (on cherche un mot qui correspond à un nombre ou un adjectif se rapportant à la police). Chaque prédiction instanciée alimente le contexte courant et force le prédicteur à affiner ses prochaines prédictions, jusqu'à ce que le script n'ait plus d'éléments non instanciés. L'initialisation du prédicteur se base sur les premières phrases du texte. FRUMP produit ainsi des résumés d'articles de presse, longs d'une à quelques lignes.

En 1998, Radev & McKeown présentent SUMMONS, un des premiers résumeurs abstraectifs multi-documents, qui suit fidèlement la méthode de FRUMP. Quelques années plus tard, White *et al.* (2001) reprennent aussi cette méthode, où l'utilisateur, en plus de fournir un ensemble de documents, choisit à l'avance un scénario pour le système, réglant ainsi le problème de l'initialisation. La structure du système multi-document SUMUM (Saggion & Lapalme 2002) s'avère similaire à celle de White *et al.*. On citera enfin Genest & Lapalme (2012), qui, eux, définissent des « schémas », dont les éléments sont instanciés par la reconnaissance de mots dans le texte. La reconnaissance se fait par règles et reconnaissance de patrons (*patterns*). Ils présentent un premier module d'extraction de l'information

5. Pour reprendre les mots de l'auteur : *Each sketchy script is the repository for the knowledge FRUMP has about what can occur in a given situation* (Dejong 1982 : 150).

qui propose des candidats, un second module de sélection de contenu, qui sélectionne le meilleur pour chaque catégorie, et un dernier module de génération de texte, tous implémentés manuellement.

Ce qui s'avère intéressant dans les techniques proposées d'un point de vue computationnel, c'est que le système, pour renvoyer un résultat, doit simplement choisir le bon scénario et le compléter avec les éléments du texte, sans être obligé de le parcourir en entier. L'opération de « lecture » du texte, représentée par le substantiateur ou le module de sélection de l'information, ne se fait que si elle est nécessaire et est arrêtée dès que l'on trouve un élément adéquat.

Toutefois, la méthode présente des insuffisances et désavantages considérables. Le système de scripts rudimentaires, qui doivent être écrits à la main, ne peut constituer une piste optimale pour le résumé car elle restreint le contenu traitable. Avec cette solution, on ne peut effectivement pas résumer des textes dont on n'a aucune idée du type de contenu ou du genre, sans compter que le vocabulaire des langues évolue avec le temps. Les scripts posent également des limites de génération de texte. La syntaxe des phrases produites est généralement déterminée à l'avance, et n'est pas forcément adaptée à tous les types de contenus. De façon générale, le style du résumé ne dépend que peu de l'information présente dans le texte original, ce qui n'est pas souhaitable pour une solution générique. Ainsi, à moins de trouver une méthode dynamique d'apprentissage automatique de scripts, aucune solution globale ne pourra être trouvée selon cette méthode. Elle nécessite par ailleurs l'usage de vocabulaires très étendus, accompagnés de modules robustes de désambiguïsation de mots. Cela tend à ralentir les capacités du système produit.

2.4 Approche extractive

Moins ambitieuse que l'abstraction, le résumé extractif se fait par repérage et sélection des segments de textes (généralement les phrases) les plus importants. Ce processus semble *a priori* s'éloigner fortement de la façon humaine de procéder puisqu'elle ne paraphrase pas, mais diverses études pointent des éléments allant dans le sens contraire. Copeck & Szpakowicz (2004) montrent par exemple qu'un grand nombre de syntagmes et mots sont repris exactement dans le résumé : dans leurs expériences, 55 % des tokens et 37 % des syntagmes (de tout type) de textes sources étaient en moyenne repris exactement dans les résumés humains. Le pourcentage monte jusqu'à 73 % pour les syntagmes partiellement copiés, et ce indépendamment de la taille du texte source (Copeck & Szpakowicz 2004 : 5-6). Même lorsque les phrases sources subissent des modifications conséquentes, Jing (2002) a montré d'autre part qu'il était possible de retrouver leur origine via des modèles de Markov cachés (HMM). Ainsi, on aurait tort de voir l'extraction de phrases représentatives comme un fourvoiement ou une restriction non pertinente du problème de RAT. L'homme réutilise – tantôt de façon exacte, tantôt en les modifiant – une très grande partie des expressions qu'il rencontre dans le texte, qu'il réassemble de façon organisée par la suite. Il hiérarchise également l'information qu'il lit et retranscrit cette hiérarchisation dans son texte abrégé. Les méthodes extractives entendent automatiser ces deux processus de hiérarchisation de l'information et de réemploi d'unités de discours, dans une approche du résumé qu'on peut certes qualifier de primaire, mais néanmoins tout à fait valable et essentielle dans la construction de résumés de qualité.

Un avantage de la méthode extractive par rapport à la méthode abstractive se situe dans le fait que la production du résumé ne se base sur aucun discours ou type de contenu prééta-

bli. Les informations sont directement retranscrites, sans passer par une étape complexe d'analyse ou d'identification. Elle peut donc s'appliquer à tout type de texte. Puisque les mots du texte sont directement repris, on évite en outre tout problème de génération de langage naturel.

L'objectif principal des algorithmes extractifs est d'attribuer un poids à chaque phrase en entrée pour effectuer un classement général des phrases dans leur ordre d'importance. Le résumé est composé des n premières instances de ce classement, n dépendant du taux de compression souhaité. Pour beaucoup de systèmes, cette pondération inter-phrase est obtenue via une pondération intra-phrase, c'est-à-dire l'attribution d'un poids aux mots ou syntagmes contenus dans les phrases. Cela s'effectue à l'aide de formalismes statistiques et/ou sémantiques. On peut représenter la méthode canonique de l'extraction par le schéma suivant (figure réalisée d'après Torres-Moreno 2014 : 31 ; les crochets désignent des étapes facultatives) :

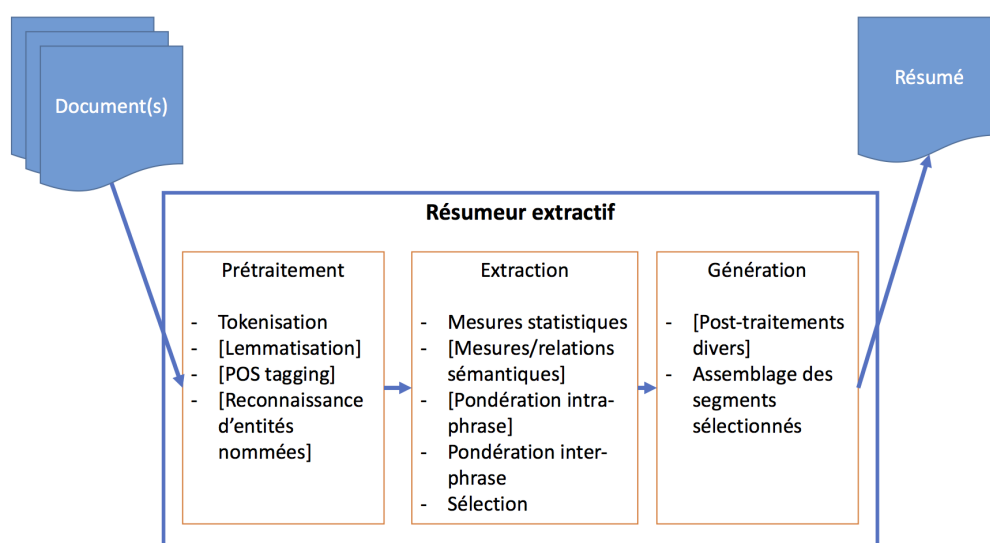


FIGURE 2.1 – Architecture générale d'un résumeur automatique extractif

Quels que soient le modèle et les mesures utilisées pour les pondérations – dont nous présentons la liste plus loin dans ce chapitre – une phase préalable de prétraitement du texte source est nécessaire.

2.4.1 Prétraitement

Dans l'approche extractive, le texte source est d'abord soumis à un prétraitement basique, qui consiste à délimiter les paragraphes, les phrases et les mots. Les tokens peuvent être lemmatisés avant d'être éventuellement tagués (POS, entités nommées...) (Torres-Moreno 2014 : 23-26). La seconde étape de prétraitement est l'attribution d'un poids à chaque token (voire syntagme), lequel sert à déterminer l'importance du mot dans le document. Cette mesure peut être de type binaire (présent ou absent d'un texte) ou fréquentiel (nombre d'occurrences), parfois pondéré (par exemple le $tf * idf$ ⁶). Ces calculs permettent, par la suite, de représenter chaque document sous la forme d'un espace vectoriel à deux dimensions, à savoir les mots et les phrases. La table 2.1 montre la matrice d'occurrences ré-

6. Pour une explication de cette mesure, consulter <https://fr.wikipedia.org/wiki/TF-IDF>.

sultante d'un document fictif contenant les phrases S_1 et S_2 suivantes : *Ceci est un exemple un peu simple. Simple, il l'est !*

	<i>ceci</i>	<i>est</i>	<i>exemple</i>	<i>il</i>	<i>l</i>	<i>peu</i>	<i>simple</i>	<i>un</i>
S_1	1	1	1	0	0	1	1	2
S_2	0	1	0	1	1	0	1	0
Total	1	2	1	1	1	1	2	2

TABLE 2.1 – Exemple de matrice d'occurrences

Ce prétraitement est commun à toutes les techniques extractives, à l'exception de certaines approches sémantiques, où la génération de matrices d'occurrences n'est pas nécessaire (cf. point 2.4.8).

2.4.2 Mesures statistiques de surface

Les premiers travaux en RAT, publiés dans les années 1950, se basent uniquement sur des mesures statistiques issues de l'analyse de surface du texte pour en sélectionner les phrases les plus importantes. Par « analyse de surface », nous désignons la récupération des valeurs qui, après un prétraitement du même type que celui décrit à la section précédente, apparaissent comme directement accessibles d'un point de vue informatique : nombre d'occurrences, position des mots et des phrases, longueur des phrases et du document... Les tout premiers résumeurs s'affranchissent en effet facilement des contraintes syntaxiques et sémantiques de la tâche de résumé, qu'il est, à l'époque, beaucoup plus difficile de prendre en compte. L'extraction est alors envisagée comme une solution avec laquelle il n'est pas nécessaire de se soucier de telles contraintes.

Un des premiers algorithmes établis est celui de Luhn (1958). Ce chercheur entend mesurer l'importance des phrases (*sentence signifiance*) d'un texte selon l'importance cumulée de ses mots. Celle-ci se calcule à partir des deux seules données suivantes : la fréquence brute, calculée sur la racine (*stem*) des tokens, et la position relative. Une phrase contenant des mots fréquents du texte et proches les uns des autres est ainsi considérée comme plus représentative qu'une autre n'en contenant aucun ou à des positions plus éloignées (Luhn 1958 : 160-161). On obtient alors un classement des phrases selon leur importance, et toutes les phrases mesurées au-dessus d'un certain seuil sont reprises pour former le résumé. Pour éviter que des mots fréquents non spécifiques au texte ne faussent le calcul, on utilise une liste de mots vides.

Avec son résumeur uni-document, Luhn (1958) ouvre la voie extractive en présentant des méthodes dont une partie de la littérature actuelle s'inspire encore. L'aspect purement statistique de la méthode et l'absence de modèle sémantique sont totalement assumés par Luhn à l'époque : « La méthode qui est développée ici est probabiliste et se base sur les propriétés physiques des textes écrits. Aucune attention n'est accordée au sens des mots ou aux arguments exprimés par leurs combinaisons. »⁷ (Luhn 1958 : 160). Les systèmes présentés dans les années qui suivent reprennent l'algorithme de Luhn (avec une version multi-document de Baxendale (1958)), en ajoutant des caractéristiques supplémentaires

7. [T]he method to be developed here is a probabilistic one based on the physical properties of written texts. No consideration is to be given to the meaning of words or the arguments expressed by word combinations.

pour le calcul de l'importance des phrases, comme leur position, leur longueur, la présence de noms propres, les mots contenus dans le titre... (Torres-Moreno 2014 : 62).

2.4.3 Mots-signaux

Un peu plus tard, on tente de repérer les séquences de mots importantes et non importantes en incluant la reconnaissance d'expressions donnant un indice quant à la représentativité des mots qu'elles contiennent. Ainsi, si une phrase contient un exemple introduit par la locution *par exemple*, on pourra lui attribuer une pénalité, puisque les exemples n'ont généralement pas leur place dans un résumé. On peut faire également le raisonnement inverse : des mots comme *essentiel* et *important* indiquent souvent une volonté de l'auteur d'insister sur ce qu'il est en train d'énoncer. Par exemple, l'algorithme d'Edmundson, présenté en 1969, mesure le poids d'une phrase $p(s)$ via la formule suivante :

$$p(s) = a_k K(s) + a_l L(s) + a_c C(s) \quad (2.1)$$

$K(s)$ vaut le nombre de mots jugés importants (c'est-à-dire dont le poids est grand), et $L(s)$ est la position de la phrase, les premières étant avantagées. $C(s)$ représente le nombre de « mots-signaux » (*cue-words*) présents dans la phrase. Ceux-ci sont des expressions qui manifestent de l'importance d'une séquence et sont compilés dans un dictionnaire créé manuellement (Edmundson 1969). Enfin, a_k , a_l et a_c sont des constantes entières, dont les valeurs peuvent être ajustées pour pondérer l'importance de chaque critère.

L'analyse de mots-signaux semblant plus performante, une partie de la recherche continue dans cette voie dans les années 70. Pollock & Zamora (1975) présentent ADAM, un résumeur extractif uni-document spécialisé dans le traitement d'articles scientifiques concernant la chimie. Par repérage de mots-signaux et une analyse syntaxique, il détecte les phrases importantes par élimination de phrases non pertinentes : si une instance contient des expressions introductives ou illustratrices (*en guise d'exemple, en conclusion...*) ou une construction avec virgule(s) introduisant des incises (appositions, relatives...), elle est automatiquement rejetée. Le fait de ne traiter qu'un domaine très spécifique permet d'obtenir plus facilement une liste d'expressions discriminantes et de réduire grandement les problèmes de synonymie et polysémie. Plus tard encore, Farzindar *et al.* (2005) implémentent un résumeur multi-document nommé CATS. Le poids des phrases est calculé à partir de comptages d'occurrences et d'entités nommées, de la position, de la similarité cosinus entre les phrases et une question ou un sujet guidant le résumé (donné en entrée), et du repérage de mots-signaux particuliers à l'anglais.

Toutefois, le caractère spécifique de la méthode représente aussi un grand désavantage à cause de la difficulté à la généraliser. Les mots-signaux comme la position des phrases importantes dépendent effectivement beaucoup du genre du texte : par exemple, si pour les articles de presse, les premières phrases ont une plus grande probabilité d'être représentatives, pour les articles scientifiques, ce sont plutôt les dernières (Torres-Moreno 2014 : 64). C'est pourquoi, dans les années 90, on se demande si l'apprentissage automatique (ou *machine learning*) peut résoudre le problème. Sur base de corpus tagués composés de textes relatifs à un domaine particulier et de leurs résumés, il est possible d'estimer la valeur des paramètres que l'on trouve pertinents pour l'extraction. S'il l'on reprend l'algorithme d'Edmundson, il s'agit donc de déterminer la valeur des constantes a_k , a_l et a_c . Kupiec *et al.* (1995) sont parmi les premiers à présenter un classificateur automatique qui reprend les paramètres de l'algorithme d'Edmundson. Quelques années plus tard, Lin (1999) utilise

un classificateur basé sur des arbres de décision, qui a été jugé le plus efficace dans sa catégorie à l'époque.

Après la méthode basée sur une analyse de surface combinée au repérage et l'interprétation de mots-signaux, on assiste dans les années 2000 à une large diversification des techniques extractives, notamment due à un intérêt général croissant pour le RAT. À partir des points suivants, l'état de l'art perdra donc de sa composante chronologique, car nous avons préféré structurer le propos méthodologiquement.

2.4.4 Approches probabilistes

Une série d'approches font appel aux outils de la statistique probabiliste pour résoudre le problème du résumé. La première que les chercheurs envisagent est celle des modèles de Markov cachés ou HMM, faisant l'hypothèse que l'inclusion d'une phrase dans un résumé dépend aussi de l'inclusion des phrases précédentes. Conroy & O'leary (2001 : 5-10) utilisent ainsi les mêmes paramètres qu'Edmundson pour son résumeur uni-document, qu'il entraîne sur un corpus aligné d'articles et de résumés manuels. Suneetha & Fatima (2011) emploient quant à eux le formalisme des HMM pour déterminer si un mot est important ou non, toujours suivant l'hypothèse que la probabilité qu'un mot soit essentiel au résumé dépend de celles des mots qui le précèdent.

Selon une approche probabiliste assez différente, on peut également songer à utiliser la divergence Kullback-Leibler, dans le but de classer les phrases d'un texte par ordre de représentativité. Cette mesure statistique quantifie la divergence entre deux distributions de probabilités (Kullback & Leibler 1951). Pour des distributions de probabilités discrètes P et Q , cette mesure s'exprime mathématiquement comme suit :

$$D_{KL}(P||Q) = \frac{1}{2} \sum_{m \in P} P(m) \log_2 \frac{P(m)}{Q(m)} \quad (2.2)$$

où P_m est le nombre d'occurrences du mot m divisé par le nombre total de mots au sein du document p . La valeur de ces probabilités peut être soumise à un certain lissage. On peut également utiliser la version symétrique de la divergence KL, à savoir la divergence Jensen-Shannon (D_{JS}) :

$$D_{JS}(P||Q) = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M); \quad M = \frac{P + Q}{2} \quad (2.3)$$

Dans le cas du résumé, on compare, pour chaque phrase, la probabilité de distribution entre cette phrase et toutes les autres phrases du document. On classe ensuite la valeur obtenue pour chaque phrase dans l'ordre croissant et l'on sélectionne les n premières, soit celles qui s'écartent statistiquement le moins de l'ensemble du document.

Torres-Moreno (2014 : 83) nomme cette technique DIVTEX, mais aucun système extractif finalisé portant ce nom n'a vu le jour. Bien que pertinente statistiquement, elle est en réalité peu envisagée, car on la réserve généralement pour la phase d'évaluation. Les divergences KL et JS ont en effet d'abord été développées dans ce cadre (cf. section 2.5.1), et ce avec des résultats concluants, comme le montre notamment l'étude comparative de Louis & Nenkova (2009).

2.4.5 Modèle vectoriel

Au même moment, une série de systèmes extractifs ayant recours au calcul vectoriel voient le jour (Torres-Moreno *et al.* 2002, Boudin 2008, Litvak *et al.* 2010, Torres-Moreno 2012).

Ces algorithmes reprennent les mesures statistiques de l'analyse de surface précédemment utilisées (occurrences et cooccurrences, position...), en abandonnant l'analyse des « mots-signaux ». On cherche en revanche à mesurer l'importance des phrases à partir de celles des mots, à l'aide du modèle vectoriel. Cette technique a été développée de façon quasi exclusive par la recherche française. Nous citerons d'abord deux systèmes uni-documents relativement reconnus : ARTEX et CORTEX.

ARTEX (Torres-Moreno 2012) vectorise le texte donné en entrée (matrice d'occurrences, cf. prétraitement) et calcule le vecteur moyen de ce texte, qui symbolise le sujet principal du texte. De là, en tenant compte de la longueur des phrases, le résumeur n'a plus qu'à attribuer des scores de proximité entre ce vecteur moyen et chaque phrase via une mesure d'angle, et extraire les phrases pour lesquelles l'angle est le plus petit. Lorsque des phrases sont trop proches les unes des autres, on en retient qu'une seule.

CORTEX (Torres-Moreno *et al.* 2002) combine quant à lui un ensemble de mesures statistiques issues de comptages, du calcul vectoriel et de la théorie de l'information pour trouver les phrases les plus pertinentes d'un texte. Parmi les mesures venant de la théorie de l'information⁸, on retrouve l'entropie (voir Shannon 1948 : 388) et diverses versions de la distance de Hamming. Nous ne retranscrivons pas la liste exhaustive parmi la quantité de mesures utilisées ; nous renvoyons le lecteur aux écrits de Torres-Moreno (Torres-Moreno *et al.* 2002, Torres-Moreno 2014 : 86-89). Pour évaluer le poids d'une phrase, un algorithme de décision prenant en compte les valeurs de la dizaine de mesures effectuées à partir de mots et phrases a été mis en place.

Dans le cadre du résumé multi-document, Boudin (2008 : 47-63) présente NEO-CORTEX, une version de CORTEX adaptée au traitement d'un ensemble de documents. Pas moins de treize paramètres y sont pris en compte dans la pondération de phrases.

2.4.6 Mesures de similarité inter-phrases

En complément à ces mesures statistiques de l'importance d'un mot et d'une phrase, une série de méthodes plus complexes mesurant des taux de similarité entre les phrases ont également vu le jour. L'objectif est d'identifier, grâce à ces mesures, des groupes de phrases similaires du point de vue de leur contenu au sein d'un seul ou plusieurs documents, afin d'extraire la plus représentative de chacun d'entre eux.

Une première approche est celle de l'analyse sémantique latente (ou LSA pour *Latent Semantic Analysis*). Publiée pour la première fois en 1990 par Deerwester *et al.*, cette technique établit à l'aide du calcul vectoriel des relations de similarité sémantique entre des termes et des phrases ou des documents les contenant. Cela se réalise par la décomposition en valeurs singulières (DVS) (souvent notée SVD pour *singular value decomposition*). L'idée est de définir mathématiquement, à partir d'une matrice d'entrée, un nombre défini de dimensions latentes ou de concepts qui permettent de représenter cette même matrice, mais avec un nombre moindre de dimensions. Dans notre cas, les dimensions latentes de la matrice d'occurrence sont des groupes de phrases où chaque groupe lie un ensemble de phrases donnant des informations similaires. Pour le résumé, on choisit typiquement entre 100 et 300 dimensions latentes ou concepts (Torres-Moreno 2014 : 73). Algébriquement, si on a s phrases et n termes et que l'on recherche d dimensions latentes, le théorème de DVS approxime la matrice d'occurrence S en trois matrices de taille réduite selon l'équa-

8. Théorie probabiliste de l'information développée à partir de 1948 par Shannon. Pour plus d'informations, voir (Shannon 1948).

tion suivante :

$$S_{[n \times s]} = U \Sigma V^T \quad (2.4)$$

où $U_{[n \times d]}$ est la matrice gauche de vecteurs singuliers qui donne pour chaque terme un nombre indiquant le poids de la relation entre celui-ci et chaque concept ; $\Sigma_{[d \times d]}$, la matrice des valeurs singulières, est une matrice diagonale qui classe le poids de chaque concept dans l'ordre décroissant ; et $V_{[d \times s]}$ est la matrice droite transposée de vecteurs singuliers et renseigne pour chaque phrase, le poids de son lien avec chaque concept. Avec les matrices Σ et V^T , Gong & Liu (2001) résument par exemple un texte en évaluant la valeur de chaque vecteur de V^T et en incluant la phrase correspondant au vecteur dont la valeur est la plus grande pour un concept donné, jusqu'à atteindre la taille requise.

D'autres travaux reposent leur approche de la classification des phrases sur la manipulation de graphes. Les nœuds de ces graphes sont les phrases (éventuellement les documents), et leurs arêtes dénotent leur similarité. L'extraction se fait ici en deux grandes étapes : d'abord, créer le graphe et mesurer la similarité entre les différents nœuds, puis implémenter un algorithme d'analyse de liens ou d'identification de groupes permettant de repérer les n phrases les plus significatives (Torres-Moreno 2014 : 79). Cette méthode est motivée par l'abondante littérature mathématique et algorithmique sur l'analyse de liens dans les graphes pondérés (voir le livre de référence Ore & Ore 1962), et, plus particulièrement, sur la notion de « centralité ». Cette notion, créée dans les années 1970 (voir l'article fondateur de Freeman 1977), sert à repérer les nœuds les plus importants d'un graphe selon diverses mesures (centralité de degré, de proximité, d'intermédiation...). L'utilisation de graphes a également l'avantage de ne pas avoir de phase d'apprentissage et se dispense donc de corpus d'entraînement.

La construction du graphe requiert l'usage de mesures de similarité entre les phrases, qui déterminent la valeur de chaque arête du graphe. Il s'agit généralement de la similarité cosinus, où les phrases sont représentées par des vecteurs ou sacs de mots (*bag-of-words*), ou du nombre de mots, stems ou lemmes communs. Bien que généralement le graphe ne soit pas dirigé, on peut donner des directions aux arêtes, selon qu'une phrase précède ou suit celle avec laquelle elle est liée. Une fois que le graphe est créé, on attribue un poids à chaque phrase avant d'obtenir le classement des phrases du point de vue de leur centralité (c'est-à-dire en fonction de leur importance ou représentativité). Les pondérations de phrases les plus connues sont celles de LEXRANK (Erkan & Radev 2004) et TEXTRANK (Mihalcea & Tarau 2004). Le premier de ces deux résumeurs extractifs base le calcul du poids sur la similarité cosinus pondérée par la fréquence inverse du document (*idf*), là où le second le base sur le nombre de mots communs entre deux phrases, en pénalisant les longues phrases. Les deux équations du poids $p(s)$ sont récursives, et, par souci de clarté et parce que nous n'utiliserons pas cette technique dans notre projet, nous renvoyons le lecteur en annexe pour une retranscription complète de ces équations.

Une autre approche moins formelle est celle développée par Goldstein & Carbonell (1998, 2000), dans le cadre du RAT multi-document. Le calcul de similarité inter-phrase se base sur des comptages d'occurrences et un regroupement de phrases au contenu similaire entre les documents (voir les équations complètes dans Goldstein *et al.* 2000 : 44). Les mesures de similarité sont par la suite prises en compte dans le calcul de la « pertinence marginale maximale ». Cette autre mesure, issue du domaine de l'extraction d'information, définit en termes mathématiques les éléments des textes sources (les phrases) qui, d'une part, maximisent la pertinence de leur contenu, et, d'autre part, en minimisent la redondance.

On citera enfin le résumeur multi-document MEAD qui mesure également un taux de similarité inter-phrases. Il compare la similarité de documents avec un ensemble de mots considérés comme thématiques (le « centroïde »), et tient compte de la position des phrases. Ces trois paramètres sont pondérés pour ensuite donner le poids final de chaque phrase (Radev *et al.* 2004, Torres-Moreno 2014 : 123-126).

2.4.7 Réseaux de neurones

Une dernière avancée dans l'extraction statistique réside dans l'utilisation des réseaux de neurones ou ANN (*artificial neural networks*) pour résoudre le problème de classement des phrases.

Kaikhah présente, dès 2004, un résumeur qui prend un document en entrée, en extrait 7 paramètres et l'envoie dans un réseau de neurones entraîné spécifiquement pour la tâche. Les paramètres qu'il choisit pour l'entraînement du réseau renseignent la position de la phrase, sa longueur et les mots qu'elle contient (Kaikhah 2004 : 41). On entraîne le réseau de neurones, composé de 3 couches, grâce à des corpus annotés manuellement, puis on élimine les poids synaptiques et/ou neurones dont l'influence est faible. Une fois l'apprentissage effectué, le réseau devient comme un « filtre à résumé » qui, pour tout document en entrée, ne laisse passer que les phrases les plus représentatives.

Svore *et al.* (2007) produisent NETSUM, un autre résumeur neuronal uni-document, basé sur un seul paramètre pour discriminer les phrases entre elles : leur score ROUGE⁹. Un corpus d'entraînement composé de documents d'articles de presses, de leurs résumés extractifs et du score ROUGE de chaque phrase est donné en entrée de RANKNET (Burges *et al.* 2005), un système qui crée et entraîne automatiquement un réseau de neurones à deux couches (sans couche cachée).

Plus récemment, Torres-Moreno implémente un système qu'il nomme ENERTEX, qui calcule le score de similarité en prenant en compte un ensemble de poids synaptiques présents dans un champ local de neurones. Un champ local de neurones comprend l'ensemble des poids synaptiques entre un neurone et tous les autres (Torres-Moreno 2014 : 93-95). Suivant la théorie physique du spin sur laquelle s'appuie la notion de poids synaptique, l'on peut quantifier l'énergie d'un champ de neurones, qui est la somme cumulée de ses poids synaptiques et de la valeur de chaque neurones. Dans le cadre du traitement de texte, le poids synaptique P d'une phrase peut se calculer à partir de sa matrice d'occurrences S selon la formule $P = S^T \times S$, où S^T est la matrice transposée de S . Dès lors, ce qu'on appelle « l'énergie textuelle » s'obtient ainsi : $E = -\frac{1}{2}(S^T \times S)^2$ (Torres-Moreno 2014 : 95-97). Il existe également une version multi-document d'ENERTEX. Sur base d'un sujet donné, ce système mesure l'énergie textuelle des phrases de tous les documents en les plaçant dans l'ordre chronologique (du point de vue de leur contenu). Le jugement de la pertinence et de la redondance de l'information se base sur cette seule quantité (Torres-Moreno 2014 : 121-123).

2.4.8 Modèles sémantiques pour l'extraction

Jusqu'ici, nous nous sommes attardé sur les systèmes de résumés essentiellement statistiques, c'est-à-dire ne prenant en compte que des éléments du texte issus de l'analyse de

9. Mesure utilisée normalement dans le cadre de l'évaluation du résumé. Voir section 2.5.1 pour sa définition.

surface, paramètres d'un modèle probabiliste ou statistique plus ou moins complexe. Il convient à présent d'évoquer des systèmes travaillant avec des unités plus abstraites qu'un lemme, sa position et son nombre d'occurrences dans les phrases, à savoir des unités sémantiques. D'une façon similaire, mais bien plus complexe qu'Edmundson et ses mots-signaux, l'application de modèles sémantiques pour l'extraction repose sur l'idée que les relations de sens entre les unités d'un texte jouent un rôle primordial dans leur présence ou absence dans un résumé de ce texte. Deux modèles sémantiques ont principalement été exploités par les ingénieurs linguistes concernant l'extraction : les chaînes lexicales et la structure rhétorique.

Chaînes lexicales

Le paradigme des chaînes lexicales, développé par l'article de Morris & Hirst (1991) d'après la théorie de la cohésion lexicale de Halliday & Hasan (1976), propose de lier des segments de textes à travers des relations sémantiques, identifiées par l'exploitation de thésaurus. Les unités considérées sont les verbes (excepté les auxiliaires) et les noms. Une relation est créée entre deux unités lorsqu'un lien est trouvé lors de la fouille du thésaurus. Morris & Hirst (1991 : 30) distinguent 5 types de liens différents : des liens de synonymie (*internet-web*), d'hyponymie (*arbre-végétal*), d'hyponymie (*métier-facteur*) ou simplement de liens de contiguïté (*conduire-voiture*). Ces relations sont éventuellement pondérées, généralement de façon discrète.

Le résumé par chaînes lexicales sélectionne simplement les phrases possédant le plus de relations fortes parmi tout le texte (Barzilay & Elhadad 1997, Brunn *et al.* 2001, Silber & McCoy 2002). Le thésaurus le plus utilisé pour l'anglais est WordNet (Miller 1995), une base de données pour les relations sémantiques dont la taille n'a pas d'équivalent à l'heure actuelle. Pour le français, on note le travail de Minel *et al.* (2001), qui, eux, construisent les relations à partir de la reconnaissance « d'indices et indicateurs », équivalents des mots-signaux d'Edmundson.

Cette technique semble aujourd'hui avoir été délaissée, aux faveurs d'une autre méthode sémantique pour le résumé, se basant sur la « structure rhétorique ».

Structure rhétorique

La théorie de la structure rhétorique (souvent abrégée RST pour *Rhetorical Structure Theory*) a été développée par Mann & Thompson (1988). Elle définit tout texte comme une série d'unités discursives imbriquées dans un arbre rhétorique hiérarchique. Les unités élémentaires de ce système, les unités discursives (*Elementary Discourse Units*), peuvent être des noyaux ou des satellites. Les premiers représentent les constituants sémantiques principaux du texte, portant une information pertinente par rapport à son propos, tandis que les seconds sont des unités supplémentaires non obligatoires qui sont reliées à un noyau par une relation rhétorique. La RST étant hiérarchique, un couple noyau-satellite ou deux noyaux distincts peuvent constituer un noyau pour un autre noyau ou satellite à un degré supérieur de l'arbre rhétorique. Les relations rhétoriques représentent les grands types de relations que l'on peut signifier dans un discours, telles que la concession, l'illustration, l'exemple, la reformulation... (Mann & Thompson 1988, Molina Villegas 2013 : 33). Pour les détecter, on s'aide de marqueurs discursifs (conjonctions et prépositions comme *bien que*, *par exemple*, *pendant que*...), d'éléments contextuels et/ou du résultat d'un apprentissage automatique.

L'intérêt de la RST réside dans le fait que le modèle distingue dès le début du processus les unités principales (les noyaux) de celles qui n'apportent qu'une information additionnelle (les satellites), et ce, d'un point de vue *sémantique*. Ce traitement donne un attrait aux arbres rhétoriques dans un contexte où l'on cherche à supprimer des segments de texte non représentatifs, redondants ou inutiles par rapport à son propos principal.

Les premiers à avoir utilisé le modèle RST pour le RAT sont Ono *et al.* (1994) pour le japonais et Marcu (1997, 2002) pour l'anglais. Les systèmes qu'ils implémentent se basent sur le repérage de marqueurs discursifs pour identifier les unités élémentaires, qui sont ici les phrases. À partir des arbres rhétoriques, on donne un poids à chaque phrase, en fonction de sa hauteur dans l'arbre et de sa nature (noyau ou satellite), pour ensuite sélectionner les phrases les plus lourdes du classement. La méthode décrite dans les articles de Ono *et al.* (1994) et Marcu (1997) a été reprise dans un certain nombre d'études (O'Donnell 1997, Teufel & Moens 2002, Hachey & Grover 2004, Yoshida *et al.* 2014).

La technique a également été exploitée pour le résumé multi-document par Radev (2000). Il a développé une adaptation de la RST appliquée à plusieurs documents, qu'il a nommée « structure inter-document » (*Cross-document Structure Theory*). Dans cette théorie, on peut joindre par une relation rhétorique une unité élémentaire d'un texte à une unité venant d'un autre texte (pour autant que ces deux documents traitent du même sujet).

2.5 Évaluation

La question de l'évaluation s'avère particulièrement complexe pour le RAT, en grande partie à cause de l'impossibilité d'élire objectivement un résumé idéal pour un document donné. Pour autant, tous les résumés – et donc tous les résumeurs automatiques – ne se valent pas, raison pour laquelle on ne peut se passer de moyens d'évaluation. Les mesures d'évaluation les plus connues pour le RAT ont d'ailleurs été développées dans le cadre de conférences, ateliers et concours scientifiques, où discriminer les systèmes de façon objective relève de la nécessité.

Il y a traditionnellement deux approches possibles : l'évaluation intrinsèque et l'évaluation extrinsèque. La première juge directement les performances du résumeur, en donnant une note au résumé produit selon des critères linguistiques et sémantiques. La seconde évalue le résumeur sur sa propension à améliorer le résultat d'une autre tâche du TAL. Il s'agit généralement des tâches de questions-réponses ou de recherche ou d'extraction d'information : si le système qui effectue une de ces tâches externes au RAT obtient de meilleurs résultats en disposant en entrée des résumés à la place ou en supplément des textes sources, alors le résumeur évalué recevra un bon score. En d'autres termes, l'approche intrinsèque atteste de la qualité du résumé en tant que produit final, tandis que l'approche extrinsèque atteste plutôt de son utilité en tant qu'outil pour d'autres tâches automatiques. Il existe des variantes manuelles, semi-automatiques et entièrement automatisées pour chacune des approches, que nous présentons juste après.

Dans tout type d'évaluation, on s'attend au minimum à ce que la qualité ou l'utilité d'un résumé produit surpassent celle de résumés de base (ou *baseline*). Ce sont des résumés extractifs obtenus en utilisant des règles très simples, telles que :

- Extraire aléatoirement n phrases ou syntagmes.
- Extraire les n premiers ou derniers mots ou phrases.

- Extraire les $n/2$ premières phrases et les $n/2$ dernières phrases.

Il arrive aussi, généralement dans le contexte de conférences scientifiques, que la référence à surpasser soit tout simplement un autre résumeur automatique.

2.5.1 Évaluation intrinsèque

Évaluation manuelle

L'évaluation d'un résumé selon ses caractéristiques intrinsèques s'est tout d'abord développée manuellement, notamment sous l'influence des conférences scientifiques SUMMAC (1998) (*Summarization Conference*), DUC¹⁰ (*Document Understanding Conference*) et TAC¹¹ (*Text Analysis Conference*).

La méthode d'évaluation utilisée dans les ateliers DUC et TAC depuis 2005 nécessite un ensemble de juges qui apprécient (avec une note sur 5) les sept caractéristiques suivantes :

- La grammaticalité
- La non-redondance (tant d'un point de vue lexical qu'informatif)
- La clarté des références : absence d'anaphores non résolues, d'ambiguïtés pronominales...
- L'emphase (*focus*) : un seul sujet est traité, et toutes les phrases ont un lien direct avec lui.
- La structure et la cohérence du texte
- Le dynamisme (*responsiveness*) du contenu : proportionnel à la quantité d'informations utiles.
- La qualité informative générale

Les cinq premiers critères jugent plutôt la forme du texte, tandis que les deux derniers s'attardent sur le contenu. On remarque le caractère ambitieux d'un tel type d'évaluation, puisque le résumé produit automatiquement est jugé selon des critères utilisables pour des résumés écrits par l'homme, que l'on considère ici comme modèles. Cette technique permet ainsi d'apprécier au mieux la similitude entre ces deux types de résumés. Toutefois, elle a pour désavantage principal de demander un temps et des moyens humains considérables. Sa validité se trouve également fortement diminuée lorsque l'accord inter-annotateur est faible.

Évaluation semi-automatique

Pour éviter cet écueil et réduire le temps d'évaluation, des chercheurs ont développé des méthodes dites semi-automatiques, basées sur des techniques d'extraction de documents et de l'information. L'idée est de calculer par voie automatique le taux de recouvrement (en mots ou en phrases) entre un résumé candidat et un ou des résumé(s) de référence écrit(s) par des humains. Cela permet alors de fournir des scores de rappel, de précision et F_1 , mesures classiquement utilisées en TAL.

10. Voir <http://www-nlpir.nist.gov/projects/duc/index.html>

11. Voir <https://tac.nist.gov/about/index.html>

Le travail ayant popularisé cette approche est celui de Lin, publié en 2004. Il y présente ROUGE (acronyme de *Recall-Oriented Understudy for Gisting Evaluation*), un ensemble de mesures d'évaluation semi-automatiques inspiré des techniques d'évaluation de la traduction automatique. Le taux de recouvrement y est calculé à partir d'un comptage de cooccurrences de n -grams (ROUGE- N_n), de la taille des séquences de mots communes (ROUGE-L et ROUGE-W) ou de *skip*-bigrams (ROUGE- S_n et ROUGE- SU_n), un *skip*-bigram étant deux mots qui se suivent à 0 ou n mots près dans la phrase (Lin 2004 : 4-5). Lin a démontré la pertinence de ROUGE en mesurant sa corrélation avec les jugements humains. Pour les résumés de petite taille, il obtient des corrélations de Pearson allant jusqu'à 0,99 et 0,9 pour des résumés multi-documents (Lin 2004 : 6-7). Les mesures les plus utilisées de nos jours sont ROUGE-2 et ROUGE-SU4.

Plus rapide que l'approche intrinsèque manuelle, ROUGE ne demande aucune manipulation humaine lorsque l'on dispose déjà d'un résumé de référence (comme c'est le cas pour des articles scientifiques ou certains articles de presse). Néanmoins, la technique n'est pas exempte de défauts. Tout d'abord, en ne prenant en compte que les occurrences des mots, elle ne compte pas comme bons les éléments repris par des synonymes ou des paraphrases, figures de styles pourtant propres au résumé. Et, inversement, les unités polylexicales ont un poids trop important. D'autre part, de façon plus inquiétante, Sjöbergh (2007) a pointé une faille dans le système d'évaluation, en démontrant empiriquement la possibilité de truquer les résultats. En soumettant des résumés contenant les bigrams les plus courants de leurs textes sources, on obtient en effet de très bons scores ROUGE, même lorsque les textes ainsi produits sont incohérents et agrammaticaux.

Cette faille appelle donc à utiliser pour le comptage d'occurrences des unités plus abstraites que les tokens. Donaway *et al.* (2000) avaient déjà proposé des méthodes prenant en compte la synonymie. Dans leur étude comparative, ils consultent pour ce faire un thésaurus avant de calculer la similarité de contenu entre deux documents grâce au modèle vectoriel.

Actuellement, deux systèmes plus récents font autorité dans l'approche semi-automatique : BASIC ELEMENTS et PYRAMID. Les mesures BASIC ELEMENTS (Hovy *et al.* 2006) reprennent l'idée de comptage de n -grams, mais l'effectuent sur base d'unités plus abstraites que le mot tel qu'il est présent dans un texte source. Grâce à des arbres de constituants et diverses informations sémantiques, BASIC ELEMENTS pondère mieux les mots composés et gère une partie des phénomènes de synonymie et paraphrase, tout en produisant des résultats similaires à ROUGE (une forte corrélation statistique entre les deux mesures a été mise en évidence (Hovy *et al.* 2006 : 3)).

Le populaire PYRAMID, ensemble de mesures créées par Nenkova & Passonneau (2004, 2007), entend également compter les occurrences d'unités sémantiques, mais de façon légèrement différente. Les mots du texte source sont rassemblés en séquences en fonction de leur contenu, auxquelles on attribue un poids en fonction de leur taille et de leur fréquence relative. On obtient ainsi un classement pyramidal des « unités de contenu », les unités les plus présentes et les plus longues (peu nombreuses) se retrouvant en haut de la pyramide et les unités composées d'un ou plusieurs mots apparaissant rarement. Avec ce classement, on peut attribuer un poids à chaque résumé à évaluer, qui dépend du nombre d'unités de contenu communes entre la source et le résumé et de leur position dans la pyramide. Pour obtenir un chiffre entre 0 et 1, on compare le poids cumulé de chaque unité du résumé avec celui de la source.

Évaluation complètement automatique

Si l'on veut se passer de l'humain dans la phase d'évaluation pour le résumé, on ne peut plus s'appuyer sur des résumés de références, dont les mesures précédemment présentées sont dépendantes. Nous n'avons donc à notre disposition que le(s) texte(s) source(s) et le résumé à évaluer. Les méthodes automatiques d'évaluation s'attachent dès lors à juger un résumé en fonction de sa ressemblance avec sa source.

Mathématiquement, cette ressemblance s'évalue avec le calcul de la divergence de probabilités de distribution : la distribution des mots dans le résumé avec celle des mots de la source. La notion de divergence de probabilités de distribution vient de la théorie de l'information et peut être calculée selon la formule de Kullback et Leibler ou de Jensen et Shannon (voir la section 2.4.4 pour les équations de ces deux mesures).

Cette technique a initialement été développée selon l'approche semi-automatique, c'est-à-dire en utilisant des résumés de référence au lieu du texte source (Donaway *et al.* 2000, Hovy *et al.* 2006). Néanmoins, plusieurs expériences ont montré l'intérêt de comparer le résumé directement avec sa source. Tout d'abord, Louis & Nenkova (2009) ont produit une étude comparative, dans laquelle ils mettent au jour de fortes corrélations entre les divergences KL et JS et PYRAMID, avec, pour la divergence JS, une corrélation de Spearman autour de 0,8 quand ROUGE avait une corrélation de 0,92. Plus tard, Torres-Moreno *et al.* (2010) développent FRESA (*Framework for evaluating summaries automatically*), un package pour l'évaluation automatique de résumés en anglais, français, espagnol et catalan. Les probabilités de distribution se basent sur les n -grams. FRESA s'avère hautement corrélé avec ROUGE, surtout pour les résumés uni-documents en français (Torres-Moreno 2014 : 268-269).

2.5.2 Évaluation extrinsèque

Comme pour l'évaluation intrinsèque, on peut envisager pour la voie extrinsèque des méthodes qui incluent un juge humain ou non. Dans les deux cas, il s'agira de tester l'utilité d'un résumé dans une tâche externe au RAT par rapport à des résumés de base ou au(x) texte(s) source(s).

La conférence SUMMAC (1998) présente une des premières méthodes extrinsèques manuelles développées de façon indépendante. Parmi les trois tâches de la conférence, deux jugent un résumé sur son influence sur des tâches d'extraction de documents ou d'information. La tâche *ad-hoc* demande à des juges humains leur avis sur la pertinence d'un résumé en fonction d'un sujet donné, en sachant que ce sujet était au préalable donné en entrée du résumeur. Dans la tâche de catégorisation, les juges doivent indiquer si un résumé est pertinent pour un ou plusieurs sujets parmi un ensemble prédéfini. Dans le processus, les textes sources côtoient les résumés, de sorte que le juge ne sait pas s'il a affaire à l'un ou l'autre lorsqu'il effectue une tâche (SUMMAC 1998 : 4-5).

On note également la méthode de « prédiction de la pertinence » (*Relevance-Prediction*) de Dorr *et al.* (2005), où les résumés sont notés selon des critères de la tâche d'extraction de documents. Un ensemble de juges évaluent de façon binaire la pertinence d'un document (texte source ou résumé) par rapport à un sujet donné. De là, on peut mesurer la pertinence moyenne d'un résumé en voyant si, en moyenne, le résumé et le texte source ont été jugés pertinents pour un même sujet.

Concernant les approches extrinsèques automatiques, on reporte tout simplement le résultat d'un test automatique conçu pour évaluer les performances d'un système externe au résumeur envisagé. En guise d'exemple, on citera le travail de Hirao *et al.* (2001), qui justifient l'intérêt de leur résumeur pour des systèmes de questions-réponses.

2.6 Limites de l'extraction de phrases

L'approche extractive au résumé, bien qu'elle semble plus prometteuse que l'approche abstractive, n'est pas dénuée de défauts. Parmi les problèmes récurrents, on mentionne souvent le manque de cohérence et de cohésion, ainsi que le traitement des phrases longues. En plus de ces problèmes plutôt techniques, l'extraction de phrases se révèle également intrinsèquement insatisfaisante.

2.6.1 Cohésion et cohérence

Dans son état de l'art sur le RAT, Torres-Moreno (2014 : 39-40) met en exergue le manque de cohésion et de cohérence des résumés extractifs. Si l'on reprend les explications de Halliday & Hasan, la cohésion désigne l'ensemble des « relations de sens qui existent dans le texte » (Halliday & Hasan 1976 : 4), soit tous les éléments à l'intérieur du texte qui relient sémantiquement les énoncés entre eux. Selon Halliday & Hasan, c'est le phénomène de cohésion qui différencie un amas de phrases quelconques d'un texte. Apothéloz (1995) signale toutefois que la seule cohésion ne suffit pas pour qualifier une séquence de phrases de texte, car elle doit également faire preuve de cohérence. « La cohérence résulterait ainsi de la capacité du destinataire à fonder son texte sur un savoir partagé, en sorte que le destinataire dispose de tous les éléments lui permettant de reconstruire l'intention de signification qui était au départ du texte. » (Apothéloz 1995 : 10). En d'autres termes, la cohérence renvoie à la pertinence du contenu des phrases entre elles, à leur « raison d'être ». Dans le cas du résumé, on peut la situer, comme le fait Torres-Moreno (2014 : 41), dans l'absence de contradictions et de redondance entre les phrases du résumé.

La cohésion d'un texte dont on veut supprimer des phrases se maintient difficilement à l'extraction parce que chaque phrase est susceptible de contenir des éléments nécessaires à la fluidité du texte. Par exemple, supprimer des phrases contenant des marqueurs de discours (tels que *cependant*, *malgré*, *ensuite*, etc.) peut nuire à l'enchaînement logique des informations d'un texte, et modifier la structure originale du discours, et ce même si ces phrases ne reprenaient pas des informations essentielles du texte. Un autre phénomène qui garantit la fluidité d'un texte est l'anaphore. Si l'on reprend la terminologie française de Apothéloz (1995 : 17-78), le terme *anaphore* désigne une expression référentielle non autonome dont la source est endophorique. Elle est une expression référentielle non autonome, car elle renvoie à un signifié (concept désigné par une unité lexicale) qui n'est pas directement exprimé par son signifiant (représentation graphique ou orale de l'unité lexicale). Pour illustrer cela, comparons les deux phrases d'exemple suivantes :

Un garçon voit une pomme sur la table. Il la mange.

Les syntagmes *un garçon* et *une pomme* sont des expressions qui font référence à une entité existante (c'est-à-dire qu'elles sont référentielles) et dont l'identité est révélée par leur signifiant même. En revanche, on ne peut pas en dire autant pour le pronom *la* de la deuxième phrase : pour établir la nature de son signifié (c'est-à-dire la pomme qu'un garçon a vue), il faut savoir quel est l'antécédent du pronom ou référé. Pour cela, on a besoin de connaissances grammaticales (*la* ne peut que référer à un nom féminin), mais

aussi sémantiques (entre la pomme et la table, seule la première se mange). *La* est donc dans ce texte une expression référentielle non autonome. Sa source est dite *endophorique* ou intra-textuelle car son référé se trouve dans le texte¹².

Par ailleurs, la résolution d'anaphore ne concerne pas que le RAT, et cette tâche constitue un domaine à part du TAL, aussi vieux que le RAT (voir Mitkov 2014). Ainsi, pour résoudre ces problèmes de coréférence, les résumeurs peuvent compter sur des systèmes de résolution d'anaphores comme celui de Stanford (Lee *et al.* 2013).

Le maintien de la cohérence se situe quant à lui au niveau du contenu du texte et des liens entre les différentes informations qu'il contient. Il n'est pas acceptable, dans un résumé, de trouver des informations contradictoires là où le texte original n'en présente pas. L'autre enjeu de cohérence essentiel pour le résumé est d'éviter la redondance de l'information, et ce même lorsqu'elle est expliquée de façon différente (avec des mots ou structures de phrase différents).

2.6.2 Traitement des longues phrases

Un certain nombre de résumeurs extractifs ont préférentiellement tendance à sélectionner les phrases les plus longues dans un texte, comme l'expliquent Gagnon & Da Sylva (2005 : 1). Ce défaut surgit pour les approches statistiques les moins abstraites (voir section 2.4) : étant donné qu'on repère les mots ou concepts les plus cités du texte source ou qu'on recherche les phrases qui contiennent le plus d'information pour sélectionner les phrases les plus importantes, celles qui ont le plus de mots ont statistiquement plus de chances d'être extraites. Plusieurs systèmes ont tenté d'éviter cet écueil en pénalisant les phrases trop longues (notamment TEXTRANK et SUMMONS). Toutefois, les résumeurs statistiques, lorsqu'ils sont confrontés à un conflit entre importance et taille de la phrase, ne disposent bien souvent d'aucun moyen extérieur (informations sémantiques, contextuelles...) pour prendre la bonne décision, contrairement à l'homme.

De façon générale, les longues phrases constituent un problème pour la méthode extractive. Lorsqu'une longue phrase est incluse dans le résumé, elle réduit drastiquement la place restante et peut empêcher la sélection de plusieurs autres phrases. Ce genre de phrase donne également de mauvais résultats au niveau du style du texte final, en particulier dans le genre du résumé, où l'on attend un discours plus direct et précis qu'un texte original. Pour autant, rejeter les phrases longues de façon systématique ne peut être accepté sur le plan théorique.

2.6.3 Insuffisance intrinsèque de l'extraction

En plus de ces défauts pratiques, l'extraction de phrases pure ne suffit pas à l'élaboration de résumés. En fait, cette opération est plus proche de celle du surlignage que l'ont fait à la lecture d'un livre ou d'un article que celle du résumé abstraktif. Dès lors que l'objectif dépasse le « résumé par surlignage » (ou *highlight* en anglais), on ne peut théoriquement se contenter de l'extraction de phrases, quand bien même elle produirait un résultat cohésif et cohérent.

12. Les anaphores s'opposent sur cette caractéristique aux expressions dites *déictiques*, dont la source est *exophorique* ou extratextuelle. Les déictiques sont des expressions référentielles dont le référé dépend de la situation d'énonciation ou *deixis*, sa nature étant donc souvent implicite. Typiquement, il s'agira d'unités comme *je*, *ici*, *demain*...

C'est du moins de ce qu'a démontré l'expérience HEXTAC réalisée en 2009-2010 par Genest *et al.* (2010). Ceux-ci ont, dans le cadre de la conférence TAC 2009, comparé des résumés de plusieurs types afin de mettre en évidence une certaine « limite supérieure aux performances » de résumeurs extractifs (Genest *et al.* 2010 : 1). Un corpus de résumés produits par extraction manuelle a été constitué par les chercheurs à partir du corpus de test de TAC 2009, puis évalués par les experts de la conférence. En comparant les notes obtenues pour ce corpus avec celles du corpus de référence de la compétition TAC (résumés manuels des mêmes textes, mais produits sans contrainte de rédaction), il ressort que les résumés humains par extraction sont de qualité nettement inférieure que ceux produits de façon libre (Genest *et al.* 2010 : 9).

Ces résultats ne suggèrent pas forcément qu'il faille retourner à la méthode abstractive. Ils prouvent simplement qu'on ne peut pas réduire la tâche du RAT à celle de l'extraction de phrases, comme la recherche a eu tendance à le faire en évaluant des résumés extractifs sur base de résumés paraphrastiques. Ils poussent à envisager l'extraction comme la première étape d'un processus plus complexe. Depuis les années 2000, on considère les tâches de compression et fusion automatiques de phrases comme les étapes suivantes (Genest *et al.* 2010 : 9), rapprochant le résumé extractif de l'abstraction.

Conclusion

Dans ce chapitre, nous avons d'abord introduit le RAT en tant que domaine du TAL. On a vu l'intérêt de disposer de systèmes capables de résumer du texte : rendre plus efficace la recherche automatique d'information dans des textes et aider l'homme à traiter une quantité toujours croissante de données textuelles. On a également remarqué que la définition du résumé peut varier en fonction des objectifs qu'on lui attribue.

Ensuite, nous avons présenté les travaux majeurs du domaine. L'approche abstractive résume un texte sur un mode similaire à l'homme, mais s'avère difficilement généralisable. Une raison pour laquelle la recherche actuelle délaisse cette approche au profit de l'approche extractive. Cette dernière a engendré de nombreux travaux, utilisant des formalismes très variés. Avec le temps, les techniques d'extraction se sont complexifiées, allant de la simple reconnaissance d'expressions à des modèles probabilistes plus sophistiqués (modèles vectoriels, graphes, réseaux de neurones). Plusieurs tentatives intéressantes ont été faites pour inclure un maximum de composantes sémantiques dans le choix des phrases à extraire.

Après cela, nous avons abordé le sujet de l'évaluation des résumés, un enjeu plus complexe pour le RAT que pour d'autres domaines du TAL. En raison de la subjectivité de la tâche, il est difficile d'envisager une évaluation intrinsèque automatisée. Les mesures les plus utilisées actuellement sont BASIC ELEMENTS et PYRAMID pour les mesures semi-automatiques et les divergences KL et JS, pour les mesures entièrement automatiques. Les évaluations extrinsèques se font le plus souvent manuellement.

Nous avons finalement évoqué les limites de l'extraction de phrases pour le RAT. Ces limites amènent le chercheur à dépasser la simple extraction de phrases pour envisager l'usage d'autres traitements de la phrase, comme l'homme ne se contente pas d'extraire des phrases lorsqu'il résume. Parmi ces traitements, la compression de phrases se pose en action élémentaire vers un résumé plus intelligent. Les résumeurs extractifs bénéficieraient donc pleinement de l'usage d'un module de compression dans leurs algorithmes. Voilà pourquoi, dans les années 2000, la compression connaît un (re)gain d'intérêt certain chez les chercheurs en RAT. Dans le chapitre suivant, nous introduisons ce nouveau domaine utile au RAT, de la même façon que nous l'avons fait pour le RAT.

Chapitre 3

Compression automatique de phrases

Introduction

Dans ce chapitre, nous traitons de la compression automatique de phrases (CAP). Cette tâche du TAL cherche à réduire des phrases en évitant de les rendre agrammaticales ou de changer leur sens initial. Le domaine de la CAP s'est développé autour de l'an 2000, dans des objectifs, au départ, distincts du résumé automatique. En raison des intérêts et buts communs entre les deux domaines, leur développement s'est néanmoins rapidement lié. Un compresseur automatique de phrases semble en effet être un bon atout pour la construction de résumés extractifs plus pertinents. Comme nous consacrons le chapitre suivant à l'utilisation de la compression pour le RAT, nous renvoyons le lecteur au chapitre 4 pour une explication détaillée des raisons qui motivent l'emploi de compresseurs pour des résumeurs extractifs.

Ce chapitre se découpe comme suit. Nous commençons par définir la tâche compression automatique de phrases et ses applications, en particulier dans son rapport avec le résumé (3.1). Nous présentons, dans les sections 3.2 à 3.4, les approches méthodologiques existantes pour la CAP, en exposant leurs qualités et leurs défauts. Nous dédions la section 3.5 à la question de l'évaluation, dont la complexité se rapproche de celle du résumé.

3.1 Définitions et enjeux

3.1.1 Définition de la tâche

Comparé au RAT, cela fait très peu de temps que la recherche s'intéresse à la compression automatique de phrases (CAP). Les premiers travaux en CAP ont en effet été publiés autour de l'an 2000, et ce avec des objectifs divers. Pour cette raison, on ne trouve pas de consensus sur la définition de la CAP dans la littérature actuellement. Le mot *compression* suggère de façon évidente l'idée de réduire la longueur d'une phrase, mais les définitions proposées par les chercheurs divergent sur la liste d'opérations à effectuer pour y arriver et sur les propriétés du résultat final.

Knight & Marcu (2002) définissent la CAP comme l'action de supprimer des mots, dont le résultat est grammatical et contient l'essentiel du contenu de la phrase d'origine. D'autres chercheurs imaginent plutôt la CAP comme un ensemble d'opérations plus complexes, incluant la paraphrase, le réarrangement et la substitution (Clarke & Lapata 2006, 2009), dans une vision de la tâche se rapprochant de la simplification automatique de texte. En fait, les délimitations de la tâche de CAP dépendent plus du résultat souhaité que du processus de compression en lui-même. Classiquement, en RAT, on considérera par exemple que la compression de phrases ne fait qu'éliminer des segments, ce qui ne sera pas le cas en simplification automatique. Ainsi, il nous semble plus pertinent, pour définir précisément la CAP, de centrer la description autour du produit final. Molina Villegas (2013 : 24-25) a récemment proposé une définition de la CAP allant dans ce sens : « Étant donné [une] phrase d'origine, un algorithme [de compression] doit [en] trouver une nouvelle version devant à la fois être plus courte, contenir les informations les plus importantes [et] être grammaticalement correcte. » On produira et jugera donc une compression selon trois critères : sa longueur, son informativité et sa grammaticalité.

D'un point de vue théorique, nous privilégions une définition de la CAP du même acabit que celle donnée par Molina Villegas. D'un point de vue pratique, comme nous comptons appliquer la compression de phrases en vue de résumer, il s'agira, dans ce travail, de compression dans son sens strict, c'est-à-dire la simple suppression de mots. Nous reléguons les opérations de paraphrase et réarrangement de l'ordre des mots au domaine de la simplification de texte (voir Siddharthan 2014 et Saggion 2017 pour plus d'informations sur ce domaine du TAL).

On remarquera que les critères décrivant la CAP sont les mêmes critères que ceux mis en avant dans la définition du résumé. En réalité, la frontière entre le résumé et la compression est très fine, et certains définissent même la CAP comme le résumé appliqué au niveau de la phrase (Turner & Charniak 2005, Clarke & Lapata 2009). Torres-Moreno mentionne quant à lui la CAP comme une approche particulière du RAT. Nous affichons dans ce travail une posture différente de ce dernier, considérant plutôt la compression comme une tâche singulière.

3.1.2 Enjeux et applications

Supprimer les parties non pertinentes de phrases écrites en langage naturel s'avère utile dans plusieurs domaines du TAL. Historiquement, la CAP a d'abord été envisagée par Grefenstette (1998) pour des synthétiseurs vocaux. Le but était de réduire les phrases pour réduire la complexité temporelle du système. On note aussi des utilisations dans la géné-

ration automatique de sous-titres (notamment Daelemans *et al.* 2004), un domaine dans lequel il apparaît nécessaire de débarrasser le texte source d'éléments inutiles (hésitations, répétitions faites à l'oral) et de réduire la longueur de phrases trop longues (pour la transcription de discours rapides). Dans le domaine de la recherche de documents (*information retrieval*), Corston-Oliver & Dolan (1999) éliminent les subordonnées de documents à indexer et améliorent de cette façon les performances de leur système. La CAP peut également intervenir dans le cadre de la simplification automatique de texte, dont l'objectif est de réduire la difficulté de lecture d'un texte d'un point de vue lexical et/ou syntaxique (voir Siddharthan 2014).

L'application majeure de la CAP dans le TAL reste toutefois le résumé automatique. Les deux tâches partagent en effet des objectifs communs, et leur développement s'effectue aujourd'hui en parallèle. Au début des années 2000, la communauté scientifique s'est penchée sur le problème de la CAP dans le but précis d'améliorer les systèmes de résumés extractifs, à un moment où ceux-ci semblent avoir atteint leurs limites. De façon générale, la CAP est actuellement considérée, avec la combinaison de phrases (*multi-sentence fusion*), comme une étape vers une méthode abstractive qui se passe de modèle sémantique préexistant. Nous développons l'utilité de la compression pour le résumé plus loin (voir chapitre 4).

3.2 Méthodologies

Comme pour beaucoup de domaines du TAL, la compression automatique de phrases peut s'appréhender selon deux approches distinctes : l'approche symbolique et l'approche statistique. Ces approches se différencient plus par les ressources et outils qu'elles nécessitent que par leur façon de compresser. Il s'agit en effet, dans la plupart des cas, de définir un ensemble de règles de compression. Dans l'approche symbolique, le jeu de règles s'élabore en grande partie manuellement, avec un ensemble plus ou moins compliqué de contraintes sur la compression pour garantir la grammaticalité. Dans l'approche statistique, les règles sont obtenues par entraînement supervisé : on fournit un corpus aligné de phrases originales et de phrases compressées à un modèle probabiliste, puis on l'entraîne de sorte qu'il puisse compresser des phrases nouvelles.

Nous présentons, dans la suite de ce chapitre, les différents travaux publiés en CAP. Comme nous présenterons deux systèmes de compression suivant chacun une approche de la CAP différente (voir partie II), nous nous attardons sur chacune des approches, en commençant par l'approche symbolique.

3.3 Approche symbolique

L'élaboration des règles de compression dans l'approche symbolique résulte du travail préalable d'un linguiste. Celui-ci écrit des règles portant en général sur la nature des mots et la structure syntaxique de la phrase. Ainsi, Grefenstette (1998) construit manuellement 8 niveaux de réduction lui permettant d'envisager des compressions plus ou moins courtes. Chaque niveau indique les types d'arguments à conserver, le premier niveau supprimant tout ce qui n'est pas un nom propre (la compression produite n'est pas grammaticale dans ce cas), et le dernier niveau ne conservant que les sujets, verbes principaux, complément d'objet, les prépositions et les phrases enchâssées. Conroy *et al.* (2006 : 2) suppriment quant à eux de façon systématique les conjonctions, la plupart des adverbes, les mots

entre parenthèses, les subordonnées introduites au participe présent... Il arrive souvent que certains mots ou expressions soient identifiés tels quels : dans le système CLASSY de Conroy *et al.*, les locutions d'enchaînements comme *at this point*, *however* et *as a matter of fact* ainsi que les discours rapportés (*police said*) sont éliminés sans vérification. Les travaux de Toutanova *et al.* (2007) et Wang *et al.* (2016) suivent une approche similaire à CLASSY.

On peut en outre travailler à un niveau plus élevé d'abstraction, notamment au niveau des fonctions grammaticales, comme le proposent Yousfi-Monod & Prince (2006). Dans leur système, ceux-ci suppriment, à certaines conditions, les compléments circonstanciels, les épithètes et les appositions identifiés par l'analyseur SYGFRAN.

Peu de travaux présentant une approche exclusivement symbolique à la compression ont été publiés jusqu'à présent. Cela procède vraisemblablement, d'une part, de la jeunesse du domaine concerné, les premiers travaux en CAP datant de 1998, et d'autre part, de la tendance générale en TAL, depuis le succès des méthodes par l'apprentissage automatique et des HMM (modèles de Markov cachés) pour certaines tâches fondamentales, à privilégier l'approche statistique. Le principal désavantage de l'approche manuelle est son coût au niveau du temps d'élaboration des règles. Elle requiert également souvent l'accès à des informations syntaxiques et sémantiques difficiles à produire automatiquement et rapidement, ce qui ralentit le processus final. Toutefois, elle garantit la plupart du temps de meilleurs résultats sur le plan de la grammaticalité et de la cohérence textuelle.

3.4 Approche statistique

3.4.1 L'étude pionnière de Knight et Marcu

Si l'on veut créer un jeu de règles de façon automatique, il faut alors exploiter le contenu de corpus de phrases originales alignées avec leur(s) compression(s). Les pionniers de ce type d'approche sont Knight & Marcu, qui présentent en 2002 une méthode probabiliste de la compression à partir d'un corpus de textes originaux et de résumés du corpus Ziff-Davis. Il est important de comprendre la formalisation du problème de la compression présentée dans ce travail, car elle a influencé toute la recherche qui l'a suivi. Pour résoudre statistiquement le problème de la CAP, Knight & Marcu se servent du modèle du canal bruité, modèle notamment utilisé en traduction automatique. Conceptuellement, cela revient à prendre le problème en sens inverse : on considère que les phrases sources étaient originellement compressées, et que des mots y ont été ajoutés (Knight & Marcu 2002 : 3). Le modèle du canal bruité sert à « retrouver » la phrase compressée, en définissant mathématiquement la probabilité d'une phrase compressée c étant donné une phrase source s :

$$P(c|s) = P(c)P(s|c) \quad (3.1)$$

$P(c)$ est le modèle de la source : il définit la vraisemblance de c et permet de donner un mauvais score aux phrases agrammaticales. $P(s|c)$ est le modèle du canal, soit la probabilité que c puisse être étendue en s . Dès lors, étant donné une certaine phrase source, on considère que la meilleure compression possible est celle avec laquelle on obtient les probabilités les plus hautes. Les auteurs remarquent toutefois que cette méthode a tendance à favoriser les compressions grammaticales les plus courtes possible, entraînant dans de nombreux cas une perte d'information non acceptable. Ils remédient à ce problème en divisant la probabilité $P(c|s)$ en échelle logarithmique par la longueur de s . Pour trouver

c , on a donc la formule suivante :

$$c = \underset{c}{\operatorname{argmax}} \frac{\log(P(c)P(s|c))}{|s|} \quad (3.2)$$

Les probabilités sont obtenues par entraînement sur corpus. Comme il est évident qu’aucun corpus aligné ne peut contenir toutes les compressions possibles pour toutes les phrases possibles, le comptage d’occurrences s’opère à partir des arbres syntaxiques. Les auteurs utilisent le formalisme des grammaires probabilistes indépendantes du contexte ou PCFG (*Probabilistic context-free grammar*) (voir Jurafsky & Martin 2009). Ils créent ainsi deux grammaires, une PCFG « classique » pour calculer $P(c)$, et une seconde qui contient les règles de transformation et qui permet de calculer $P(s|c)$.

La méthode de Knight & Marcu (2002) possède un double avantage. Elle est à la fois plus rapide à produire et à évaluer. Tout ce qu’elle requiert est un corpus aligné, un matériau certes assez rare dans le domaine récent de la CAP. Mais une fois ce corpus obtenu, il suffit d’entraîner le modèle et de l’évaluer automatiquement avec un corpus test. Pour la première fois, l’automatisation de l’entièreté du processus de compression devient possible.

3.4.2 Travaux postérieurs

L’article de Knight & Marcu (2002) a fait grand bruit lors de sa sortie, et a grandement participé au développement de l’intérêt pour la CAP. Il suscite également un regain d’intérêt pour le RAT, puisque la CAP y est précisément désignée comme une première étape vers l’abstraction¹. La méthode statistique basée sur l’apprentissage automatique supervisé semble très prometteuse, tant et si bien que la majorité des systèmes présentés après 2002 suivent cette voie (Turner & Charniak 2005, Galley & McKeown 2007, Unno *et al.* 2006, Martins & Smith 2009). Ils se différencient néanmoins dans l’utilisation des PCFG, certains incluant des paramètres supplémentaires et d’autres employant des formalismes syntaxiques différents.

Pour commencer, Galley & McKeown (2007) mettent en avant certaines faiblesses des PCFG et en particulier de celles respectant le format de la *Penn Treebank* (Marcus *et al.* 1994). D’une part, il arrive parfois qu’une même structure syntaxique soit taguée différemment dans le corpus, entraînant des disparités problématiques pour une approche entièrement automatique. D’autre part, l’étiquetage manquerait de finesse pour la tâche de la compression, notamment à cause de l’absence de différenciation entre compléments obligatoires et non obligatoires. Les chercheurs rajoutent donc des annotations permettant de distinguer les constituants principaux des modificateurs, et indiquent dans l’étiquetage le fait qu’un complément soit fortement lié à un verbe ou non (Galley & McKeown 2007 : 183). Ils modifient par ailleurs la façon de Knight & Marcu d’aligner les arbres sources avec leur compression. Là où ces derniers ne comptaient que les phrases pour lesquelles on retrouvait les mêmes mots dans la compression, Galley & McKeown s’autorisent à compter les phrases ayant subies des insertions ou substitutions légères, engrangeant plus d’exemples pour l’entraînement. Les auteurs ont entraîné et testé leur système sur le même corpus que Knight & Marcu, et ils parviennent à des résultats meilleurs en matière d’informativité (Galley & McKeown 2007 : 186-187).

1. Le titre de l’article de Knight & Marcu (2002) y fait directement référence : *Summarization beyond sentence extraction : A probabilistic approach to sentence compression*.

Dans leur travail en résumé et compression conjointes, Martins & Smith (2009) préfèrent quant à eux les grammaires de dépendances aux PCFG pour la CAP. Ils ont également recours à la programmation linéaire en nombres entiers (PLNE ou ILP en anglais), un formalisme qui permet de calculer rapidement la compression la plus probable d'une phrase donnée.

3.4.3 Approches récentes

Une série d'approches récentes emploient des formalismes statistiques plus complexes que le modèle du canal bruité. Nous en présentons trois : la structure rhétorique, les champs conditionnels aléatoires et les réseaux de neurones.

Une première approche intéressante est celle de Molina Villegas, développée dans sa thèse de 2013. Comme vu au début de ce chapitre, la CAP y est envisagée selon trois paramètres : sa grammaticalité, son informativité et sa longueur. Molina Villegas calcule la grammaticalité des phrases grâce à un modèle de langage classique, tandis qu'il modélise l'informativité via l'énergie textuelle (cf. section 2.4.7). La nouveauté de son approche se situe surtout dans l'emploi de la structure rhétorique (introduite à la section 2.4.8) : plutôt que d'élaguer des arbres syntaxiques, la méthode travaille à partir d'arbres *discursifs*. Pour disposer de données en espagnol et annotées spécifiquement pour la compression, Molina Villegas a créé un corpus participatif annoté par des non-experts. Grâce à ces données, il entraîne un modèle de régression linéaire et produit ainsi un système capable de choisir quels segments discursifs conserver dans la phrase. Durret *et al.* (2016) compressement aussi des arbres discursif, mais en ayant recours à la PLNE pour le calcul de l'informativité.

Une seconde approche probabiliste est l'emploi de champs aléatoires conditionnels (abréviés CRF pour *Conditional Random Fields*) (Nomoto 2007, Li *et al.* 2013). Il s'agit d'une approche similaire à celle des HMM, mais dans une version dite « discriminative », c'est-à-dire qui modélise directement $P(c|s)$. L'avantage est qu'il ne faut plus calculer $P(c)$ (probabilité *a priori* d'une compression)². Ici, le concepteur du modèle choisit librement les variables ou « facteurs » (*features*) à observer pour chaque mot de la phrase. Il peut s'agir de mesures basiques comme la fréquence brute, la capitalisation ou le stem, mais aussi de mesures syntaxiques comme des *n*-grams de mots ou de catégories de discours (Nomoto 2007 : 1579). En théorie, tout type de phénomène linguistique peut être utilisé avec ce formalisme. Nomoto a montré que ce formalisme pouvait produire de meilleurs résultats que ceux basés uniquement sur les PCFG, bien que la grammaticalité s'avère plus compliquée à conserver (Nomoto 2007 : 1580-1585).

Enfin, la dernière approche que nous voulons mentionner est celle des réseaux de neurones récurrents (RNN), en particulier leur version LSTM (*Long Short-Term Memory*) (Filippova *et al.* 2015, Sakti *et al.* 2015, Klerke *et al.* 2016). Il s'agit d'un modèle de réseaux de neurones adapté à la prédiction de séquences et capable de prendre en compte des dépendances très éloignées (Sutskever *et al.* 2014). Un autre avantage de ce modèle est qu'il peut traiter un très grand nombre de variables. En revanche, le modèle demande généralement plus de données pour l'entraînement que les algorithmes décrits précédemment : par exemple, Filippova *et al.* (2015) disposent d'un peu moins de 2 millions d'exemples. Les variables prises en compte sont du même type que celles des CRF, soit des informations sur les tokens ou sur l'arbre syntaxique de la phrase (Filippova *et al.* 2015 : 362).

2. Nous expliquons en détail ce formalisme au chapitre 5.

3.5 Évaluation

Comme pour le RAT, on peut évaluer les performances d'un compresseur de façon intrinsèque ou extrinsèque.

3.5.1 Évaluation intrinsèque

Lorsqu'on juge directement de la qualité d'une compression, on prend généralement en compte ses trois critères définitoires : le taux de compression, la grammaticalité et l'informativité. Le taux de compression (ou τ) donne, comme dans le cadre du RAT, le pourcentage d'unités conservées dans une compression par rapport à la phrase originale. Si c est la compression de p , alors τ se calcule selon l'équation 3.3. L'unité choisie pour mesurer la longueur d'une phrase est souvent le token, mais on peut aussi prendre le caractère ou le syntagme.

$$\tau = \frac{|c|}{|p|} \quad (3.3)$$

Le taux de compression ne fait pas toujours l'objet d'une évaluation particulière, car cette caractéristique peut être jugée de façon rapide et objective, et aussi parce que souvent, les chercheurs s'assurent d'un taux de compression minimum lors du développement, soit bien avant l'évaluation. La grammaticalité peut être jugée de façon binaire ou scalaire, presque toujours par l'homme. L'informativité renvoie au rendement entre la longueur de la compression et la quantité et la qualité de l'information qu'elle a conservées de l'original. On souhaite qu'il soit le plus élevé possible. Par ailleurs, les phrases compressées peuvent être notées en considérant leur contexte (en comparaison avec le texte original ou au sein de l'entière du texte compressé), ou de façon indépendante.

Il est tout d'abord possible d'évaluer des phrases compressées de façon manuelle. On citera pour exemple Knight & Marcu (2002), Turner & Charniak (2005), Clarke & Lapata (2006), Galley & McKeown (2007), Molina Villegas (2013), Filippova *et al.* (2015). Certains jugent eux-mêmes de la qualité de leurs compressions (par exemple Yousfi-Monod & Prince 2006), mais la plupart font appel à un panel de juges indépendants. La grammaticalité et la conservation de l'information du résultat sont jugées séparément, la plupart du temps via une échelle allant de 1 à 5 (Napoles *et al.* 2011 : 93). L'analyse du taux de compression, dans ce genre d'approche, n'est pas systématiquement réalisée, un tort selon Napoles *et al.* (2011 : 94). Ces derniers ont en effet montré que les jugements humains étaient grandement influencés par le taux de compression, tant pour la grammaticalité que pour l'informativité. L'évaluation intrinsèque manuelle devient donc encore plus ardue à mettre en œuvre : les conclusions de l'étude de Napoles *et al.* (2011) suggèrent en effet que deux systèmes doivent être comparés sur un même corpus et compressant à des taux similaires. Ces deux conditions ne sont que très rarement remplies parmi les travaux produits jusqu'à aujourd'hui. Pour les systèmes qui ne prévoient pas un contrôle direct du taux de compression, les auteurs indiquent qu'il faut tout de même regrouper les phrases à évaluer selon leur taux de compression respectif (Napoles *et al.* 2011 : 95).

On peut également évaluer de façon automatique. Une des mesures disponibles est la SSA (*Simple String Accuracy*, voir Bangalore *et al.* 2000), technique d'évaluation reprise du domaine de la génération automatique de langage. Elle calcule la distance entre deux chaînes de caractères (l'une à évaluer, l'autre en tant que référence) en nombres d'insertions, de suppressions et de substitutions effectuées entre les deux. Elle peut s'utiliser pour juger des

compressions plus complexes (avec modifications syntaxiques et lexicales), ce qui élargit le champ des corpus utilisables pour les tests. Utilisée pendant quelque temps, la SSA est aujourd'hui délaissée, après que Clarke & Lapata (2009 : 7) ont montré qu'elle n'était pas corrélée avec les jugements humains.

Si l'on dispose d'un corpus de test annoté, on peut aussi calculer les scores de rappel, précision et F_1 (notamment Sakti *et al.* 2015, Klerke *et al.* 2016). Des études ont en effet montré que ce dernier score est en corrélation avec les jugements humains (Clarke & Lapata 2009). Une mesure couramment utilisée est la mesure F_1 basée sur la comparaison des relations grammaticales entre la sortie du système à tester et un corpus de référence (*gold standard*) (Napoles *et al.* 2011 : 91). Certains travaux se basent aussi sur les unigrammes (Martins & Smith 2009) ou bigrammes (Unno *et al.* 2006). Ces mesures ne donnent cependant aucune idée sur la qualité linguistique des phrases produites et ne devraient, en ce sens, pas suffire pour l'évaluation d'un compresseur.

En fin de compte, le développement des techniques automatiques intrinsèques semble encore limité de nos jours. Nous imputons ce fait au caractère somme toute récent du domaine de la CAP, ainsi qu'à sa complexité. En fait, la plupart des recherches en CAP incluent une évaluation manuelle, parfois en complément à une évaluation automatique (par exemple Martins & Smith 2009). Comme le signalent Napoles *et al.* (2011 : 93), la méthode manuelle reste actuellement préférable pour l'évaluation intrinsèque car la plus fiable.

3.5.2 Évaluation extrinsèque

L'évaluation extrinsèque est une manière courante d'évaluer un système de compression dans la littérature scientifique. Dès que des chercheurs envisagent la compression dans l'objectif de résumer, ceux-ci utilisent volontiers les techniques d'évaluation du RAT. À nouveau, cela dénote le lien particulier entre CAP et RAT. Comme nous nous sommes déjà attardé sur les techniques d'évaluation du RAT (cf. section 2.5.1), nous nous contenterons ici de citer quelques travaux et les mesures qu'ils utilisent.

Les mesures de comparaison les plus employées sont ROUGE-2 et ROUGE-SU4, comme on le note chez Conroy *et al.* (2006), Wang *et al.* (2016) et Durrett *et al.* (2016). Les mesures BLEU sont également utilisées.

Certains systèmes de compressions participent au concours DUC ou comparent leurs performances avec celles du meilleur système de la dernière conférence DUC en date. On citera pour exemple Toutanova *et al.* (2007), qui exécutent leur compresseur pour remplir une tâche de DUC 2007, et Martins & Smith (2009), qui emploient quant à eux les données de DUC 2002 et compare les scores ROUGE obtenus avec des compresseurs de base.

Conclusion

Nous avons expliqué dans ce chapitre les applications et méthodes principales du domaine de la compression de phrases. Nous avons défini l'action de compression ainsi que les critères d'une bonne compression. Ceux-ci se sont avérés être les mêmes que pour le résumé, ce qui a mis en évidence les similitudes importantes entre ces deux domaines.

Nous avons ensuite présenté l'état de l'art en CAP, en abordant d'abord l'approche symbolique, et puis l'approche statistique. Si l'approche manuelle donne, en général, de bons résultats sur le plan grammatical, l'approche statistique donne la possibilité de créer des règles plus précises et plus contextuelles.

Finalement, nous avons détaillé les différents moyens disponibles pour l'évaluation. Nous avons alors remarqué que l'évaluation intrinsèque ne va pas de soi. Tout d'abord, parce que les appréciations des juges humains dépendent fortement du taux de compression des phrases et du corpus utilisé pour l'entraînement (pour l'approche statistique), deux éléments qui varient pour presque chaque compresseur. Ensuite, à cause de l'absence de mesures d'évaluation automatique complète. L'évaluation extrinsèque reprend quant à elle les techniques utilisées en RAT.

Il convient à présent de nous attarder spécifiquement sur l'utilisation de la CAP pour le résumé extractif. C'est la thématique du chapitre suivant.

Chapitre 4

Compression de phrases pour le résumé

Introduction

Comme nous le signalions précédemment, le développement de la CAP comme domaine du TAL vient en grande partie de ses applications en résumé automatique. Dans les années 2000, après plusieurs dizaines d'années de recherche, on commence à mettre en évidence certaines limites intrinsèques à l'approche extractive du résumé (voir section 2.6). Ces limites pousseront les chercheurs à élargir leur champ de recherche et explorer des techniques d'autres domaines du TAL. La compression de phrases a alors constitué une piste sérieuse pour l'amélioration des résumés extractifs. Assez rapidement, les chercheurs ont intégré des modules de compression à des systèmes de résumés, permettant de juger aujourd'hui la pertinence de cette technique.

Dans ce chapitre, nous commençons par évoquer les raisons théoriques et pratiques qui permettent de penser que la compression est un bon moyen d'améliorer le résultat d'un résumeur (section 4.1). Nous présentons ensuite les différentes façons dont les chercheurs ont inclus des modules de compression dans des résumés et jugeons de leur efficacité (section 4.2).

4.1 Intérêt de la compression pour le résumé

La CAP est intéressante pour la tâche de résumé à plusieurs égards. Tout d'abord, avant de préciser son intérêt d'un point de vue théorique, il convient de mentionner certaines études qui motivent de façon empirique le bien-fondé de la compression pour le résumé. Nous avons déjà évoqué les travaux de Copeck & Szpakowicz (2004) et de Jing & McKeown (1999) pour justifier la pertinence de l'extraction de phrases pour le résumé. Ces arguments s'avèrent en réalité encore plus valables concernant la compression. Dans ces études sur la façon humaine de résumer, on a observé une petite quantité de phrases reprises *verbatim*, mais leur nombre augmente très significativement lorsqu'on autorise des opérations de réductions. Au niveau des syntagmes, Copeck & Szpakowicz (2004) ont également montré que la reprise partielle de noms d'entités est un phénomène fréquent, tandis que, de façon plus générale, Jing & McKeown (1999) ont identifié l'opération de réduction de phrases comme la plus effectuée, avec la combinaison de phrases.

Si le simple fait que la compression de phrases corresponde à l'usage humain pourrait sans doute suffire à justifier son utilisation dans le résumé, il existe de nombreuses autres raisons. La première étant qu'elle autorise une dissection plus fine de l'information, en travaillant à un niveau non envisagé par les extracteurs de phrases. Prenons cet exemple tiré de (Knight & Marcu 2002) pour en montrer l'avantage :

Original : *All of our design goals were achieved and the delivered performance matches the speed of the underlying device.*

Compressé : *All design goals were achieved.*

On voit ici que les mots non pertinents de la phrase, pourtant en majorité numérique, ont été supprimés, pour donner une nouvelle phrase à la syntaxe plus courte et directe. Avec cette compression, on se rapproche du résumé humain tant au niveau de la sélection de l'information que du style, sans effectuer d'opérations de transformation lexicale ou syntaxique. Le résumé gagne ainsi en pertinence et en fluidité.

La CAP devient une solution de choix pour régler le problème de sélection préférentielle des phrases longues posé par l'extraction de phrases (cf. section 2.6). Non seulement parce que la compression réduit le nombre de phrases longues de façon absolue, mais aussi parce que, même dans le cas où la longueur d'une phrase reste inchangée, elle offre une garantie quant à la pertinence de chaque élément de la phrase, contrairement à l'extraction de phrases, qui donne un avis sur l'ensemble de la phrase (Silveira & Branco 2012 : 7). D'autre part, peu importe la taille d'une phrase, la compression reste très intéressante dans le cas où seule une partie minoritaire contient de l'information à conserver.

Ces arguments montrent en définitive que la compression pour le résumé améliore le rendement entre le nombre d'informations à présenter et la taille du produit du final. En outre, lorsqu'on réduit des phrases avant de les extraire, on pourra inclure dans le résumé plus de phrases (et donc d'information) pour un même espace de texte (Vanderwende *et al.* 2006 : 2).

Un dernier avantage de la CAP a été mis en avant par Siddharthan *et al.* (2004) dans le cadre du résumé multi-document. Dans ce genre de résumé, une étape préliminaire à l'extraction est celle du regroupement de phrases (*sentence clustrering*) : les phrases de tous les documents sources sont réparties en groupes en fonction de leur similarité de contenu. Dans leur étude, Siddharthan *et al.* ont remarqué que le regroupement des phrases s'améliorait significativement s'ils les compressaient au préalable (Siddharthan *et al.* 2004 : 2).

4.2 Méthodologies

On peut classer les différents travaux en compression pour le RAT selon le moment où l'algorithme de compression des phrases est appliqué. Cela peut se faire avant ou après l'extraction des phrases, ou bien de façon conjointe.

4.2.1 La compression comme prétraitement

Un des premiers travaux de compression pour le résumé est celui de Siddharthan *et al.* (2004). Ceux-ci proposent de compresser avant de résumer et analysent l'influence de cette action sur les scores ROUGE. Ils démontrent que la suppression d'éléments syntaxiques superflus (*parentheticals*) mène à une meilleure sélection du contenu pour le résumé multi-document, notamment grâce à une meilleure étape de groupement des phrases. Ils font, de plus, une remarque intéressante concernant la compression syntaxique (Siddharthan *et al.* 2004 : 4), qui, selon eux, n'est pas aussi systématique que l'on pourrait le croire :

Nous pensons que l'inclusion d'informations superflues à propos des entités devrait être traitée comme une tâche de génération de référence plutôt qu'une tâche de sélection de contenu. Notre analyse des résumés humains suggère que les personnes sélectionnent des éléments superflus pour améliorer la cohérence et aider le lecteur dans l'identification de référents et des liens entre eux dans le discours.

Cette observation suggère que, dans certains cas, il faille tout de même inclure des éléments d'information *a priori* secondaires pour des raisons de cohérence discursive. L'exemple que donnent les auteurs de l'étude est celui de la première apposition associée à un nom propre, souvent nécessaire à la bonne compréhension de textes à caractère informatif comme les articles de presse.

La seconde étude à mentionner est celle de Yousfi-Monod & Prince (2006), faisant également usage de la compression syntaxique. Dans celle-ci, ils mesurent l'effet de leur compresseur pour le français sur des textes narratifs. Ils notent également une perte de cohérence dans le texte compressé. Ils signalent en outre que la tâche de compression dépend largement du genre du texte.

4.2.2 La compression comme post-traitement

En 2005, Gagnon & Da Sylva (2005) envisagent d'appliquer la compression sur des phrases déjà triées par ordre d'importance. Leur compression syntaxique supprime les incises, subordonnées et compléments prépositionnels de textes francophones dans le cadre du résumé uni-document. Ils obtiennent des résultats équivalents sur le plan informatif pour un meilleur taux de compression. Ils repèrent toutefois un certain nombre de cas où la compression est plutôt guidée par la sémantique et la cohérence du texte que la structure syntaxique d'une phrase.

Dans une approche similaire de la compression, Silveira & Branco (2012) ont comparé le résultat des scores ROUGE entre des résumés compressés et non compressés de textes d'actualité en portugais. Leurs conclusions vont en faveur de la compression pour le résumé, bien que la différence de score soit légère.

Wang *et al.* (2016) publient une étude dans laquelle ils évaluent les performances de résumeurs extractifs avec et sans compression post-traitement, pour trois types distincts de

compression : la compression par règles, par séquences et une méthode automatique de compression syntaxique. La compression par séquences traite le problème de la compression comme un problème de classification binaire, en utilisant le format BIO. Le dernier compresseur envisagé se base sur un modèle obtenu par apprentissage automatique sur corpus d'arbres syntaxiques élagués. Les résultats obtenus soutiennent la pertinence théorique de la CAP pour le résumé : en effet, quelle que soit la technique de compression utilisée, les résumés compressés produits par les chercheurs surpassent tous significativement les meilleurs systèmes de DUC 2006 et DUC 2007, selon l'évaluation automatique (ROUGE) et manuelle (Wang *et al.* 2016 : 8). Parmi les trois approches, la compression automatique d'élagage syntaxique obtient les meilleurs scores.

4.2.3 Compression et extraction conjointes

L'application conjointe de l'extraction et de la compression de phrases consiste à extraire des segments de phrases plutôt que des phrases entières, tout en laissant la possibilité d'extraire la phrase originale dans son entièreté. Plus tardive que les approches précédentes, elle a notamment été développée dans les travaux de Vanderwende *et al.* (2006), Molina Villegas (2013), Li *et al.* (2013) et Durrett *et al.* (2016).

Dans leur système, Vanderwende *et al.* (2006) proposent une des premières approches conjointes en offrant au module de sélection une phrase originale du texte source et sa version compressée, qui sont ainsi mises en concurrence. Il s'agit d'une compression purement syntaxique basée sur des règles manuelles. Ils remarquent tout d'abord que les versions compressées ont été choisies dans 43 % des cas et que le nombre moyen de phrases incluses dans le résumé a augmenté de 1,2 (Vanderwende *et al.* 2006 : 4), ce qui manifeste de la pertinence de la compression pour l'extraction. Si les compressions ont posé des problèmes de grammaticalité et de coréférence, ils observent une nette amélioration pour ce qui est de la non-redondance et de la pertinence du contenu (Vanderwende *et al.* 2006 : 5).

Les trois autres travaux cités précédemment suivent un même processus. Il s'agit tout d'abord de produire plusieurs versions compressées d'une phrase originale et de les soumettre, en plus de la phrase originale, à un module de sélection. La sélection s'opère alors grâce à des modèles obtenus par entraînement. Dans le cas de Molina Villegas (2013), ce modèle est celui de la régression linéaire, tandis que Li *et al.* (2013) et Durrett *et al.* (2016) utilisent le formalisme de la programmation linéaire en nombres entiers. Molina Villegas prouve l'utilité de sa méthode en effectuant un test de Turing sur des volontaires, qui doivent attester la provenance humaine ou automatique d'un résumé. Les résultats de ces attestations sont soumis à un test statistique de Fisher, et l'hypothèse nulle stipulant que l'attribution de la provenance d'un résumé ne dépend pas de cette provenance a été retenue pour 53 des 54 juges (Molina Villegas 2013 : 96-105). Li *et al.* (2013) testent leur système sur les corpus des conférences TAC 2008 et 2011 et présentent des scores R-2 et R-SU4 plus élevés que les systèmes purement extractifs. Enfin, Durrett *et al.* (2016) observent également une amélioration des scores ROUGE sur un corpus d'articles courts du *New York Times*, bien qu'ils déplorent une perte de qualité linguistique.

Conclusion

Nous venons de citer les travaux importants effectués en CAP pour le RAT. On a vu que l'on pouvait compresser avant, pendant ou après l'extraction de phrases. Aucune méthode ne paraît supérieure à une autre, bien que la recherche récente semble mieux considérer la compression conjointe. Cette dernière requiert une implémentation originale comprenant un module de sélection statistique de segments.

Nous avons au préalable listé les arguments qui justifient l'emploi de la compression pour produire de meilleurs résumés extractifs. En offrant une dissection plus fine de l'information à retenir, elle permet d'extraire plus de phrases et de faire gagner le résumé en fluidité. Des études montrent qu'elle correspond à la technique humaine du résumé. Ces arguments suggèrent que la compression est une opération essentielle vers des résumés semblables à ceux produits par les humains.

Conclusion de la partie I

Nous avons offert, dans cette première partie, une large vue d'ensemble des domaines du résumé automatique de texte et de la compression de phrases. Il s'agit de deux domaines du TAL distincts, mais dont les objectifs se rapprochent tellement, que des systèmes « hybrides » ont déjà vus le jour. C'est en évoquant les articles présentant de tels systèmes que nous avons accentué l'intérêt de la CAP pour le résumé automatique.

L'état de l'art nous a par ailleurs montré qu'il existe une grande variété dans les techniques de traitement du texte, en particulier pour le résumé. On peut dégager deux grandes tendances parmi la masse de travaux en RAT mentionnés : ceux qui ont recours à des formalismes sémantiques (en extraction de phrases comme en abstraction) et ceux ne se basant que sur la syntaxe et/ou une analyse de surface du texte. Aucun des deux groupes ne semble avoir trouvé de solution parfaite à l'heure actuelle. En CAP, la recherche a accordé moins d'attention à l'approche sémantique, tant le nombre de systèmes exploitant de l'information sémantique pour les phrases est faible. Cela est sans doute dû à un manque de ressources dans ce domaine encore récent du TAL.

Nous avons également attiré l'attention sur la complexité de l'évaluation pour les deux domaines envisagés. Celle-ci est plus grande que pour d'autres tâches du TAL, notamment à cause de la subjectivité dans le jugement de la grammaticalité et de la conservation de l'information. Un certain nombre de mesures entièrement automatiques ont été créées, mais elles ne suffisent pas toujours à exprimer la qualité du résultat, et la fiabilité de plusieurs d'entre elles a été entachée (cf. point 2.5.1 et 3.5). Des raisons qui expliquent pourquoi, en RAT comme en CAP, on a souvent recours à l'évaluation manuelle.

Il convient à présent de présenter notre approche de la compression. Fort de l'état de l'art que nous achevons, nous décrivons l'implémentation de deux compresseurs de phrases. L'objectif est de produire des compresseurs autonomes, dont l'implémentation ne dépend pas d'un système de résumé automatique, mais donnant des résultats exploitables par des résumeurs extractifs.

Deuxième partie

**Implémentation de compresseurs
automatiques**

Introduction

Convaincu de l'intérêt de la CAP pour le RAT suite à l'analyse des nombreux travaux ci-dessus exposés, nous nous proposons de continuer le travail de recherche en compression de phrases pour le résumé automatique. Nous souhaitons, d'une part, démontrer empiriquement le bienfondé des hypothèses avancées dans la section 4.1, et, d'autre part, proposer de nouvelles méthodes que celles déjà envisagées par les chercheurs en TAL pour la compression.

Nous avons mené, à ce titre, deux expérimentations principales dans lesquelles nous avons implémenté des systèmes de compression de phrases. La première, décrite dans le chapitre 5, concerne un compresseur automatique conçu selon une approche symbolique. La seconde, présentée au chapitre 6, traite de l'implémentation d'un compresseur qui repose sur le formalisme des champs conditionnels aléatoires, une technique d'apprentissage automatique. Dans ces chapitres, nous exposons les raisons qui nous ont poussé à envisager deux approches distinctes et montrons leur originalité respective dans le champ de recherche actuel.

Grâce à ces expérimentations et aux résultats produits par chacun des systèmes qui seront présentés ci-après, nous pourrions juger de la validité de notre hypothèse de départ et répondre à plusieurs questions scientifiques autour de la compression automatique pour le résumé. Le but étant de donner de nouvelles pistes méthodologiques, qui permettront, à terme, de développer des compresseurs utilisables hors du champ de la recherche.

Chapitre 5

Compression symbolique

Introduction

Le système que nous présentons dans ce chapitre compresse des phrases à partir d'un jeu de règles et d'un algorithme conçus manuellement. Nous avons développé ce système à l'occasion d'un stage au sein du *Research Group for Computational Linguistics*¹ (RGCL), un centre de recherche en TAL situé à Wolverhampton au Royaume-Uni. Les ressources avec lesquelles nous avons travaillé durant le stage étant spécialisées pour l'anglais, le compresseur traite du texte écrit en anglais.

Pour cette première expérience, nous avons voulu nous pencher sur les cas de compressions de structures syntaxiques enchâssées, difficiles à appréhender automatiquement. Nous pensons qu'une approche manuelle peut constituer une solution intéressante pour ce genre de cas. Nous n'oublions pas pour autant de nous occuper de structures moins complexes, lorsque leur suppression est prévisible.

Ce premier système et son évaluation nous permettront d'identifier les avantages et les inconvénients de l'approche symbolique pour la CAP. Nous aurons l'occasion d'appréhender empiriquement les limites de cette approche, en analysant les cas pour lesquels une compression manuelle s'avère insuffisante. Nous prendrons acte de ces réflexions lors de la construction du deuxième système de compression.

La première section de ce chapitre expose notre méthodologie pour la conception du compresseur. Nous y justifions notre approche, détaillons les grandes étapes de l'implémentation ainsi que les réflexions qui les ont guidées. La seconde section présente les résultats des évaluations menées. Pour clôturer ce chapitre (5.3), nous mettons en relief les enseignements de l'expérience quant à l'utilité de l'approche pour la CAP et les difficultés d'évaluation.

1. <http://rgcl.wlv.ac.uk>.

5.1 Méthodologie

5.1.1 Compression de structures syntaxiques complexes

Si notre objectif est de produire un compresseur polyvalent, capable de traiter tout type de phrase, nous avons voulu nous attarder sur des cas difficiles à analyser et produire de façon automatique. Nous parlons de structures syntaxiques complexes comme les incises intra-phrases, les compléments circonstanciels ou les subordonnées non complétives (dont l'absence ne nuit pas à la grammaticalité). Elles représentent des éléments de langage qu'il nous paraît important de savoir repérer pour deux raisons. Tout d'abord, parce que c'est ce genre de structures que le compresseur veut éliminer en premier : leur présence ne garantit pas la grammaticalité, et l'information qu'elles donnent n'est que très rarement essentielle. En effet, l'auteur insiste rarement sur des éléments facultatifs grammaticalement et/ou placés dans une incise. Ensuite, parce que ces structures peuvent contenir beaucoup de mots, parfois plus que la phrase principale.

De tels éléments sont assez durs à repérer informatiquement sans envisager l'intervention de l'homme. Sans même évoquer l'influence du contexte de la phrase et du texte, on remarque que la rédaction d'incises et autres structures supprimables n'est pas codifiée et qu'une reconnaissance de patrons (*patterns*) basique ne peut résoudre le problème.

5.1.2 Compression via *sign tagging*

Lors de notre collaboration avec le RGCL, nous avons eu l'occasion d'employer un « tagueur de signes », un outil d'annotation syntaxique conçu par Evans & Orasan (2013). Un « signe » est ici considéré comme toute forme de mot ou de ponctuation qui introduit ou clôt une proposition ou un syntagme dans une phrase principale. Il s'agit donc de taguer, typiquement, les conjonctions de subordination et de coordination (*wh-words*, *and*, *or*...), mais aussi les virgules, et de renseigner pour chaque signe, le type de proposition qu'il introduit : subordination ou coordination, début ou fin de proposition, nature du constituant principal, etc. Une liste précise des tags et de leur signification se trouve en annexe. Nous pensons que cet outil, qui n'était pas développé pour la CAP au départ, pourrait servir à l'écriture de règles de compression de structures complexes, en garantissant un haut taux de grammaticalité.

Le tagueur de signes a, à l'origine, été développé pour servir un simplificateur de texte automatique. Dans le cadre du projet FIRST², la fonction principale de ce simplificateur résidait dans le découpage de phrases complexes en plusieurs phrases simples, en réglant les éventuels problèmes d'anaphores. La figure 5.1 donne un exemple de simplification faite par le programme. L'enjeu était de l'implémenter sans avoir recours à l'étiquetage syntaxique (Evans & Orasan 2013 : 1-3).

Le tagueur annote les signes en appliquant une série de règles apprises automatiquement. Trois corpus d'entraînement de registres différents ont été utilisés : un corpus de presse, d'articles de médecine et d'œuvres littéraires. Rapide, le tagueur s'avère assez performant, en particulier pour les conjonctions de subordination (Evans & Orasan 2013 : 20).

2. FIRST est un projet financé par la Commission européenne soutenant le développement d'outils informatiques destinés à faciliter la compréhension et l'accès à la lecture pour les personnes souffrant d'autisme et troubles similaires. Plus d'informations sur <http://rgcl.wlv.ac.uk/projects/FIRST/>.

Phrase originale : Mr Burstein complained about the article to the Press Complaints Commission, which dismissed his claim.

Simplification : Mr Burstein complained about the article to the Press Complaints Commission. The Press Complaints Commission dismissed his claim.

FIGURE 5.1 – Exemple de simplification via sign tagging

Si notre objectif diffère de celui de Evans & Orasan (2013), nous trouvons intéressante la façon dont le tagueur de signe délimite les incises et subordonnées des phrases. Pour notre compresseur, nous décidons donc de développer des règles qui prennent en compte le résultat du tagueur pour les virgules et les conjonctions de subordination. Nous excluons les conjonctions de coordination, ne pouvant supprimer de façon régulière les éléments qu'elles introduisent. Cette seule annotation ne pouvant suffire, nous employons également un étiqueteur syntaxique afin de pouvoir envisager une compression par élagage d'arbres de constituants (à la façon de Yousfi-Monod & Prince 2006). À partir de ces 2 pré-traitements, le système applique itérativement une série de règles écrites manuellement à chaque phrase du texte. Ces règles sont en nombre très réduit grâce au haut niveau d'abstraction avec lequel le système travaille et s'appuient quasi exclusivement sur le repérage de signes tagués, qui servent en ce sens de déclencheurs pour une ou plusieurs opérations de compression. La figure 5.2 reprend les différentes étapes à la compression, depuis le pré-traitement jusqu'à l'étape finale de post-traitement, qui consiste à retirer toutes les annotations effectuées au préalable.

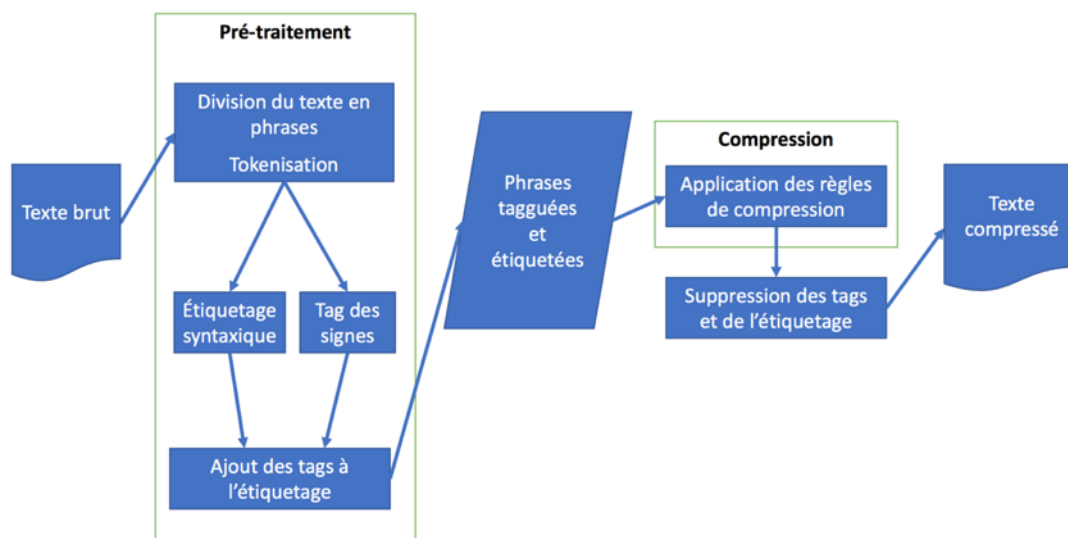


FIGURE 5.2 – Structure générale du compresseur de phrases

5.1.3 Outils d'annotation

Nous listons ici les différents outils employés pour la construction du compresseur et que nous n'avons pas implémentés nous-même, en omettant le tagueur de signes.

- *TTT2* : *TTT2* est un outil pour le TAL conçu par le *Language Technology Groupe* de l'université d'Édimbourg, dont la dernière version³ en distribution libre date de 2008. Il contient une librairie de scripts pour des opérations courantes de traitement de texte. Ici, on utilise un programme de découpe en paragraphes et phrases et un tokeniseur.
- *Stanford Unlexicalized Parser* : L'étiqueteur syntaxique pour la représentation en arbres syntaxiques des phrases est celui développé par l'université de Stanford, disponible en source libre⁴ (Klein & Manning 2003). Nous utilisons le parseur PCFG. Le programme s'occupe également d'assigner des parties de discours à chaque mot et syntagme (POS tagging).
- *Tsurgeon* : Autre programme en distribution libre issu de l'université de Stanford⁵, *Tsurgeon* s'emploie pour la manipulation d'arbres syntaxiques via le formalisme des *Tregex*, que l'on peut concevoir comme des expressions régulières pour structures arborescentes (Levy & Andrew 2006). C'est l'outil avec lequel nous avons rédigé les règles de compression et qui effectue concrètement les opérations de réduction par l'exécution de scripts *Tsurgeon*. Chaque script contient une règle de compression, qui se compose d'une expression *Tregex* et d'une instruction d'élagage de l'arbre syntaxique courant. Celle-ci n'est appliquée que lorsque l'expression est reconnue. La figure 5.3 illustre ce processus.

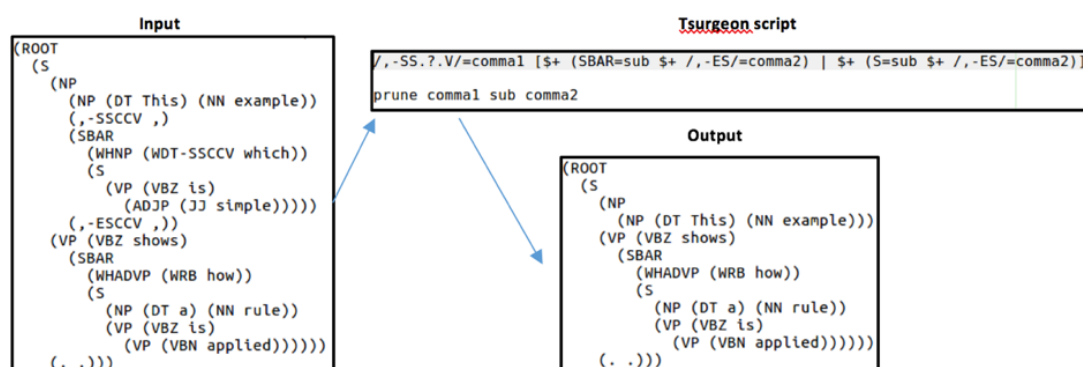


FIGURE 5.3 – Application d'une règle de compression sur une instance taguée et étiquetée

5.1.4 Justification de la méthodologie

Le fait d'envisager une approche par règles en CAP de nos jours constitue une singularité dans ce champ de recherche. Et pour cause : de plus en plus délaissée face à la rapidité et, dans certains cas, la meilleure efficacité des approches automatiques, peu de chercheurs s'emploient à défendre cette approche en CAP ou en RAT, comme le montre notre état de l'art (cf. partie I). Dans cette expérience, nous cherchons moins à promouvoir l'approche entièrement manuelle en TAL qu'à mettre en avant ses qualités et ses défauts pour le cas particulier de la compression.

3. L'outil est téléchargeable à l'adresse suivante : www.ltg.ed.ac.uk/software/lt-ttt2.

4. Voir la page <https://nlp.stanford.edu/software/lex-parser>.

5. Voir la page <https://nlp.stanford.edu/software/tregex>.

D'un point de vue théorique, l'avantage de la compression manuelle se situe principalement dans la meilleure garantie de grammaticalité du résultat. Puisque l'homme s'occupe de la phase de compression proprement dite, on s'attend à une qualité grammaticale supérieure à celle des systèmes automatiques. Le second avantage, c'est que la complexité syntaxique des segments détériore moins la qualité du résultat qu'elle ne le fait dans des systèmes automatiques. Dans l'approche probabiliste, plus la structure grammaticale est plus complexe, plus on aura besoin d'exemples de qualité pour espérer obtenir un résultat correct. Ce problème ne concerne pas le linguiste qui pourra modéliser cette structure de façon synthétique.

Notre approche nous semble d'autant plus innovante qu'elle emploie un outil d'analyse syntaxique original, donnant accès à un type d'information jamais utilisé auparavant dans ce type d'approche. Il s'agit du tagueur de signes : il est évident que les virgules et structures d'enchâssements représentent des marqueurs forts de séquences potentiellement compressibles. L'avantage duquel nous profitons avec ce tagueur se situe dans sa rapidité à reconnaître ses segments, mais aussi dans les informations supplémentaires qu'il nous donne dessus. Pour tout signe identifié, on peut savoir s'il marque le début ou la fin d'un complément, la nature du syntagme qu'elle suit ou précède et sa portée. Si la précision du tagueur n'est pas haute, nous avons fait en sorte de ne prendre en compte, pour la compression, que les signes que le tagueur traitait le mieux, à savoir les virgules et conjonctions de subordination (voir Evans & Orasan 2013 : 20). Nous gérons également certaines erreurs du tagueur grâce aux informations syntaxiques.

Par ailleurs, l'emploi conjoint du tagueur de signes et de l'analyseur syntaxique donne accès à un haut niveau d'abstraction en un temps raisonnable. Ce niveau d'abstraction permet de réduire la variation infinie de la langue à un nombre restreint de possibilités. Or, un ensemble de règles ne peut être efficace que s'il envisage une écrasante majorité des cas. Concrètement, dans notre système, une quinzaine de règles permettent de traiter la plupart des cas de compressions syntaxiques, là où par exemple, le simplificateur de Evans et Orasan, basé uniquement sur l'annotation fournie par le tagueur de signes, en compte plus d'une centaine (Evans & Orasan 2013 : 22).

Grâce à cela, nous avons, en outre, significativement réduit le temps de conception de notre système. Il nous paraît important de souligner cet élément, car le temps de développement de systèmes symboliques est souvent désigné, parfois à juste titre, comme un défaut de l'approche manuelle en TAL. Si l'on tient compte du temps passé à récolter des données annotées nécessaires pour une approche statistique, on pourrait même affirmer que notre méthode est plus rapide. Cet argument, ajouté aux bons résultats de l'approche manuelle sur le plan grammatical, achève de démontrer la validité de la méthodologie présentée précédemment.

5.2 Évaluation

5.2.1 Méthodologie

Une fois le compresseur terminé, nous l'avons fait annoter un corpus d'évaluation. Il s'agit d'un ensemble de 53 textes de 250 à 400 mots issus du corpus anglais METER (Gaizauskas *et al.* 2001), qui contient une série de résumés d'audience au tribunal ou d'articles de presse. Dans ces 53 textes, 372 phrases ont été compressées par le système.

Le corpus utilisé pour l'évaluation n'étant pas annoté pour la compression, l'évaluation s'est faite manuellement. Nous n'avons pas eu à notre disposition, durant le développement du compresseur, de corpus anglais qui nous aurait permis d'évaluer automatiquement. Le résultat du compresseur a été donc annoté par deux linguistes stagiaires volontaires au RGCL : une appréciation (*Très bien*, *OK*, *Mauvais*) a été donnée à chaque phrase pour la grammaticalité et pour la sémantique ou le maintien de la cohérence textuelle. L'appréciation *Mauvais* a été attribuée à des phrases compressées agrammaticales ou affectant très négativement la cohérence du texte. L'appréciation *OK* a été assignée aux phrases grammaticalement acceptables ou qui ne suppriment qu'une nuance légère du texte. *Très bien* signifie que la phrase proposée ne pose aucun problème particulier. Les annotateurs n'ont pas annoté les mêmes parties du corpus d'évaluation, faute de temps. En conséquence, l'accord inter-annotateur n'a pu être établi, bien que cela eût été intéressant pour cette phase d'évaluation.

En ce qui concerne l'appréciation de la cohérence, les annotateurs disposaient du texte original ainsi que du texte compressé, séparé en phrases, mais dans l'ordre original. Ils pouvaient donc se faire une idée précise du sujet du texte et de son contenu pour attribuer une note à chaque phrase compressée.

Nous n'avons pu envisager d'évaluation extrinsèque pour ce compresseur et mesurer son influence dans une tâche de résumé. Nous n'avons effectivement pas eu accès à un résumeur « état de l'art » en anglais avec lequel faire cette expérience.

5.2.2 Résultats

Les résultats de l'évaluation sont repris dans le tableau 5.1. D'un point de vue grammatical, on voit que 313 phrases (84 % du total) ont reçu la mention *Très bien*, 45 *OK* et 14 *Mauvais*. La méthode semble donc, comme attendu, assez fiable au niveau de la grammaticalité des phrases produites. D'un point de vue sémantique, on observe 215 (58 %) phrases jugées comme *Très bien*, 103 (27,5 %) comme *OK* et 54 (14,5 %) comme *Mauvaises*. Le résultat est donc moins probant, mais il est tout de même intéressant de voir que la compression d'une large majorité de phrases ne pose pas de problème majeur dans la compréhension d'un texte, et ce sans appliquer de contraintes sémantiques.

	Grammaticalité			Sémantique/Cohérence			Total
Appréciation	Très bien	OK	Mauvais	Très bien	OK	Mauvais	/
Fréquence brute	313	45	14	215	103	54	372
Fréquence relative	84 %	12 %	4 %	54 %	27,5 %	14,5 %	100 %

TABLE 5.1 – Résultats de l'évaluation manuelle

Les erreurs de compression au niveau grammatical sont majoritairement dues à des erreurs dans le prétraitement. Le tagueur comme le parseur sont en effet sujets à un taux d'erreur faible, mais tout de même existant, et celui-ci augmente en fonction de la longueur et de la complexité syntaxique des phrases. Étant donné qu'il s'agit précisément du type de phrases que nous traitons, on pouvait s'attendre à ce que ce taux d'erreur ait plus de répercussions que sur un texte ordinaire. De plus, il arrive parfois que les sorties du tagueur et du parseur ne coïncident pas, ce qui bloque la reconnaissance de patrons (*patterns*) syntaxiques et donc l'application de règles de compression. On peut prendre la phrase suivante pour illustrer ce deuxième type d'erreur :

Giving evidence in a criminal case brought against his father, Christopher Grant, 15, a businessman, and his stepmother, Meryl, 43, Mr Thomas told Cardiff Crown Court that the family subsisted on takeaway meals.

Ici, la complexité ressort de la juxtaposition de deux longues appositions et du fait que le système ne sait pas taguer doublement les virgules. Cela devient problématique lorsqu'une virgule marque la fin d'une incise en même temps que le début d'une autre : dans ce cas, le tagueur de signes ne note qu'une seule des frontières. Cela gêne le repérage d'appositions ou en délimite les frontières de façon erronée.

Au niveau de la cohérence textuelle, un certain nombre d'erreurs sont dues à des phénomènes supra-phrastiques, dont la présence n'est pas marquée dans la phrase. Certaines compressions posent en effet des problèmes au niveau de la cohésion et/ou de la cohérence par rapport au texte original. Nous évoquions déjà ces deux caractéristiques du texte au point 2.6, en parlant de l'extraction de phrases. Les remarques que nous y faisons valent également pour la CAP. Voici un exemple de compression produite par notre système posant des problèmes de cohérence.

Phrase originale : In the UK, academics have found that it takes three years for a child to acquire a basic level of competence [in spelling]. (*Au Royaume-Uni, les chercheurs ont remarqué qu'il fallait trois ans à un enfant pour acquérir un niveau de compétence de base [en orthographe].*)

Compression : Academics have found that it takes three years for a child to acquire a basic level of competence.

FIGURE 5.4 – Exemple d'erreur de cohérence

Il faut savoir que l'article traite de l'orthographe anglaise et de ses irrégularités, en la comparant avec celle d'autres langues. Si l'on supprime *In the UK* (« Au Royaume-Uni »), la phrase reste grammaticalement correcte, mais change de sens : ce ne sont plus les enfants britanniques qui prennent trois ans pour acquérir un niveau basique en orthographe, mais les enfants en général. Ce genre d'erreurs ne peuvent être évitées avec une approche purement syntaxique. Toutefois, ce type de cas ne s'est présenté que peu de fois dans notre évaluation.

5.3 Discussion

Suite à cette première expérience dans la construction d'un système de compression original, nous pouvons confirmer plusieurs de nos hypothèses.

Tout d'abord, celle de l'avantage de la compression manuelle pour la grammaticalité. Produire des phrases correctes est une spécification primordiale du problème de la compression. C'était pourtant l'un des points problématiques de plusieurs travaux en CAP pour le RAT que nous avons mentionnés précédemment, notamment Vanderwende *et al.* (2006) et Durret *et al.* (2016). Avec notre compresseur, nous obtenons un taux de grammaticalité très haut, avec seulement 4 % de phrases jugées incorrectes.

En ce qui concerne la sémantique, on observe de bons résultats également, avec plus de 80 % de phrases jugées bonnes ou acceptables au niveau de la cohérence textuelle. Cela confirme notre hypothèse selon laquelle les segments complexes non principaux peuvent, la plupart du temps, être supprimés sans aucune analyse du contexte. Les erreurs repérées

par les annotateurs sont majoritairement dues à des éléments de contexte, impossible à prendre en compte avec une approche purement syntaxique de la compression.

Le désavantage majeur de l'approche se situe, selon nous, au niveau du temps d'exécution. Comme notre système requiert une annotation syntaxique relativement lourde, toute phrase à compresser doit préalablement être soumise à un prétraitement dont la durée augmente fortement avec la taille. Néanmoins, la vitesse des processeurs augmentant d'année en année, cet inconvénient devrait se faire de moins en moins lourd.

Cette limite mise à part, nos résultats montrent la validité de notre approche. En s'appuyant sur une analyse syntaxique fine de la phrase, il est possible de construire un jeu de règles de taille réduite et performant pour des structures complexes. L'approche se justifie particulièrement pour le traitement ciblé de types précis de structures, pour des cas où le contexte n'est pas pertinent.

Conclusion

Nous avons fini de présenter notre premier compresseur automatique de phrases. Implémenté selon une approche symbolique, le compresseur est spécialisé dans le traitement de structures enchâssées, généralement difficiles à traiter avec l'approche automatique. L'évaluation manuelle a montré que, sur le plan grammatical, le compresseur offre de bons résultats. Au niveau de la cohérence du texte, les résultats sont étonnamment hauts (14,5 % d'erreurs graves), quand on considère qu'aucun critère sémantique n'est pris en compte pour choisir les mots à supprimer. Ces résultats prouvent que, lorsqu'on spécialise un compresseur pour quelques types particuliers de structures, l'approche manuelle se révèle pertinente.

Le défaut principal de notre compresseur réside dans son besoin d'une annotation syntaxique plus lourde que la moyenne. Cela tend à ralentir ses performances et diminue ainsi ses champs d'utilisation. L'annotation de la syntaxe étant une tâche assez dépendante de la langue, elle rend notre technique plus difficilement adaptable à d'autres langues.

Nous voulons à présent voir s'il est possible d'obtenir des résultats équivalents ou meilleurs avec l'approche statistique de la CAP. De plus, puisque nous nous sommes concentré sur quelques types de structures syntaxiques, nous souhaitons également vérifier si une approche statistique peut produire un compresseur plus polyvalent, capable de s'occuper de structures variées et ne dépendant pas que de l'étiquetage syntaxique.

Afin d'évaluer ces hypothèses, nous implémentons dans le chapitre 6 un second compresseur automatique de phrases, basé cette fois-ci sur un modèle probabiliste. Une fois ce système évalué, nous pourrons alors comparer les avantages et inconvénients de chaque méthode.

Chapitre 6

Compression probabiliste

Introduction

Prenant le contre-pied de l'approche présentée précédemment, nous mettons en place, dans ce chapitre, un système de compression probabiliste. Il ne s'agit plus de créer de toutes pièces un jeu de règles, mais plutôt de le définir de façon statistique en observant des données annotées manuellement. C'est la technique de l'apprentissage automatique supervisé.

Nous souhaitons également améliorer notre façon d'évaluer le système, en présentant plus de mesures que pour le compresseur précédent, notamment le rappel et la mesure F_1 . Nous envisageons pour cela une évaluation automatique, en plus de l'évaluation manuelle. Les mesures automatiques donneront une meilleure idée des capacités du compresseur, en particulier pour le rappel, qui indique le pourcentage de mots qui ont été supprimés par rapport à l'ensemble des mots qui auraient pu être supprimés dans un corpus.

Comme la littérature en CAP s'avère plus fournie pour ce type d'approche, nous nous sommes démarqué dans notre travail en créant un compresseur pour le français. En effet, la grande majorité des systèmes existants traitent l'anglais. Ceci rend le système produit et les conclusions de cette expérience plus novateurs.

Un descriptif précis des méthodes que nous avons utilisées pour construire ce compresseur probabiliste pour le français est exposé à la section suivante. Nous y présentons notre démarche et le corpus utilisé pour l'entraînement et décrivons le formalisme des champs aléatoires conditionnels (ou CRF pour *Conditional Random Fields*), sur lequel repose notre système. Nous passons ensuite à la partie proprement expérimentale, où plusieurs modèles sont évalués dans le but de trouver la configuration optimale pour le problème de la compression de phrases. Nous concluons en expliquant ce que nos manipulations nous apprennent sur la CAP.

6.1 Méthodologie

6.1.1 Compression par apprentissage automatique

L'élaboration de notre compresseur par apprentissage automatique passe par plusieurs étapes distinctes. La première étape, absente pour la compression symbolique, est de trouver un corpus d'entraînement de taille suffisante. Dans notre cas, nous devons trouver un corpus en français, une langue bien moins dotée que l'anglais en TAL et pour laquelle les ressources en CAP s'avèrent très limitées. Un tel corpus sert à entraîner notre modèle statistique.

La phase suivante consiste à trouver la valeur des paramètres du modèle, qu'un compresseur pourra ensuite utiliser pour annoter des phrases jamais vues. Avant l'entraînement proprement dit, il faut d'abord sélectionner un ensemble de « facteurs » (*features*) sur lesquels se baser pour calculer les probabilités de compression. Pour guider la sélection de facteurs, il faut déterminer une ou plusieurs méthodes d'évaluation qui rendront compte de l'efficacité du modèle.

Avant de détailler chacune de ces étapes et de présenter les outils nécessaires à leur réalisation, il nous paraît important de justifier notre méthodologie. En particulier, nous motivons le choix du modèle CRF ainsi que celui du français.

6.1.2 Justification de la méthodologie

Cette deuxième expérience suit également une approche originale dans le domaine de la compression, mais à d'autres égards que la précédente. Tout d'abord, les CRF n'ont pas fait l'objet de recherches intensives en CAP. Pour l'anglais, nous notons les seuls articles de Nomoto (2007) et Li *et al.* (2013). Nous suivons une approche semblable à ces deux études sur la CAP. Pour le français, aucun système de compression basé sur le modèle CRF n'a vu le jour, à notre connaissance.

Cette technique nous paraît néanmoins pertinente pour l'annotation automatique de séquences, sans qu'il soit nécessaire de requérir des jeux de données de taille colossale. En effet, les CRF, comme nous l'expliquons après (voir section 6.2), sont spécialisés dans la classification de séquences et tiennent mieux compte des longues dépendances que des modèles plus classiques tels que les HMM ou la régression linéaire. Ils ont d'ailleurs été utilisés avec succès dans d'autres tâches de classification en TAL (reconnaissance d'entités nommées, reconnaissance d'opinions...).

Nous présentons, en outre, un compresseur traitant des textes de langue française. On remarque en effet que, dans les domaines du RAT et de la CAP, les études réalisées pour l'anglais prédominent. Tant au niveau de la mise à disposition de données annotées que pour l'implémentation de systèmes, une large majorité d'études proposent des solutions uniquement pour l'anglais. Cette observation s'applique d'ailleurs pour la plupart des domaines du TAL. Si ce phénomène s'explique sans doute par des raisons historiques et pratiques, il n'en reste pas moins important, si l'on veut créer des systèmes performants pour le français, de disposer de données en français, mais aussi de savoir répliquer et, au besoin, d'adapter les techniques anglophones aux systèmes francophones. En présentant un système pour le français, nous explorons donc une voie très peu empruntée dans l'approche automatique de la CAP.

Le traitement du français rend, par ailleurs, le choix de l'utilisation de CRF pour l'implémentation du compresseur encore plus pertinent. Actuellement, très peu de données annotées pour la compression existent pour le français. Le corpus que nous utilisons (présenté ci-après) pour entraîner le système représente un des seuls disponibles et le plus volumineux disponible dans cette langue. Or, la taille de celui-ci semble inadéquate pour des modèles d'apprentissage automatique plus complexes que les CRF. À titre d'exemple, Filippova *et al.* (2015) ont présenté un compresseur anglophone basé sur un réseau de neurones entraîné sur plus de 2 millions d'exemples. En revanche, les 8432 phrases originales de notre corpus et chacune de leurs 4 compressions représentent un nombre tout à fait valable pour l'entraînement de modèles CRF.

6.1.3 Corpus d'entraînement

Origine et description des données

Nous employons pour notre système un corpus conçu par De Loupy, Guégan, Ayache, Seng & Torres-Moreno, une équipe de chercheurs français¹. Le corpus a été construit en 2010 à des fins évaluatives, dans le cadre du projet RMP2, un projet de recherche soutenu par l'ANR (Agence Nationale de Recherche française) qui finance le développement de résumeurs multi-documents (De Loupy *et al.* 2010 : 3113-3114). Il se compose de 8432 phrases extraites de 400 articles issus de journaux d'actualité français. Chaque phrase a été sortie de son contexte pour ensuite être compressée par 4 annotateurs différents. Il s'agit de francophones natifs, à qui l'on a demandé de supprimer les mots des phrases en s'assurant que le résultat soit cohérent, grammatical et contenant toujours l'essentiel de l'information présentée. Aucun autre type de modification n'était autorisé. Dans le cas où un annotateur convenait qu'une phrase ne pouvait pas du tout être compressée, il avait la liberté de ne pas le faire.

Ce corpus représente un des seuls échantillons de textes annotés pour la compression existant actuellement. Sa taille est également remarquable comparée à une majorité de corpus (De Loupy *et al.* 2010 : 3115), rendant possible son utilisation pour le genre d'approche que nous envisageons. Le fait que l'on dispose de plusieurs compressions pour une même phrase s'avère également assez singulier comparé aux corpus anglophones, et nous semble intéressant d'un point de vue théorique, car elle autorise des stratégies de compression distinctes et rend compte de la part de subjectivité dans chacune d'elle. En pratique, en revanche, il apparaît plus compliqué de mettre à profit cette diversité, due à la difficulté de modéliser celle-ci par des probabilités.

On remarquera enfin que le corpus contient des exemples très représentatifs du genre journalistique : on y retrouve beaucoup de noms propres, de citations et parenthèses... Des phrases très courtes (parfois sans verbe ou nom) sont également présentes. Nous considérons la présence de tels cas difficiles à traiter, mais correspondant à l'usage réel, comme un avantage pour l'entraînement de notre modèle du point de vue de sa robustesse.

Statistiques

Pour donner une meilleure idée du contenu du corpus, nous reprenons plusieurs statistiques descriptives du corpus exposées par De Loupy *et al.* (2010 : 3115-3117).

1. Nous remercions chaleureusement M. Torres-Moreno de nous avoir donné accès à ce corpus.

Les phrases originales contiennent, en moyenne, 26,2 tokens, avec une variation assez élevée ($\sigma = 15,25$; la plus grande phrase contient 130 tokens). La table suivante affiche la répartition des catégories de discours les plus présentes dans le corpus :

Nom	Prép.	Ponct.	Verbe	Déterminant	Pronom	Adjectif	Adverbe	Conj.	Chiffre
19,2	15,9	14,4	13,6	11,3	5,8	5,4	4,2	3,1	1,8

TABLE 6.1 – Catégories de discours des mots du corpus (en %)

Les phrases compressées contiennent entre 18 et 22 mots en moyenne selon les annotateurs. La table 6.2 donne diverses informations sur le taux de compression² en fonction de l'annotateur. Comparés à d'autres corpus disponibles, les annotateurs ont supprimé moins de mots et ont laissé de nombreuses phrases inchangées (environ un quart du corpus). Si l'on s'attarde sur la longueur des segments supprimés, on remarque que la plupart des compressions concernent des segments courts (1-4 mots), les cas de suppression de séquences longues (> 40 mots) se bornant souvent aux énumérations. La table 6.3 donne les contingences entre annotateurs, calculées sur le nombre de mots communs supprimés ou conservés. Si l'on calcule le coefficient ϕ à partir de ces données (ce qui revient à faire une corrélation de Pearson), on obtient des taux de corrélation entre annotateurs avoisinant tous les 50 % de corrélation (De Loupy *et al.* 2010 : 3115). Cette relative indépendance entre les annotateurs dénote de la diversité de leurs stratégies de compression. Une analyse plus poussée montre qu'elles varient notamment par rapport à la longueur de la phrase. Si, pour les phrases courtes, les annotateurs suppriment les mots aux mêmes positions, cette tendance s'affaiblit plus la phrase à compresser est longue (De Loupy *et al.* 2010 : 3117).

Annotateur	A1	A2	A3	A4
Taux de compression (τ) (en %)	75	84,4	71	79
τ moyen par phrase	80,3	86,5	77,1	82,5
Tokens par phrase après compression	19,7	22,1	18,6	20,7
τ le plus bas pour une phrase	12	26,9	7,8	17,1
Phrases non compressées	2119	2142	1630	1775
Phrases non compressées (en %)	25,1	25,4	19,3	21

TABLE 6.2 – Taux de compression par annotateur

Le tableau 6.4 indique les taux de compression en fonction de la catégorie de discours, soit le pourcentage de cas où ce type de mots n'a pas été éliminé. Sans grande surprise, les adverbes et les adjectifs se retrouvent en haut du classement, tandis que les verbes, les déterminants et les pronoms, souvent nécessaires à la grammaticalité des phrases, n'ont que très rarement été supprimés.

2. Pour rappel, le taux de compression (ou τ) se définit comme la proportion de texte conservé (cf. équation 3.3). Tout au long de ce chapitre, nous calculons τ sur base des tokens. Un $\tau < 50\%$ indique par exemple que la compression a supprimé plus de la moitié des mots de la phrase originale.

A1-A2	Supprimé	Conservé	Total	A2-A3	Supprimé	Conservé	Total
Supprimé	10,7	14,3	24,9	Supprimé	10,1	5,5	15,5
Conservé	4,9	70,2	75,1	Conservé	18,8	65,5	84,5
Total	15,5	84,5	100	Total	28,9	71,1	100
A1-A3	Supprimé	Conservé	Total	A2-A4	Supprimé	Conservé	Total
Supprimé	15,7	9,2	24,9	Supprimé	9,7	5,8	15,5
Conservé	13,2	61,9	75,1	Conservé	11,3	73,2	84,5
Total	28,9	71,1	100	Total	21	79	100
A1-A4	Supprimé	Conservé	Total	A3-A4	Supprimé	Conservé	Total
Supprimé	14,5	10,4	24,9	Supprimé	13	15,9	28,9
Conservé	6,4	68,6	75,1	Conservé	7,9	63,1	71,1
Total	21	79	100	Total	21	79	100

TABLE 6.3 – Table de contingence entre annotateurs (en % de tokens)

Adverbe	Chiffre	Ponct.	Adjectif	Conj.	Nom	Prép.	Pronom	Déterminant	Verbe
55,5	63,8	69,5	70,8	74,8	76,5	76,8	80	81,5	87,2

TABLE 6.4 – Taux de compression par catégorie de discours

Corpus d'entraînement : compressions maximales

Pour le développement d'un modèle CRF, nous ne pouvons avoir un corpus d'entraînement contenant des phrases identiques, mais annotées différemment. Nous adaptons donc le corpus pour le modèle en ne retenant que, pour chaque phrase, la compression la plus courte des quatre proposées. Nous suivons ici la méthodologie de Li *et al.* (2013), qui disposaient d'un corpus anglophone similaire au nôtre. Nous pensons que ce corpus de compressions maximales constitue un meilleur corpus d'entraînement qu'un corpus ne reprenant que les compressions d'un seul annotateur, parce qu'il possède des propriétés *a priori* plus intéressantes pour la CAP.

On peut le voir avec la table 6.5, qui montre les mêmes mesures données précédemment pour le corpus de De Louty *et al.*, mais pour le corpus de compressions maximales. On remarque que les taux de compression sont nettement inférieurs à n'importe quel annotateur du corpus : le τ moyen par phrase passe de 77 % pour l'annotateur qui avait le plus compressé à 69 %. On a également environ deux fois moins de cas de phrases non compressées, qui ne représentent maintenant plus qu'un douzième du corpus.

Un taux de compression le plus bas possible paraît, dans le contexte de l'apprentissage automatique, pertinent pour deux raisons. Tout d'abord, parce que, comparé aux autres corpus disponibles en anglais, le taux de notre corpus est plus élevé. Or, plus le τ moyen s'éloigne de la moyenne, moins nos résultats seront comparables avec les autres études en CAP. Les conclusions que nous en tirerions s'en trouveraient moins intéressantes. Nous préférons, d'autre part, que le corpus d'entraînement contienne un maximum d'exemples de compressions. Les probabilités de compressions étant calculées à partir de ces exemples,

τ (en %)	63
τ moyen par phrase	69,8
σ du τ moyen par phrase	19,8
Tokens par phrase après compression	16,75
Phrases non compressées	1031

TABLE 6.5 – Statistiques pour le corpus de compressions maximales

un nombre restreint d'exemples biaiserait fortement le calcul de ces probabilités et nous mènerait à compresseur qui... ne compresse pas ou très peu.

Dès lors, nous choisissons le corpus de compressions maximales pour l'entraînement. Aucune autre modification n'a été apportée au corpus que celle qui vient d'être décrite, si ce n'est la correction de quelques coquilles pour 5 phrases (détaillées en annexe).

6.2 Champs conditionnels aléatoires

6.2.1 Présentation du modèle

Les champs aléatoires conditionnels (souvent abrégés CRF pour *Conditional Random Fields*) sont un modèle probabiliste dédié à la prédiction de séquences de labels. Ils s'utilisent pour des tâches de classification : ils produisent un vecteur $\mathbf{y}$ constitué de valeurs issues d'un alphabet $\mathbf{Y}$, un ensemble de variables de sortie, à partir d'un vecteur $\mathbf{x}$ constitué de valeurs issues d'un alphabet $\mathbf{X}$, un ensemble de variables observées (ou variables d'entrée). Ils sont donc pratiques dans de nombreuses tâches du TAL (POS tagging, reconnaissance d'entités nommées...), mais aussi dans d'autres domaines de l'ingénierie informatique comme la bioinformatique (prédiction de séquences d'ADN) ou l'analyse automatique d'image (identification de régions dans une image) (Sutton *et al.* 2012 : 268-270).

Présenté par Lafferty, McCallum & Pereira en 2001, ce formalisme a connu un succès important, et l'on continue aujourd'hui encore à l'utiliser, même si le développement récent de modèles probabilistes plus sophistiqués comme les réseaux de neurones tend à diminuer leur emploi dans la recherche en TAL. En 2001, les CRF sont introduits comme une structure de prédiction de séquences reprenant les avantages des techniques de labellisation de séquences classiques comme les HMM et la régression logistique, sans souffrir du problème du biais de label (*label bias problem*). Ce problème se formule comme suit : lorsque, étant donné un label y_i du vecteur $\mathbf{y}$, le nombre de labels possibles pour y_{i+1} est faible, le modèle ne prend pas ou peu en compte les données observées³.

On comprendra mieux avec un exemple simplifié, inspiré de Lafferty *et al.* (2001 : 4). Soit le modèle à états finis représenté à la figure 6.1. Il permet de tokeniser les mots *lus* et *las*. Supposons maintenant que nous voulons tokeniser la séquence d'observation [l,u,s]. Pour le premier caractère, la probabilité de passer de l'état initial à l'état 1 ou 3 est égale. Lors de la seconde transition, on n'a, dans les deux cas, qu'une seule option possible (le *u* pour

3. *[S]tates with low-entropy next state distributions will take little notice of observations.* (Lafferty *et al.* 2001 : 4)

le premier chemin, le *a* pour le second). La probabilité de chaque option n'est donc pas modifiée, et ce peu importe si une observation est plus fréquente qu'une autre, que *lus* (participe passé de *lire*) ait été plus fréquemment rencontré dans l'entraînement que *las* (adjectif formel). Ainsi, la probabilité que $[l,u,s]$ soit tokenisé en *lus* est la même que pour *las*. Dans ce cas, l'observation est donc tout simplement ignorée, une situation indésirable pour toute tâche de classification.

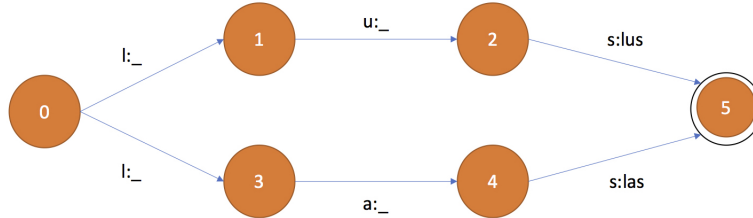


FIGURE 6.1 – Illustration du problème de biais de label. Les paires *observation-label* sont notées $o : l$.

De façon générale, puisque la somme des probabilités de transition d'un état vers un ensemble d'autres états doit toujours valoir 1, plus cet ensemble est grand, plus les probabilités de transitions seront petites. Le chemin d'un état initial vers un état final aura ainsi une plus grande probabilité s'il passe par des états pour lesquels on ne compte qu'une ou deux possibilités de transition, comparé à un autre chemin comprenant des états avec une dizaine de possibilités de transitions. On peut affirmer cela sans même connaître la valeur des probabilités concernées. Donc, plus le nombre de possibilités de transition pour un état – ou, dans notre cas, un label – est grand, plus le modèle aura de chance de le sélectionner lors de la labellisation de x .

Les CRF présentent une solution à ce problème grâce à une normalisation des probabilités de transition. Les CRF, comme d'autres modèles de type graphique, représentent des distributions d'un grand ensemble de variables en un produit de fonctions locales ne dépendant chacune que d'une sous-partie de ces variables. Cela est rendu possible en identifiant les relations d'indépendance conditionnelle entre les variables. Le modèle calcule la probabilité d'une séquence de labels selon l'équation 6.1 (Sutton *et al.* 2012 : 274), où Ψ est un ensemble de fonctions locales de taille A , dépendant chacune de y_a , un sous-ensemble de y . Là où les CRF se différencient des modèles classiques, c'est dans l'inclusion de la division par Z , un facteur normalisateur. La valeur de ce facteur dépend directement de Ψ et de A , qui représentent les probabilités transitions et le nombre de transitions.

$$p(y) = \frac{1}{Z} \prod_{a=1}^A \Psi_a(y_a) \quad ; \quad Z = \sum_y \prod_{a=1}^A \Psi_a(y_a) \quad (6.1)$$

Au-delà du problème de biais de label, Les CRF peuvent être considérés comme un technique combinant les avantages des modèles graphiques à ceux de l'approche discriminative. Le grand avantage des modèles graphiques est de réduire le nombre de variables à analyser en même temps pour le calcul d'une probabilité, grâce à la représentation de la distribution en un ensemble restreint de fonctions locales (Sutton *et al.* 2012 : 272-278).

Les modèles statistiques dits « discriminatifs » s’opposent, quant à eux, aux modèles « génératifs » dans leur façon de calculer une probabilité conditionnelle $p(x|y)$. Les premiers calculent directement cette probabilité, tandis que les seconds modélisent la probabilité que y donne x , en utilisant le théorème de Bayes ($p(x|y) = p(x)p(y|x)$). En principe, les deux approches s’équivalent grâce au théorème de Bayes, mais en pratique, il y a une différence non négligeable : l’un calcule la probabilité *a priori* du vecteur $\mathbf{x}$ ($p(\mathbf{x})$), l’autre non. Or, cette probabilité peut s’avérer très dure à modéliser. Par exemple, lorsque $\mathbf{x}$ est une suite de tokens, on peut rarement le faire, car la probabilité d’un mot dépend d’un nombre élevé de facteurs, tant grammaticaux que sémantiques. Les modèles discriminatifs sont donc avantageux lorsque la taille de $\mathbf{X}$ est grande (Sutton *et al.* 2012 : 278-283).

Ainsi, en assurant une complexité plus faible que les modèles graphiques génératifs, les CRF gagnent en rapidité et en performance dans la modélisation des probabilités conditionnelles par rapport à d’autres techniques disponibles.

En ce qui concerne l’entraînement des paramètres du modèle, il y a deux approches : les méthodes d’optimisation des probabilités maximales et les méthodes de gradient stochastique. Les premières nécessitent de parcourir l’ensemble du corpus d’entraînement pour ajuster les valeurs des paramètres, tandis que les secondes prennent des échantillons plus petits pour l’optimisation (Sutton *et al.* 2012 : 341). Nous ne rentrons pas dans les détails concernant l’entraînement du modèle CRF, et renvoyons le lecteur en quête d’exhaustivité à l’article très complet de Sutton *et al.* (2012 : 314-318).

6.2.2 Les CRF pour la CAP

Pour le problème particulier de la compression de phrases, un modèle statistique doit trouver, pour une suite de tokens observée, la suite compressée correspondante. $\mathbf{x}$ est donc un vecteur de tokens et $\mathbf{y}$ un vecteur encodant une suite d’informations binaires indiquant si un token doit être retenu dans la compression. Nous utiliserons tout au long du travail les labels 1 et 0 pour respectivement indiquer qu’un mot a été conservé ou supprimé. Rappelons que $\mathbf{y}$ n’a pas de taille de fixe, contrairement à d’autres problèmes de classification.

Généralement, le vecteur $\mathbf{x}$ n’encode pas réellement les tokens tels qu’ils apparaissent dans la phrase. Il contient plutôt, pour chaque token, un ensemble de « facteurs » (*features*) que l’ingénieur linguiste pense décisifs dans le choix de conserver un mot ou non. La sélection de ces facteurs et leur encodage dans un format exploitable représentent sans doute la partie la plus importante du développement du modèle CRF dans le cadre de la CAP. Pour notre sélection de facteurs, nous nous sommes notamment basé sur les travaux en compression de Nomoto (2007) et Li *et al.* (2013).

Nomoto (2007) présente une dizaine de facteurs pour son modèle. Ils consistent presque tous en des mesures booléennes et reposent sur une analyse syntaxique faible : on note si le mot commence avec une majuscule, finit avec une ponctuation, est constitué uniquement de chiffres ou d’une lettre majuscule et s’il est tagué comme nom, verbe ou pronom. Un facteur particulier indique si le mot est le verbe *die*. Ceci semble plus correspondre à une particularité du corpus d’entraînement ou de genre qu’à un motif théorique. Un dernier facteur intéressant est celui que l’auteur nomme le « facteur de constituant » (*chunk feature*). Son but est de mieux prendre en compte la grammaticalité en indiquant si le mot courant commence ou se trouve dans un constituant de l’arbre syntaxique. Concrètement, on utilise le format BIO, et ce pour chaque type de constituant dans lequel le mot

se trouve (Nomoto 2007 : 1578-1579). Prenons par exemple l'arbre syntaxique dessiné en figure 6.2. Les tokens *Achille* et *sa* seront tagués *B-NP* tandis que *colère* sera tagué *I-NP*.

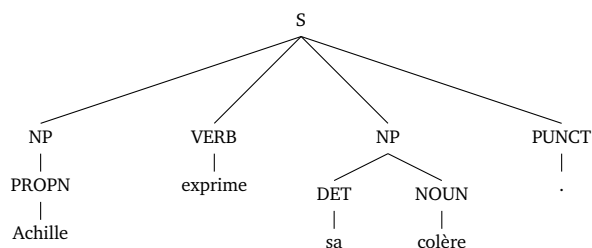


FIGURE 6.2 – Un arbre syntaxique

Le travail de Li *et al.* (2013) utilise des facteurs plus conventionnels pour le TAL : on prend en compte des n -grams de mots et de catégories de discours, le type d'entité nommée, si le mot est un mot vide, la profondeur de l'arbre syntaxique, si le mot est dans une proposition enchâssée, ainsi que des informations présentes sur l'arbre de dépendance (Li *et al.* 2013 : 494).

6.3 Facteurs et outils d'annotation

Nous listons ici les facteurs que nous avons envisagés pour notre modèle CRF à la compression, ainsi que les outils d'annotation qui nous permettent de les extraire du texte brut. Nous expliquons dans le point suivant quels facteurs seront retenus pour le modèle final, après avoir testé leur efficacité.

La table 6.6 reprend les facteurs utilisés pour nos expériences. Chaque facteur est une fonction retournant un booléen ou un nombre entier ou décimal.

La « position relative » est le résultat arrondi de la division de la position du mot par la longueur de la phrase. La « forme de référence » équivaut au lemme, mais non désambiguïsé. Lorsque plusieurs lemmes existent pour une forme donnée, la forme de référence est le lemme le plus fréquent. Pour la mesure de la fréquence, plutôt que de retenir le nombre décimal en entier, nous l'avons simplifiée en trois sous-fonctions booléennes : la première retourne *Vrai* si la fréquence de référence est haute, la seconde si elle est moyenne et la troisième s'il s'agit d'un mot rare ou inconnu. Cette simplification s'est avérée plus efficace dans nos tests, même si aucune des trois fonctions ne sera retenue pour le modèle final (voir discussion de la section 6.4.1). La plupart de ces fonctions existent également en version « contextuelle », c'est-à-dire qu'elles donnent les mêmes valeurs, mais pour le token situé à n positions du mot courant.

Pour obtenir les différentes informations que nous venons de lister, nous recourons aux outils suivants :

- Une liste de mots vides créée par nos soins contenant 790 mots.
- *Stanford Core NLP*⁴ : Nous utilisons ce système de l'Université de Stanford (Manning *et al.* 2014) pour la tokenisation, l'étiquetage des catégories de discours et la reconnaissance d'entités nommées.

4. Voir la page <https://stanfordnlp.github.io/CoreNLP/>.

- *Stanford Unlexicalized Parser*⁵ : pour l'étiquetage syntaxique.
- *Lexique 3* : C'est avec cette base de données lexicales de 135 000 formes fléchies du français (New *et al.* 2001)⁶ que nous avons pu définir une forme et une fréquence de référence pour chaque token. La forme *INC* a été attribuée aux mots inconnus du lexique.
- *Librairie NLTK* : Nous employons plusieurs fonctions de cette librairie disponible en Python (Bird *et al.* 2009), notamment pour la manipulation d'arbres syntaxiques.

Pour les fonctions f_{frf} (forme de référence), f_{pos} (catégorie de discours), f_{ner} (entité nommée) et f_{nop} (nœud père), on a une fonction par valeur possible. Le nombre total de facteurs que représentent ces fonctions dépend de la taille du vocabulaire concerné et, par conséquent, des outils utilisés pour la tokenisation et l'étiquetage syntaxique. Pour les catégories de discours, notre tagueur dispose de 17 étiquettes différentes. Pour les constituants syntaxiques, il y a en 12. On compte également 12 types d'entités nommées. Enfin, pour ce qui est des formes de référence, *Lexique 3* en contient 55 000 (sans compter la forme *INC*). Toutefois, comme on ne peut pas rencontrer toutes les formes des mots dans un corpus de texte, le nombre réel de facteurs f_{frf} vaut le nombre de formes différentes rencontrées dans le corpus d'entraînement (4 077 dans notre cas).

$f_{lon}(m)$	Longueur du mot en caractère.
$f_{lp}(p)$	Longueur de la phrase en token.
$f_{maj}(m)$	Vrai si le mot commence par une majuscule.
$f_{mv}(m)$	Vrai s'il s'agit d'un mot vide.
$f_{pre}(m)$	Vrai s'il s'agit du premier token de la phrase.
$f_{der}(m)$	Vrai s'il s'agit du dernier token de la phrase.
$f_{pr}(m)$	Position relative du mot dans la phrase.
$f_{frf}(m, forme)$	Vrai si la forme de référence du mot est celui mentionné.
$f_{frq}(m)$	Fréquence de référence du mot.
$f_{pos}(m, tag)$	Vrai si le mot appartient la catégorie de discours <i>tag</i> .
$f_{ner}(m, ner)$	Vrai si le mot est une entité nommée du type mentionné.
$f_{nop}(m, tag)$	Vrai si le nœud père du mot dans l'arbre syntaxique est de type <i>tag</i> .
$f_{sub}(m)$	Vrai si le mot se trouve dans une subordonnée.
$f_{pro}(m)$	Profondeur dans l'arbre syntaxique.
$f_{ltn}(m)$	Nombre de nœuds frères du mot (fils sous le nœud père).
$f_{fc}(m)$	Facteur de constituant (cf. supra).

TABLE 6.6 – Facteurs envisagés pour la CAP

5. Voir section 5.1.3 pour plus d'informations.

6. Voir la page <http://www.lexique.org>.

6.4 Conception du modèle

6.4.1 Choix des facteurs

Nous souhaitons à présent relater les expériences qui nous ont amené à l'élaboration de notre compresseur probabiliste, en particulier en ce qui concerne le choix des facteurs. Pour juger de l'efficacité d'un modèle et de ses facteurs, nous utilisons les mesures automatiques de rappel, précision et F_1 , ainsi que le taux de compression (τ).

Pour l'entraînement, nous prenons les 7000 premiers exemples du corpus de compressions maximales (cf. section 6.1.3) et laissons le reste (1432 phrases) pour le test. Le taux de compression moyen du corpus test s'élève à 69,3 %, ce qui correspond à la moyenne du corpus d'entraînement dans son ensemble.

Nous commençons par implémenter un modèle basique. Celui-ci prend en compte un nombre restreint de facteurs et porte sur des propriétés linguistiques assez simples, comme la longueur du mot, de la phrase, la valeur du tag, si le mot commence par une majuscule, s'il commence ou termine la phrase... La liste complète des fonctions de ce premier modèle est la suivante : f_{maj} , f_{lon} , f_{lp} , f_{mv} , f_{pre} , f_{der} , f_{pr} et f_{pos} . En plus de cela, nous regardons aussi la valeur des fonctions f_{maj} et f_{pos} pour les mots suivant et précédent. Ce modèle offre les résultats suivants (6.7) :

	Précision	Rappel	F1
1	0,659	0,904	0,763
0	0,604	0,24	0,343
Moyenne	0,638	0,651	0,603
τ moyen	91,3 %		

TABLE 6.7 – Évaluation automatique du modèle basique sur le jeu de test

À la vue de ceux-ci, le système présente un gros défaut : il ne compresse pas assez. Cela se remarque grâce au taux de compression (particulièrement élevé) et au rappel du label 0. La précision n'est également pas très satisfaisante.

Nous construisons un second modèle engageant des variables syntaxiques plus poussées. En plus des facteurs précédemment envisagés, nous rajoutons la valeur de l'entité nommée, du nœud parent dans l'arbre syntaxique, la profondeur et le nombre de nœuds frères dans l'arbre syntaxique, si le mot est dans une subordonnée et les facteurs de constituants (fonctions f_{ner} , f_{nop} , f_{pro} , f_{sub} , f_{inn} et f_{fc}). Nous regardons la valeur de ces fonctions aux positions voisines. Bien que le nombre de facteurs ait plus que quadruplé, les résultats ne s'améliorent presque pas, avec un taux de compression de 89 %. Cela pourrait s'expliquer par le fait que les annotations syntaxiques sont le signe de dépendances plus longues que celles envisagées.

C'est pourquoi nous reprenons le même modèle, mais avec les valeurs des facteurs aux mots distants de 1 à 3 positions à gauche et à droite par rapport au mot courant. On obtient une légère amélioration pour le rappel du label 0 (26,7 %), mais les résultats ne

sont pas ceux escomptés. Il semblerait que, dans notre corpus, l'information grammaticale soit peu déterminante.

Nous envisageons alors de prendre en compte un nombre de facteurs plus important, mais relevant d'une annotation plus simple. Au modèle basique, nous ajoutons les formes de référence du mot courant et des mots voisins (taille du contexte = 1), suivant la méthode de Li *et al.* (2013) avec les lemmes. On obtient un total de 11 107 facteurs. Les résultats en sont nettement améliorés : le taux de compression descend à 82 % et le rappel du label 0 monte à 0,4. Si l'on prend le même modèle, mais avec un contexte de taille 4 pour tous les facteurs, ces mesures valent respectivement 78 % et 0,47. Pour ce qui est de la précision, on note également une amélioration, avec 0,7 de précision pour le label 1. Nous avons également testé l'influence de la fréquence de référence comme facteur, mais elle s'est avérée nulle lorsqu'elle se combine avec la forme de référence.

Combinons à présent les facteurs de ce nouveau modèle (facteurs du modèle basique + forme de référence + contexte de taille 4) avec les facteurs syntaxiques précédemment testés, pour voir si ce dernier peut tout de même profiter de l'analyse syntaxique. On remarque étonnamment que cette combinaison procure une amélioration quasi inconsciente : seulement 1 % de gagné pour le rappel du label 0 et 0,01% pour le taux de compression. Une courte évaluation manuelle de notre part (sur une cinquantaine de phrases d'évaluation) ne montre aucune amélioration significative des compressions d'un point de vue grammatical, voire, dans certains cas, une détérioration du résultat. Nous décidons dès lors de ne pas inclure les facteurs de type syntaxique (sauf la catégorie de discours) dans notre modèle final.

Le modèle final est donc celui qui a comme facteurs les fonctions f_{lon} (longueur du mot), f_{lp} (longueur de la phrase), f_{mv} (mot vide), f_{pre} (premier token), f_{der} (dernier token), f_{pr} (position relative), f_{maj} (majuscule), f_{pos} (catégorie de discours) et f_{fff} (forme de référence), en regardant la valeur de ces trois dernières fonctions pour les mots voisins distants de 1 à 4 positions à gauche et à droite. Le nombre total de facteurs, après entraînement, s'élève à 28 201. La table 6.8 reprend les résultats de ce modèle. Le modèle est radicalement différent de celui que nous proposons au chapitre précédent. Il demande en effet une annotation très légère des phrases à compresser : seuls un tokeniseur, un lexique et tagueur de catégorie de discours sont nécessaires.

	Précision	Rappel	F1
1	0,712	0,813	0,759
0	0,604	0,465	0,525
Moyenne	0,671	0,68	0,67
τ moyen	78 %		

TABLE 6.8 – Évaluation automatique du modèle final

6.4.2 Apprentissage et exécution

Nous utilisons des paramètres d'apprentissage classiques pour les modèles CRF. L'algorithme d'apprentissage utilise la méthode BGFS comme algorithme du gradient pour trouver le minimum local de la fonction d'optimisation (voir Broyden 1970). La courbe d'ap-

prentissage, qui montre l'évolution des résultats en fonction du nombre d'exemples donnés pour l'entraînement, est représentée en figure 6.3. On voit qu'elle se stabilise à partir de 2000 exemples. Bien que ce résultat soit étonnant, cela peut s'expliquer par le fait que notre compresseur se base sur les formes graphiques et catégories de discours les plus fréquentes dans le langage et dispose ainsi rapidement d'un nombre suffisant d'exemples pour l'entraînement.

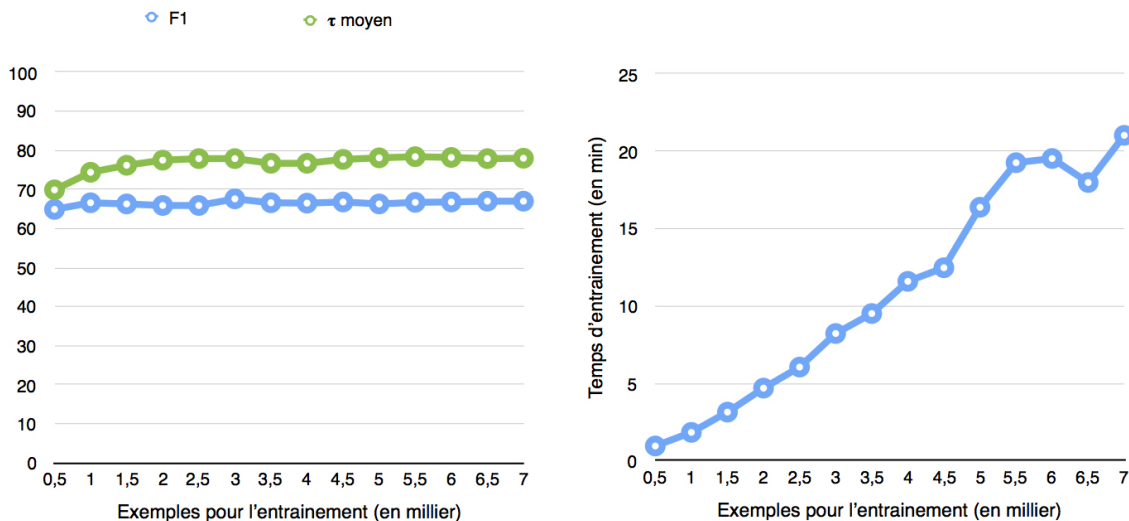


FIGURE 6.3 – Courbe et temps d'apprentissage en fonction du nombre de données

Comme le montre la courbe à la droite de la figure 6.3, le temps d'apprentissage croît significativement en fonction du nombre de données d'entraînement. Toutefois, le processus reste relativement rapide : une vingtaine de minutes suffisent pour entraîner le modèle avec 7000 exemples. Concernant le temps d'exécution, le compresseur s'avère très performant. Une phrase non annotée prend, en moyenne, 3 secondes pour être compressée. Selon nos tests, ce temps d'exécution ne dépend pas de la longueur de la phrase, un avantage certain pour la tâche de CAP. L'annotation syntaxique (tokenisation et POS tagging) prend la majeure partie du temps d'exécution, la compression en elle-même se réalisant très rapidement.

Nous donnons accès aux scripts d'entraînement et d'exécution en annexe. Le programme a été conçu en Python (version 3.4) et fait appel à la librairie *crfsuite* de *sklearn*⁷ pour l'implémentation du CRF. Les autres outils de développement auxquels nous avons eu recours ont été mentionnés dans la section précédente.

6.5 Évaluation

6.5.1 Évaluation automatique

Il convient maintenant d'évaluer notre compresseur de façon intrinsèque. Le corpus de test sont les 1432 dernières phrases du corpus décrit en début de chapitre, soit le même que celui utilisé au point 6.4.1. Ce corpus test présente des caractéristiques semblables

7. Documentation disponible à l'adresse <https://sklearn-crfsuite.readthedocs.io/en/latest/index.html>.

au corpus de façon générale (taux de phrases non compressées et taux de compression moyen similaires).

Nous avons déjà relaté les résultats du système de façon automatique, en mentionnant les mesures de rappel, précision, F_1 et τ . Comparons ces résultats avec la littérature récente en CAP. Le meilleur modèle de Nomoto (2007) montre un taux de compression moyen plus bas que le nôtre (70 %). Cela s'explique par le fait que son corpus d'entraînement possède un taux de compression humain bien plus bas que le nôtre : dans son cas, les phrases d'entraînement ont, en moyenne, été délestées de la moitié de leurs mots (Nomoto 2007 : 1584). Sakti *et al.* (2015) ont travaillé avec un corpus d'entraînement compressé en moyenne à 72 %, un τ d'entraînement très proche du nôtre. On remarque que le τ obtenu dans notre évaluation s'aligne avec celui du meilleur modèle de Sakti *et al.* (2015), qui, chez eux, est de 80 %. Les autres travaux ne renseignent pas le taux de compression ou le calculent d'une façon non comparable avec la nôtre. Concernant la mesure F_1 , nous obtenons un meilleur score que Nomoto (2007 : 1584) (67 % contre 63 %). À nouveau, nous égalisons les scores de Sakti *et al.* (2015 : 256) sur le plan de la précision (le rappel et F_1 ne sont pas reportés dans leur article). Les seuls travaux ayant produit des résultats supérieurs sont ceux de Filippova *et al.* (2015) et Klerke *et al.* (2016), avec des F_1 avoisinant les 0,8.

Il faut noter que tous les systèmes avec lesquels nous comparons le nôtre traitent l'anglais. Nous n'avons pas connaissance de l'existence d'un compresseur récent pour le français, comme le montre notre état de l'art du domaine de la CAP (cf. section 3.4.3). Nous discutons l'influence de ce constat sur nos résultats en section 6.5.3.

Intéressons-nous maintenant plus en détail à l'étendue des compressions. On remarque tout d'abord que de nombreuses phrases n'ont pas été compressées : on en recense 557 (38 %), là où le corpus test n'en contient que 171 (11 %). De façon générale, notre compresseur privilégie les compressions courtes, comme le montre la figure 6.4⁸. Si la courbe que dessine cet histogramme est descendante, elle reste néanmoins assez linéaire.

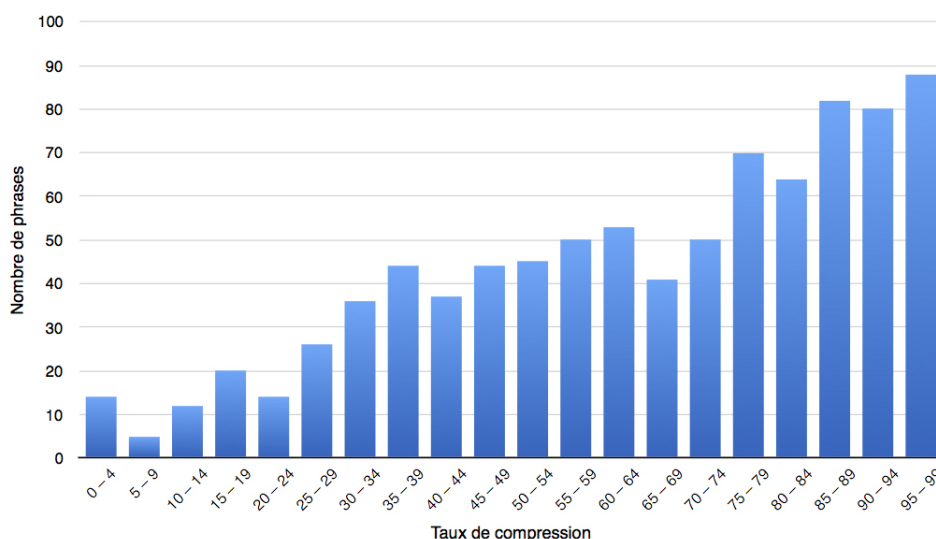


FIGURE 6.4 – Nombre de phrases en fonction du taux de compression

8. La figure ne concerne que les phrases dont un mot au moins a été supprimé.

6.5.2 Évaluation manuelle

Pour mieux juger de la qualité du compresseur sur les plans de la grammaticalité et de la cohérence, nous menons une évaluation automatique du même type que celle effectuée dans le chapitre précédent (voir la section 5.2 pour une explication détaillée de la méthodologie). La seule différence avec celle-ci est que les annotateurs n'ont pas accès au texte dans son ensemble pour juger de la cohérence, mais uniquement à la phrase originale. Nous avons recruté deux annotateurs non experts pour juger la grammaticalité et la cohérence d'un panel de 200 phrases issues du corpus test et choisies aléatoirement.

Les résultats de cette évaluation manuelle sont repris dans la table 6.9. En premier lieu, on remarque qu'il y a presque autant de phrases agrammaticales et incohérentes que de phrases entièrement correctes : en moyenne près de 50 % des phrases ont été jugées grammaticalement justes, contre 40 % inacceptables. Les 10 % restant concernent les cas mitigés. Le rapport est presque le même pour la cohérence textuelle. En réalité, la grammaticalité a beaucoup influencé la cohérence, certains énoncés agrammaticaux rendant difficile (voire impossible) l'accès au sens. En effet, lorsque les annotateurs ont jugé la grammaticalité mauvaise, pour une grande majorité des cas, ils ont fait de même pour la cohérence.

<i>Annotateur 1</i>	Grammaticalité			Sémantique/Cohérence		
Appréciation	Très bien	OK	Mauvais	Très bien	OK	Mauvais
Fréquence brute	83	32	85	83	21	96
Fréquence relative	41,5 %	16 %	42,5 %	41,5 %	10,5 %	48 %
<i>Annotateur 2</i>	Grammaticalité			Sémantique/Cohérence		
Appréciation	Très bien	OK	Mauvais	Très bien	OK	Mauvais
Fréquence brute	106	18	76	98	29	73
Fréquence relative	53 %	9 %	38 %	49 %	14,5 %	36,5 %
<i>Moyenne</i>	Grammaticalité			Sémantique/Cohérence		
Appréciation	Très bien	OK	Mauvais	Très bien	OK	Mauvais
Fréquence brute	94,5	25	80,5	89,5	25	84,5
Fréquence relative	47,25 %	12,5 %	40,25 %	44,75 %	12,5 %	42,25 %

TABLE 6.9 – Résultats de l'évaluation manuelle

On note d'autre part une variation non négligeable entre les annotateurs. De toute évidence, le second annotateur s'est montré plus clément que le premier, diminuant la part d'erreur à un tiers des phrases, lorsque celle du premier est plus proche de la moitié. L'accord inter-annotateur (table 6.10), calculé via le κ de Cohen⁹ (Artstein & Poesio 2008 : 559-561) pour deux variables ordinales, montre d'ailleurs des taux de corrélation modérés, tournant autour de 50 %. Pour annoter la cohérence, en particulier, les annotateurs divergent beaucoup : le taux de corrélation descend à 44 %. En termes d'occurrences brutes, seule la moitié des phrases (108 sur 200) ont obtenu un jugement identique concernant la grammaticalité.

9. Mesure classiquement utilisée en TAL pour l'accord inter-annotateur, voir l'article de Artstein & Poesio (2008).

Grammaticalité	0,55
Cohérence	0,44
Moyenne	0,495

TABLE 6.10 – Accord inter-annotateur : κ de Cohen

Les deux juges ont également eu l'occasion d'émettre des commentaires sur certaines compressions ou un avis général sur les phrases. Ces remarques nous ont aidé à mieux cerner les différents types d'erreurs de notre compresseur. Dans l'ensemble, notre compresseur s'avère fiable pour l'élimination de segments courts (adverbes, compléments circonstanciels, courtes incises...), mais commet parfois de graves erreurs concernant les longues compressions. Divers exemples de compressions sont exposés à la figure 6.5.

<i>Phrase originale</i> : Pour autant, le malaise ne se limite pas à un conflit de personne. <i>Compression</i> : Le malaise ne se limite pas à un conflit de personne.
<i>Phrase originale</i> : Jeudi, comme la veille, les dirigeants ont discuté toute la journée pour tenter de sortir de la paralysie. <i>Compression</i> : Les dirigeants ont discuté toute la journée pour tenter de sortir de la paralysie.
<i>Phrase originale</i> : Cambodge : mise en garde contre les risques d'un « conflit armé » avec la Thaïlande. <i>Compression</i> : Cambodge : mise en garde contre les risques d'un "conflit armé.
<i>Phrase originale</i> : Si la fête a commencé par une ouverture au monde avec le message de bienvenue de l'incontournable Confucius (« N'est-ce pas un bonheur d'avoir des amis qui viennent de loin ? »), elle est revenue rapidement à la mère-patrie. <i>Compression</i> : Si la fête a commencé par une ouverture au monde avec le message de bienvenue de l'incontournable Confucius ("N'est-ce pas un bonheur d'avoir des amis) elle est revenue.
<i>Phrase originale</i> : « J'ai une vie privée compliquée et c'est la limite que je me pose vis-à-vis de la presse. » <i>Compression</i> : « J'ai une vie privée compliquée »
<i>Phrase originale</i> : Il devra s'inscrire dans le cadre des initiatives coordonnées au niveau de l'Union européenne qui seront examinées par les ministres des Finances des 27 à Bruxelles mardi. <i>Compression</i> : Il devra s'inscrire dans le cadre des initiatives coordonnées au niveau de l'Union.

FIGURE 6.5 – Exemples de compressions réalisées par notre système

Pour la grammaticalité, nous rencontrons tout d'abord des problèmes similaires à ceux évoqués par Nomoto (2007 : 1580). Certaines règles syntaxiques basiques sont en effet enfreintes, laissant des déterminants sans leur nom ou des verbes sans leur sujet (voir les deux premiers exemples de la figure 6.6). Le second problème concerne le traitement des noms propres : lorsqu'il y en a plusieurs, le compresseur paraît plus sujet à la faute (voir les exemples 3 et 4 de la figure 6.6).

Concernant la sémantique, il y a peu de fautes de cohérence pure, la grammaticalité étant la source principale d'incohérence, comme nous l'avons déjà dit. Nous donnons deux exemples de compressions grammaticales, mais incohérentes ou ne reprenant pas l'infor-

1) <i>Phrase originale</i> : Le groupe Société Nationale Immobilière (SNI), filiale immobilière du logement social de la CDC, avait ainsi immédiatement indiqué son intention d'acquérir 10.000 logements, soit en direct, soit via les organismes HLM qu'il contrôle. <i>Compression</i> : Avait indiqué son intention d'acquérir 10.000 logements.
2) <i>Phrase originale</i> : Ensuite elle envisage de s'installer à New York, où elle pourrait occuper un poste d'ambassadrice pour la paix aux Nations unies et aider son pays dans la lutte contre la corruption et les narcotrafics. <i>Compression</i> : Elle envisage de s'.
3) <i>Phrase originale</i> : On rappelle aussi, bien sûr, la maternité de Ségolène Royal, alors secrétaire d'État, ou celle de l'actuelle ministre de la Défense espagnole, Carmen Chacon. <i>Compression</i> : On rappelle aussi, la maternité de Ségolène Royal, Chacon.
4) <i>Phrase originale</i> : Cette longue parenthèse dans ce paradis de l'océan Indien, agrémentée d'un séjour dans l'île Desroches, est le signe de la grande fatigue physique et nerveuse dans laquelle se trouve Ingrid Betancourt. <i>Compression</i> : Cette longue parenthèse dans ce paradis de l'océan Indien, agrémentée d'un séjour dans l'île Desroches, est le signe de la Betancourt.

FIGURE 6.6 – Exemples d'erreur de grammaticalité

mation originale de façon satisfaisante à la figure 6.7. Ces compressions changent le sens initial de la phrase, en supprimant des segments très informatifs. Ces cas, peu fréquents, sont difficiles à traiter, car ils dénotent des dépendances longues et complexes, engageant des facteurs à la fois sémantiques et syntaxiques. Par exemple, pour la première phrase de la figure 6.7, il aurait fallu que le modèle prenne en compte le fait que *augmenté* et *redescendre* sont, dans ce contexte, de sens contraire et se rapportent au même sujet, pour que la probabilité de ne pas supprimer le segment final augmente.

1) <i>Phrase originale</i> : Les taux ont en moyenne augmenté de 0,35 % entre la mi-juin et la mi-août, avant de redescendre de 0,20 % ces dernières semaines. <i>Compression</i> : Les taux ont en moyenne augmenté de 0,35 % entre la mi-juin et la mi-août.
2) <i>Phrase originale</i> : Les trois patrons « s'interrogent » sur la validité juridique d'une taxe destinée à « financer un concurrent direct ». <i>Compression</i> : Les trois patrons « s'interrogent ».

FIGURE 6.7 – Exemples d'erreur de cohérence

6.5.3 Analyse des erreurs

Au niveau de la grammaticalité, le comportement de notre compresseur s'explique, en grande partie, par les spécificités du corpus d'entraînement. Si le modèle traite mieux et plus souvent des segments courts, cela est dû à leur supériorité numérique dans le corpus. Comme nous l'évoquions à la description du corpus (voir la table 6.2), le taux de compression moyen est élevé et la plupart des compressions éliminent 5 mots ou moins. On remarque d'ailleurs que la courbe montrant le nombre de compressions réalisées pour chaque taux de compression (représentée à la figure 6.4) est la même que celle du corpus d'entraînement (voir De Loupy *et al.* 2010 : 3116). Cela explique la préférence de notre modèle pour la compression de segments courts, puisque les annotateurs eux-mêmes ont favorisé ce type de compression. On comprend également pourquoi, lorsqu'une phrase contient beaucoup de segments potentiellement suppressibles, elle est moins bien an-

notée : ces cas étaient moins bien représentés dans le corpus que les compressions peu étendues.

On peut aussi invoquer les particularités de la langue que nous traitons. La phrase française a généralement besoin de plus de mots pour garantir la grammaticalité que la syntaxe anglaise et ne supportera pas souvent un taux de compression aussi bas. Ceci pourrait justifier que notre corpus d'entraînement contienne plus de compressions longues que la normale. Cela peut aussi expliquer les meilleures performances au niveau grammatical de notre compresseur pour le traitement des segments supprimables courts et réguliers.

Les erreurs se rapportant aux noms propres s'expliquent, quant à elles, par le fait que le modèle se base en grande partie sur les formes de référence des tokens. Les variables observées sont identifiées selon un lexique prédéfini, ne comprenant pas de noms propres. Ces derniers sont alors traités par le modèle comme des mots inconnus, ce qui explique la fréquence des erreurs liées aux noms propres.

On peut développer plusieurs types de solutions à ces problèmes de grammaticalité. Le premier serait de travailler avec un corpus contenant encore plus d'exemples, avec une grande proportion de cas difficiles à compresser. Les CRF pourraient, dans ce cas, être remplacés par des modèles de calculs jugés plus performants, comme les réseaux de neurones. C'est ce qu'ont expérimenté Filippova *et al.* (2015) pour l'anglais, avec des résultats prometteurs. Les réseaux de neurones pourraient s'avérer plus appropriés pour la tâche de la compression en raison de la grande diversité des facteurs à prendre en compte pour des résultats optimaux.

Le second type de solution est de développer un ensemble de petits modèles spécialisés dans le traitement de cas particuliers, plutôt que d'en construire un seul. Nomoto (2007) a été le premier à proposer une solution de ce style. Dans son système, le chercheur entraîne une série de modèles secondaires entraînés pour traiter des types de propositions et syntagmes particuliers. Par exemple, au moment de compresser une phrase contenant une subordonnée, le système, pour choisir quels mots conserver au sein de cette proposition subordonnée, fait appel à un modèle CRF secondaire entraîné exclusivement sur base d'exemples de propositions subordonnées. La technique augmente significativement les résultats sur le plan grammatical (Nomoto 2007 : 1583-1584). Elle exige néanmoins une variété importante dans les données fournies pour l'entraînement.

Nous voyons également une troisième voie d'amélioration, moins demandeuse en données d'entraînement : une approche mixte entre règles de compression manuelles et automatiques. Cette solution est discutée plus en détail au moment de clôturer la seconde partie du travail.

Finalement, certaines erreurs de cohérence se produisent parce qu'elles dépendent de facteurs que nous n'avons pas modélisés et qu'il serait difficile de modéliser. Pour notre compresseur, nous pensons que cela ne serait pas révélé avantageux d'inclure des facteurs sémantiques, parce qu'ils requièrent une annotation lourde. Une partie de l'intérêt de notre approche repose sur la simplicité de l'annotation des variables observées, ce qu'une annotation sémantique gâcherait. En revanche, on pourrait peut-être améliorer la cohérence des compressions plus facilement si l'on avait accès au texte original dans son entièreté (comme c'est le cas dans le travail de Li *et al.* 2013). Des facteurs comme la position de la phrase ou le nombre de mots en commun avec le titre pourraient aider le compresseur à limiter les erreurs de cohérence textuelle, sans sacrifier la rapidité d'exécution.

Conclusion

Nous avons terminé de présenter notre second compresseur automatique de phrases, développé selon une approche statistique.

Nous avons fait preuve d'originalité en présentant un programme pour le français. Nous avons relevé le défi de trouver des données annotées en français, en quantité suffisante pour une approche probabiliste. Notre corpus d'entraînement est le plus grand disponible pour la CAP en français, et parmi les plus larges si on compare avec les corpus anglophones. Il n'avait jusqu'ici jamais été utilisé à des fins d'entraînement supervisé.

L'implémentation du modèle a fait l'objet de diverses expérimentations, exploitant les techniques décrites dans les rares travaux reposant sur le formalisme des champs conditionnels aléatoires en CAP. Celles-ci nous ont toutefois conduit à un modèle assez différent de ceux proposés dans la littérature : notre modèle connaît un nombre élevé de facteurs, mais ne requiert qu'une annotation simple des données observées. Notre approche remet ainsi en cause les recherches précédentes en CAP en ce qui concerne l'impact des informations syntaxiques dans les CRF.

Par rapport au compresseur précédent, nous avons également accordé plus de temps et de moyens à l'évaluation. On a tout d'abord pu voir que l'approche statistique offre un gain de temps d'exécution et de conception considérable par rapport à son équivalente symbolique, moyennant un corpus annoté pour l'entraînement. L'évaluation automatique montre des résultats équivalents à plusieurs travaux récents. L'évaluation manuelle a souligné un grand nombre d'erreurs au niveau de la grammaticalité, bien que, en moyenne, 60 % des phrases du corpus test ont été jugées au minimum acceptables.

Ces résultats, moins brillants que ceux du chapitre précédent, ne prouvent pas pour autant la supériorité de l'approche manuelle. L'approche statistique, en s'appuyant sur des données réelles, gère très bien les cas fréquents. Si elle compresse moins bien les structures irrégulières, c'est parce qu'il est difficile d'obtenir un nombre d'exemples suffisant pour l'entraînement. Cette difficulté apparaît liée à la tâche de compression, qui admet une part importante de liberté. Comme on peut compresser de plusieurs façons une même phrase, et comme on ne pourra jamais disposer d'un corpus reprenant un ensemble assez représentatif de toutes les phrases d'une langue, il est intrinsèquement complexe d'obtenir un modèle statistique performant pour tout type de cas. Néanmoins, elle permet d'identifier et de traiter un nombre plus varié de cas différents que l'approche par règles. Notre approche probabiliste ne s'érige donc pas en solution finale au problème de la CAP, mais présente tout de même des avantages importants par rapport à l'approche manuelle.

Conclusion de la partie II

Dans cette seconde partie, nous avons détaillé l'implémentation de deux compresseurs de phrases suivant des approches distinctes. Le premier suivait une approche symbolique et reposait sur un jeu de règles conçues manuellement. Le programme compressait des phrases rédigées en anglais et était spécialisé dans le traitement de structures syntaxiques complexes. Le second système a été implémenté par apprentissage automatique supervisé : sur base d'un corpus aligné de phrases originales et compressées, nous avons entraîné un modèle statistique permettant de prédire la probabilité qu'un mot soit conservé dans une phrase donnée. Le formalisme avec lequel nous avons travaillé sont les champs conditionnels aléatoires. Ce second système proposait une solution adaptée au français, une langue peu traitée en compression automatique de phrases.

Chacune des deux méthodologies présentées a sa part d'originalité. Pour la première, nous nous sommes concentré sur un type précis de compression, alors que les chercheurs en CAP tentent généralement de trouver des solutions globales. Pour le type de structures que nous avons choisi de traiter, on a vu qu'une approche manuelle était tout à fait valide, en particulier sur le plan de la grammaticalité. Cela contraste avec la tendance actuelle en TAL de trouver des solutions entièrement automatiques, certes moins coûteuses en temps de conception et d'exécution, mais pas forcément optimales ni généralisables.

Toutefois, nous sommes conscient de la nécessité d'une solution globale au problème de la CAP, et avons voulu, pour cela, dans la seconde approche, implémenter un système capable de compresser des phrases variées. Nous nous sommes attaché à produire un modèle pour le français, afin de voir si les méthodes anglophones des études récentes pouvaient être répliquées. Bien que notre modèle se base sur une annotation syntaxique légère – une méthode qui s'éloigne des approches développées en CAP –, nos résultats concurrencent plusieurs travaux récents en CAP. Le compresseur reste, malgré tout, améliorable, en particulier en ce qui concerne la grammaticalité, le défi principal des approches statistiques.

Passer du paradigme symbolique au paradigme statistique nous permet de mieux cerner les avantages, tant théoriques que pratiques, de l'un par rapport à l'autre. L'approche manuelle a demandé plus de temps de conception et requiert une lourde annotation des données à traiter, mais elle a montré de meilleurs résultats en matière de grammaticalité et de cohérence. Néanmoins, nous n'avons pas pu connaître la part de structures non traitées par le système. Le compresseur probabiliste, quant à lui, agit bien plus rapidement et prend moins de temps à être construit. Sur le plan théorique, l'approche statistique offre également la possibilité de traiter tout type de phrase, peu importe sa structure syntaxique et son contenu. S'il nous a semblé que le compresseur probabiliste identifiait plus de segments à compresser, on a aussi vu qu'il avait plus tendance à se tromper, surtout lorsque ces segments sont longs ou contiennent des irrégularités.

Nos deux expériences démontrent que, d'un côté, l'approche manuelle offre des compressions de meilleure qualité, et ce même pour les structures syntaxiques complexes, et que, d'un autre côté, l'approche statistique permet d'envisager un plus grand nombre de structures différentes et compresse une majorité d'entre eux de façon acceptable. Les atouts de chacune des approches nous ont poussé à imaginer une solution nouvelle pour produire un

compresseur robuste tant sur le plan qualitatif que quantitatif. Il s'agit d'une approche que l'on pourrait qualifier de mixte. L'idée serait de créer un système reposant sur deux jeux de règles différents, l'un conçu par apprentissage automatique, l'autre manuelle. Le modèle probabiliste s'occuperait des compressions les moins risquées grammaticalement (c'est-à-dire les plus courtes ou celles apparaissant souvent dans le corpus d'entraînement), tandis qu'un compresseur manuel se chargerait des structures de phrases complexes. Si cette approche demande sans doute plus de temps de développement et requiert une annotation plus conséquente que l'approche statistique, elle devrait, en contrepartie, garantir de meilleures performances que celles présentées jusqu'à présent.

Chapitre 7

Conclusion générale

7.1 Résultats

Tout au long de cette étude théorique et expérimentale sur la compression de phrases pour le résumé automatique, nous avons répondu à plusieurs questions fondamentales.

Premièrement, pourquoi la compression de phrases est-elle intéressante pour le domaine du résumé automatique de texte ? Dans un contexte où le RAT n'envisage pas de paraphraser le contenu des textes, toute technique de dissection de l'information par suppression de segments a son rôle à jouer. Plus particulièrement, une fois qu'on dispose d'un moyen de sélectionner les phrases les plus importantes d'un texte, la compression règle certains problèmes posés par l'extraction de phrases, tels que la sélection préférentielle des phrases longues et la présence d'unités intra-phrases non pertinentes. Elle offre ainsi la possibilité d'inclure plus d'informations dans le résumé et de les présenter de façon plus concise et directe. Une solution d'autant plus élégante qu'il a été démontré qu'elle correspond à des pratiques humaines de résumé.

Deuxièmement, comment combiner les nouveaux algorithmes de compression de phrases avec les systèmes d'extraction de phrases déjà implémentés ? On a vu dans notre état de l'art des recherches en compression appliquée au résumé qu'on peut se servir d'un compresseur avant, après ou en même temps que l'opération d'extraction de phrases. Il n'y a pas, dans la littérature, d'argument empirique supposant la supériorité de l'une ou l'autre méthode. L'approche conjointe présente tout de même plus de nouveauté, en redéfinissant un nouvel algorithme de résumé qui combine, en une seule étape, l'extraction et la compression de phrases.

Troisièmement, à quel type de résultats peut-on s'attendre selon le genre d'approche suivie en CAP ? Nous avons montré qu'il était plus facile de maintenir la grammaticalité des phrases en appliquant un jeu de règles de compression créé manuellement qu'en utilisant une méthode probabiliste. L'approche manuelle permet aussi un meilleur traitement des phrases complexes, si le système dispose d'assez d'informations syntaxiques. Néanmoins, il semble impossible d'écrire un ensemble de règles prenant en compte tous les cas possibles de compression, surtout si l'on tient compte de l'importance du contexte dans la tâche de compression. L'approche statistique rend possible la construction d'un jeu de règles étendu et varié. Avec une annotation assez simple des phrases, sans recourir à l'étiquetage syntaxique (ou *parsing*), nous avons montré qu'il était possible de construire un compresseur efficace pour les opérations de suppression les plus simples. Par ailleurs, on a vu que les résultats de l'approche statistique dépendaient fortement du corpus d'entraînement. Celui-ci doit contenir un nombre élevé d'exemples les plus variés possible pour garantir un résultat satisfaisant.

Quatrièmement, pour l'approche statistique, peut-on reproduire les méthodes anglo-saxonnes pour implémenter un compresseur pour le français, et à quel point leurs résultats sont-ils comparables ? Même si le français dispose de moins de données annotées pour la compression, nous avons démontré qu'il était possible de développer un compresseur dont les résultats équivalent plusieurs travaux récents en CAP. Nous avons, à ce sujet, émis des réserves sur le caractère comparable des systèmes : l'anglais semble supporter un taux de compression moins élevé que le français. Cette remarque s'ajoute à celles soulignées par Napoles *et al.* (2011), qui mettaient déjà en garde contre la comparaison de compresseurs aux taux de compression distincts et évalués sur des corpus non similaires.

7.2 Perspectives

Répondre à ces quatre questions a conduit à en poser une cinquième : comment surmonter les difficultés rencontrées dans nos expériences en CAP et améliorer les résultats produits ? En comparant les résultats de nos deux compresseurs, nous avons remarqué qu'ils étaient, d'une certaine façon, complémentaires : l'un compresse bien des phrases contenant de longues structures enchâssées, tandis que l'autre a un traitement rapide et efficace des segments courts. Une approche mixte, à la fois symbolique et statistique, s'envisage dès lors naturellement : un compresseur entraîné sur corpus se chargerait des cas les plus fréquents et, lorsqu'il rencontrerait un long segment à supprimer, ferait appel à un compresseur symbolique. Si nous n'avons pas pu développer de système implémentant cette solution parce que nos compresseurs ne traitaient pas la même langue, nous ne doutons pas qu'elle puisse surpasser les performances des compresseurs actuels, et ce, avec un besoin modéré en données annotées.

Dans nos méthodes, nous avons, par ailleurs, fait peu de cas de la conservation de la cohérence. Nous avons cependant repéré quelques erreurs de sémantique dans nos systèmes, bien qu'en des proportions restreintes. La voie d'amélioration la plus prometteuse est, selon nous, l'inclusion de facteurs relatifs aux phrases voisines et au texte dans son ensemble. Mais cela nécessite des corpus de documents compressés, un type de ressources encore plus rare qu'un corpus de phrases compressées, d'ailleurs inexistant pour le français. Les recherches futures devront néanmoins se pencher sur la question de l'inclusion de la sémantique dans la compression de phrases, qui est loin d'être triviale, comme le signale Molina Villegas (2013 : 107-109).

Si nous n'avons pas présenté de méthode parfaite pour la compression automatique de phrases, nous pensons toutefois avoir contribué à ce domaine en identifiant précisément les besoins du résumé automatique en matière de compression et en imaginant de nouvelles voies d'amélioration pour les systèmes à venir. Ces conclusions nous rapprochent un peu plus d'un résumé plus intelligent, d'un résumé plus « abstraktif », comme l'annonçaient Knight & Marcu (2002) au moment de présenter leur compresseur statistique ; bref, d'un résumé utile pour toute personne en quête de connaissance.

Bibliographie

- APOTHÉLOZ D. 1995 *Rôle et fonctionnement de l'anaphore dans la dynamique textuelle*, Librairie Droz S.A., Genève. 31
- ARTSTEIN R. & POESIO M. 2008 « Inter-coder agreement for computational linguistics » dans *Computational Linguistics*, vol. 34(4), pp. 555–596. 81
- BANGALORE S., RAMBOW O. & WHITTAKER S. 2000 « Evaluation metrics for generation » dans *Proceedings of the first international conference on Natural language generation*, vol. 14, pp. 1–8. 41
- BARZILAY R. & ELHADAD M. 1997 « Using lexical chains for text summarization » dans *Advances in automatic text summarization*, pp. 111–121. 26
- BAXENDALE P.B. 1958 « Machine-made index for technical literature — an experiment » dans *BM Journal of Research and Development*, vol. 2(4), pp. 354–361. 20
- BIRD S., KLEIN E. & LOPER E. 2009 *Natural language processing with Python : analyzing text with the natural language toolkit*, O'Reilly Media, Inc. 76
- BOUDIN F. 2008 *Exploration d'approches statistiques pour le résumé automatique de texte*, Thèse de doctorat Université d'Avignon et des Pays de Vaucluse. 22, 23
- BROYDEN C.G. 1970 « The convergence of a class of double-rank minimization algorithms 1. general considerations » dans *IMA Journal of Applied Mathematics*, vol. 6(1), pp. 76–90. 78
- BRUNN M., CHALI Y. & PINCHAK C.J. 2001 « Text summarization using lexical chains » dans *Proceedings of DUC*. 26
- BURGES C., SHAKED T., RENSHAW E., LAZIER A., DEEDS M., HAMILTON N. & HULLENDER G. 2005 « Learning to rank using gradient descent » dans *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96. 25
- CLARKE J. & LAPATA M. 2006 « Constraint-based sentence compression an integer programming approach » dans *Proceedings of the COLING/ACL on Main conference poster sessions*. 36, 41
- CLARKE J. & LAPATA M. 2009 « Models for sentence compression : A comparison across domains, training requirements and evaluation measures » dans *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. 36, 42
- CONROY J.M. & O'LEARY D.P. 2001 « Text summarization via hidden markov models » dans *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. 22
- CONROY J.M., SCHLESINGER J.D., O'LEARY D.P. & GOLDSTEIN J. 2006 « Back to basics : Classy 2006 » dans *Proceedings of DUC*. 37, 38, 42

- COPECK T. & SZPAKOWICZ S. 2004 « Vocabulary agreement among model summaries and source documents » dans *ACL Text Summarization Workshop*. 18, 46
- CORSTON-OLIVER S.H. & DOLAN W.B. 1999 « Less is more : eliminating index terms from subordinate clauses » dans *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pp. 349–356. 37
- DAELEMANS W., HÖTHKER A. & SANG E.F.T.K. 2004 « Automatic sentence simplification for subtitling in dutch and english » dans *LREC*. 37
- DE LOUPY C., GUÉGAN M., AYACHE C., SENG S. & TORRES-MORENO J.M. 2010 « A french human reference corpus for multi-document summarization and sentence compression. » dans *LREC*. 69, 70, 71, 83, 102
- DEERWESTER S., DUMAIS S.T., FURNAS G.W., LANDAUER T.K. & HARSHMAN R. 1990 « Indexing by latent semantic analysis » dans *Journal of the American society for information science*, vol. 41(6). 23
- DEJONG G. 1982 « An overview of the frump system » dans *Strategies for natural language processing*, vol. 113(149-176). 17
- DONAWAY R.L., DRUMMEY K.W. & MATHER L.A. 2000 « A comparison of rankings produced by summarization evaluation measures » dans *Proceedings of the 2000 NAACL-ANLP Workshop on Automatic summarization*. 29, 30
- DORR B., MONZ C., SCHWARTZ R. & ZAJIC D. 2005 « A methodology for extrinsic evaluation of text summarization : does rouge correlate ? » dans *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. 30
- DURRET G., BERG-PATRICK T. & KLEIN D. 2016 « Learning-based single-document summarization with compression and anaphoricity constraints » dans *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. 40, 42, 48, 63
- EDMUNDS A. & MORRIS A. 2000 « The problem of information overload in business organisations : a review of the literature » dans *International journal of information management*, vol. 20(1), pp. 17–28. 5
- EDMUNDSON H.P. 1969 « New methods in automatic extracting » dans *Journal of the ACM (JACM)*, vol. 16(2), pp. 264–285. 6, 21
- ERKAN G. & RADEV D.R. 2004 « Lexrank : Graph-based lexical centrality as salience in text summarization » dans *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–479. 24, 99
- EVANS R. & ORASAN C. 2013 « Sentence simplification without a parser », paru non publié, Research Group for Computational Linguistics, University of Wolverhampton. 58, 59, 61, 101
- FARZINDAR A., ROZON F. & LAPALME G. 2005 « Cats a topic-oriented multi-document summarization system at duc 2005 » dans *Proceedings of the 2005 Document Understanding Workshop*. 21
- FILIPPOVA K., ALFONSECA E., COLMENARES C.A., KAISER L. & VINYALS O. 2015 « Sentence compression by deletion with lstms » dans *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 360–368. 40, 41, 69, 80, 84

- FREEMAN L.C. 1977 « A set of measures of centrality based on betweenness » dans *Sociometry*, vol. 40(1), pp. 35–41. 24
- GAGNON M. & DA SILVA L. 2005 « Text summarization by sentence extraction and syntactic pruning » dans *Papyrus (Université de Montréal)*. 32, 47
- GAIZAUSKAS R., FOSTER J., WILKS Y., ARUNDEL J., CLOUGH P. & PIAO S. 2001 « The meter corpus : a corpus for analysing journalistic text reuse » dans *Proceedings of the corpus linguistics 2001 conference*, pp. 214–223. 61
- GALLEY M. & MCKEOWN K. 2007 « Lexicalized markov grammars for sentence compression » dans *Human Language Technologies 2007 : The Conference of the North American Chapter of the Association for Computational Linguistics : Proceedings of the Main Conference*, pp. 180–187. 39, 41
- GENEST P.E. & LAPALME G. 2012 « Fully abstractive approach to guided summarization » dans *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics : Short Papers*, vol. 2, pp. 354–358. 17
- GENEST P.E., LAPALME G. & YOUSFI-MONOD M. 2010 « Jusqu'où peut-on aller avec des méthodes par extraction pour la rédaction de résumés ? » dans *Actes de TALN, 2010*. 33
- GOLDSTEIN J. & CARBONELL J. 1998 « The use of mmr, diversity based reranking for reordering documents and producing summaries » dans *ACM Special Interest Group on Information Retrieval*, pp. 335–336. 24
- GOLDSTEIN J., MITTAL V., CARBONELL J. & KANTROWITZ M. 2000 « Multi-document summarization by sentence extraction » dans *Proceedings of the 2000 NAACL-ANLP Workshop on Automatic summarization*, vol. 4, pp. 40–48. 24
- GONG Y. & LIU X. 2001 « Generic text summarization using relevance measure and latent semantic analysis » dans *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 19–25. 24
- GREFENSTETTE G. 1998 « Producing intelligent telegraphic text reduction to provide an audio scanning service for the blind » dans *Working notes of the AAAI Spring Symposium on Intelligent Text summarization*, pp. 111–118. 36, 37
- HACHEY B. & GROVER C. 2004 « A rhetorical status classifier for legal text summarisation » dans *Proceedings of the ACL-2004 Text Summarization Branches Out Workshop*. 27
- HALLIDAY M.A.K. & HASAN R. 1976 *Cohesion in english*, Longman Group. 26, 31
- HIRAO T., SASAKI Y. & ISOZAKI H. 2001 « An extrinsic evaluation for question-biased text summarization on qa tasks » dans *Proceedings of the NAACL 2001 Workshop on Automatic Summarization*. 31
- HOVY E., LIN C.Y., ZHOU L. & FUKUMOTO J. 2006 « Automated summarization evaluation with basic elements » dans *Proceedings of the fifth Conference on Language Resources and Evaluation (LREC 2006)*. 29, 30
- JING H. 2002 « Using hidden markov modeling to decompose human-written summaries » dans *Computational linguistics*, vol. 28(4), pp. 527–543. 18

- JING H. & MCKEOWN K.R. 1999 « The decomposition of human-written summary sentences » dans *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 129–136. 46
- JURAFSKY D. & MARTIN J.H. 2009 *Speech and language processing*, Pearson International Edition 2nd edn.. 39
- KAIKHAH K. 2004 « Automatic text summarization with neural networks » dans *Intelligent Systems, 2004. Proceedings. 2004 2nd International IEEE Conference*, vol. 1. 25
- KLEIN D. & MANNING C.D. 2003 « Accurate unlexicalized parsing » dans *Proceedings of the 41st annual meeting of the association for computational linguistics*. 60
- KLERKE S., GOLDBERG Y. & SØGAARD A. 2016 « Improving sentence compression by learning to predict gaze » dans *arXiv preprint arXiv :1604.03357*. 40, 42, 80
- KNIGHT K. & MARCU D. 2002 « Summarization beyond sentence extraction : A probabilistic approach to sentence compression. » dans *Artificial Intelligence*, vol. 139(1), pp. 91–107. 36, 38, 39, 41, 46, 90
- KULLBACK S. & LEIBLER R.A. 1951 « On information and sufficiency » dans *The annals of mathematical statistics*, vol. 22(1), pp. 79–86. 22
- KUPIEC J., PEDERSEN J. & CHEN F. 1995 « A trainable document summarizer » dans *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 68–73. 21
- LAFFERTY J., MCCALLUM A. & PEREIRA F.C. 2001 « Conditional random fields : Probabilistic models for segmenting and labeling sequence data ». 72
- LEE H., CHANG A., PEIRSMAN Y., CHAMBERS N., SURDEANU M. & JURAFSKY D. 2013 « Deterministic coreference resolution based on entity-centric, precision-ranked rules » dans *Computational Linguistics*, vol. 39(4), pp. 885–916. 32
- LEVY R. & ANDREW G. 2006 « Tregex and tsurgeon : tools for querying and manipulating tree data structures » dans *Proceedings of the fifth international conference on Language Resources and Evaluation*, pp. 2231–2234. Citeseer. 60
- LEWIS D. et al. 1996 « Dying for information » dans *Reuters Business Information, London*. 5
- LI C., LIU F., WENG F. & LIU Y. 2013 « Document summarization via guided sentence compression » dans *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 490–500. 40, 48, 68, 71, 74, 75, 78, 84
- LIN C.Y. 1999 « Training a selection function for extraction » dans *Proceedings of the eighth international conference on Information and knowledge management*, pp. 55–62. 21
- LIN C.Y. 2004 « Rouge : A package for automatic evaluation of summaries » dans *Text Summarization Branches Out*. 29
- LITVAK M., LAST M. & FRIEDMAN M. 2010 « A new approach to improving multilingual summarization using a genetic algorithm » dans *48th Meeting of the Association for Computational Linguistics*, pp. 927–936. 22

- LOUIS A. & NENKOVA A. 2009 « Automatically evaluating content selection in summarization without human models » dans *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, vol. 1, pp. 306–314. 22, 30
- LUHN H.P. 1958 « The automatic creation of literature abstracts » dans *IBM Journal of research and development*, vol. 2(2), pp. 159–165. 6, 20
- MANN W.C. & THOMPSON S.A. 1988 « Rhetorical structure theory : Toward a functional theory of text organization » dans *Text-Interdisciplinary Journal for the Study of Discourse*, vol. 8(3), pp. 243–281. 26
- MANNING C.D., SURDEANU M., BAUER J., FINKEL J., BETHARD S.J. & MCCLOSKEY D. 2014 « The Stanford CoreNLP natural language processing toolkit » dans *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60. 75
- MARCU D. 1997 « From discourse structures to text summaries » dans *Proceedings of the ACL*, vol. 97, pp. 82–88s. 27
- MARCU D. 2002 *The Theory and Practice of Discourse Parsing and Summarization*, MA : MIT Pres, Cambridge. 27
- MARCUS M., SANTORINI B. & MARCINKIEWICZ M. 1994 « Building a large annotated corpus of english : The penn treebank » dans *Computational Linguistics*, vol. 19(1), pp. 313–330. 39
- MARTINS A.F. & SMITH N.A. 2009 « Summarization with a joint model for sentence extraction and compression » dans *Proceedings of the Workshop on Integer Linear Programming for Natural Language Processing*, pp. 1–9. 39, 40, 42
- MIHALCEA R. & TARAU P. 2004 « Textrank : Bringing order into text » dans *Proceedings of the 2004 conference on empirical methods in natural language processing*. 24
- MILLER G.A. 1995 « Wordnet : a lexical database for english » dans *Communications of the ACM*, vol. 38(11), pp. 39–41. 26
- MINEL J.L., DESCLÉS J.P., CARTIER E., CRISPINO G., HAZEZ S.B. & JACKIEWICZ A. 2001 « Résumé automatique par filtrage sémantique d'informations dans des textes » dans *Traitement automatique du langage naturel*, vol. 20(3), pp. 369–395. 14, 26
- MITKOV R. 2014 *Anaphora Resolution*, Routledge. 32
- MOLINA VILLEGAS A. 2013 *Compression automatique de phrases : une étude vers la génération de résumés*, Thèse de doctorat Université d'Avignon et des Pays de Vaucluse Avignon, France. 26, 36, 40, 41, 48, 90
- MORRIS J. & HIRST G. 1991 « Lexical cohesion computed by thesaural relations as an indicator of the structure of text » dans *Computational Linguistics*, vol. 17(1), pp. 21–48. 26
- NAPOLES C., VAN DURME B. & CALLISON-BURCH C. 2011 « Evaluating sentence compression : Pitfalls and suggested remedies » dans *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pp. 91–97. 41, 42, 90
- NENKOVA A. & PASSONNEAU R. 2004 « Evaluating content selection in summarization : The pyramid method » dans *Proceedings of the human language technology conference of the north american chapter of the association for computational linguistics : Hlt-naacl 2004*. 29

- NENKOVA A., PASSONNEAU R. & MCKEOWN K. 2007 « The pyramid method : Incorporating human content selection variation in summarization evaluation » dans *ACM Transactions on Speech and Language Processing (TSLP)*. 29
- NEW B., PALLIER C., FERRAND L. & MATOS R. 2001 « Une base de données lexicales du français contemporain sur internet : LexiqueTM//a lexical database for contemporary french : LexiqueTM » dans *L'année psychologique*, vol. 101(3), pp. 447–462. 76
- NOMOTO T. 2007 « Discriminative sentence compression with conditional random fields » dans *Information processing & management*, vol. 43(6), pp. 1571–1587. 40, 68, 74, 75, 80, 82, 84
- O'DONNELL M. 1997 « Variable-length on-line document generation » dans *Proceedings of the 6th European Workshop on Natural Language Generation*, pp. 82–91. 27
- ONO K., SUMITA K. & MIIKE S. 1994 « Abstract generation based on rhetorical structure extraction » dans *Proceedings of the 15th conference on Computational linguistics*, vol. 1, pp. 344–348. 27
- ORE O. & ORE Y. 1962 *Theory of graphs*, vol. 38, Providence, RI : American Mathematical Society. 24
- POLLOCK J.J. & ZAMORA A. 1975 « Automatic abstracting research at chemical abstracts service » dans *Journal of Chemical Information and Computer Sciences*, vol. 15(4), pp. 226–232. 21
- RADEV D., ALLISON T., BLAIR-GOLDENSOHN S., BLITZER J., ÇELEBI A., DIMITROV S., DRABEK E., HAKIM A., LAM W., LIU D., OTTERBACHER J., QI H., SAGGION H., TEUFEL S., TOPPER M., WINKEL A. & ZHANG Z. 2004 « Mead — a platform for multidocument multilingual text summarization » dans *LREC*. 6, 25
- RADEV D.R. 2000 « A common theory of information fusion from multiple text sources step one : cross-document structure » dans *Proceedings of the 1st SIGdial workshop on Discourse and dialogue*, vol. 10, pp. 74–83. 27
- RADEV D.R. & MCKEOWN K.R. 1998 « Generating natural language summaries from multiple on-line sources » dans *Computational Linguistics*, vol. 4(3). 17
- SAGGION H. 2017 « Automatic text simplification » dans *Synthesis Lectures on Human Language Technologies*, vol. 10(1), pp. 1–137. 36
- SAGGION H. & LAPALME G. 2002 « Generating indicative-informative summaries with sumum » dans *Computational Linguistics*, vol. 28(4), pp. 497–526. 15, 17
- SAKTI S., ILHAM F., NEUBIG G., TODA T., PURWARIANTI A. & NAKAMURA S. 2015 « Incremental sentence compression using lstm recurrent networks » dans *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pp. 252–258. IEEE. 40, 42, 80
- SHANNON C.E. 1948 « A mathematical theory of communication » dans *Bell System Technical Journal*, vol. 27, pp. 379–423. 23
- SIDDHARTHAN A. 2014 « A survey of research on text simplification » dans *ITL-International Journal of Applied Linguistics*, vol. 165(2), pp. 259–298. 36, 37

- SIDDHARTHAN A., NENKOVA A. & MCKEOWN K. 2004 « Syntactic simplification for improving content selection in multi-document summarization » dans *Proceedings of the 20th international conference on Computational Linguistics*. 46, 47
- SILBER H.G. & MCCOY K.F. 2002 « Efficiently computed lexical chains as an intermediate representation for automatic text summarization » dans *Computational Linguistics*, vol. 28(4), pp. 487–496. 26
- SILVEIRA S.B. & BRANCO A. 2012 « Enhancing multi-document summaries with sentence simplification » dans *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 6, 46, 47
- SJÖBERGH J. 2007 « Older versions of the rougeeval summarization evaluation system were easier to fool » dans *Information Processing and Management*, vol. 43(6), pp. 1500–1505. 29
- SUMMAC 1998 « Final report of summac », Rap. Tech. National Institute of Standards and Technology (NIST). 28, 30
- SUNEETHA M. & FATIMA S.S. 2011 « Corpus based automatic text summarization system with hmm tagger » dans *International Journal of Soft Computing and Engineering (IJSCE) ISSN*, pp. 2231–2307. 22
- SUTSKEVER I., VINYALS O. & LE Q.V. 2014 « Sequence to sequence learning with neural networks » dans *Advances in neural information processing systems*, pp. 3104–3112. 40
- SUTTON C., MCCALLUM A. *et al.* 2012 « An introduction to conditional random fields » dans *Foundations and Trends® in Machine Learning*, vol. 4(4), pp. 267–373. 72, 73, 74
- SVORE K.M., VANDERWENDE L. & BURGESS C.J. 2007 « Enhancing single-document summarization by combining ranknet and third-party sources » dans *Emnlp-conl*. 6, 25
- TEUFEL S. & MOENS M. 2002 « Summarizing scientific articles : experiments with relevance and rhetorical status » dans *Computational linguistics*, vol. 28(4), pp. 409–445. 27
- TORRES-MORENO J.M. 2012 « Artex is another text summarizer » dans *arXiv.org (Cornell University Library)*. 22, 23
- TORRES-MORENO J.M. 2014 *Automatic Text Summarization*, John Wiley & Sons. 14, 15, 16, 19, 21, 22, 23, 24, 25, 30, 31, 36, 99, 100
- TORRES-MORENO J.M., SAGGION H., DA CUNHA I., VELÁZQUEZ-MORALES P. & SANJUAN E. 2010 « Évaluation automatique de résumés avec et sans référence » dans *17e Conférence sur le Traitement Automatique des Langues Naturelles (TALN) à l'Université de Montréal et Ecole Polytechnique de Montréal*. 30
- TORRES-MORENO J.M., VELÁZQUEZ-MORALES P. & MEUNIER J.G. 2002 « Condensés de textes par des méthodes numériques » dans *JADT*. 22, 23
- TOUTANOVA K., BROCKETT C., GAMON M., JAGARLAMUDI J., SUZUKI H. & VANDERWENDE L. 2007 « The pythy summarization system : Microsoft research at duc 2007 » dans *Proceedings of DUC*. 38, 42

- TURNER J. & CHARNIAK E. 2005 « Supervised and unsupervised learning for sentence compression » dans *Proceedings of the 43rd Annual Meeting of ACL*, pp. 290–297. 36, 39, 41
- TURNER V., GANTZ J.F. & REINSEL D. 2014 *The Digital Universe of Opportunities : Rich Data and the Increasing Value of the Internet of Things (Executive Summary)*, IDC Analyze the Future. 14
- UNNO Y., NINOMIYA T., MIYAO Y. & TSUJII J.I. 2006 « Trimming cfg parse trees for sentence compression using machine learning approaches » dans *Proceedings of the COLING/ACL on Main conference poster sessions*, pp. 850–857. 39, 42
- VANDERWENDE L., SUZUKI H. & BROCKETT C. 2006 « Microsoft research at duc 2006 : task-focused summarization with sentence simplification and lexical expansion » dans *Proceedings of the Document Understanding Conference, DUC-2006, New York, USA*, Cite-seer. 6, 46, 48, 63
- VILLARS R.L., OLOFSON C.W. & EASTWOOD M. 2011 « Big data : What it is and why you should care » dans *White Paper, IDC*, vol. 14, pp. 1–14. 5
- WANG L., RAGHAVAN H., CASTELLI V., FLORIAN R. & CARDIE C. 2016 « A sentence compression based framework to query-focused multi-document summarization » dans *arXiv.org (Cornell University Library)*. 38, 42, 47, 48
- WHITE M., KORELSKY T., CARDIE C., NG V., PIERCE D. & WAGSTAFF K. 2001 « Multidocument summarization via information extraction » dans *Proceedings of the first international conference on Human language technology research*, pp. 1–7. 17
- YOSHIDA Y., SUZUKI J., HIRAO T. & NAGATA M. 2014 « Dependency-based discourse parser for single-document summarization » dans *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1834–1839. 27
- YOUSFI-MONOD M. & PRINCE V. 2006 « Compression de phrases par élagage de leur arbre morphosyntaxique. une première application sur les phrases narratives. » dans *Revue des Sciences et Technologies de l'Information-Série TSI : Technique et Science Informatiques*, vol. 25(4), pp. 437–468. 38, 41, 47, 59

Annexes

A.1 LEXRANK : équation de pondération de phrase

Soient :

- $p(s_i)$ le poids de la phrase s à la position i (ou vecteur i)
- N le nombre de phrases (ou nœuds)
- d le facteur d'amortissement (*damping factor*)
- $In(s_i)$ l'ensemble de nœuds connectés pointant vers s_i
- $Out(s_i)$ l'ensemble de nœuds connectés partant de s_i
- $sim(s_i, s_j)$ le cosinus des vecteurs s_i et s_j

On a :

$$p(s_i) = \frac{d}{N} + (1 - d) \sum_{s_j \in In(s_i)} \frac{sim(s_i, s_j)}{\sum_{s_k \in Out(s_j)} sim(s_k, s_j)} p(s_j) \quad (1)$$

L'équation est récursive, ce qui induit que le poids de chaque nœud dépend de celui de ceux qui lui sont connectés. L'itération est arrêtée lorsque la valeur entre le tour courant et le précédent n'a pas été modifiée de plus de 0.0001.

Références : Erkan & Radev (2004) et Torres-Moreno (2014 : 80-81).

A.2 TEXTRANK : équation de pondération de phrase

Soient :

- $p(s_i)$ le poids de la phrase s à la position i
- N le nombre de phrases (ou nœuds)
- d le facteur d'amortissement (*damping factor*)
- $In(s_i)$ l'ensemble de nœuds connectés pointant vers s_i
- $Out(s_i)$ l'ensemble de nœuds connectés partant de s_i
- $p(s_i, s_j)$ le poids entre les phrases s_i et s_j (arrêtes du graphe)

On a :

$$p(s_i) = (1 - d) + d \sum_{s_j \in In(s_i)} \frac{p(s_i, s_j)}{\sum_{s_k \in Out(s_j)} p(s_j, s_k)} p(s_j) \quad (2)$$

L'opération est également récursive et a la même condition d'arrêt que LEXRANK. Le poids entre deux phrases se base sur un comptage d'occurrences, donné par l'équation suivante :

$$p(s_i, s_j) = \frac{\sum_{m \in s_i, s_j} C_m^{s_i} + C_m^{s_j}}{\log |s_i| + \log |s_j|} \quad (3)$$

$C_m^{s_i}$ vaut le nombre de fois que la phrase s_i contient le mot m . La valeur est normalisée par les logarithmes de la longueur des phrases considérées, afin de désavantager les phrases trop longues.

Références : Torres-Moreno (2014 : 81-83).

A.3 Informations supplémentaires sur le tag des signes

Voici quelques informations complémentaires sur les tags des signes. Chaque tag est un acronyme composé de trois parties :

1. Les deux premières lettres indiquent s'il s'agit d'une frontière gauche (SS) ou droite (ES) d'un élément subordonné.
2. La lettre suivante indique le niveau de projection syntaxique de l'élément subordonné (voir Evans & Orasan 2013 : 10).
3. Enfin, les dernières lettres indiquent la catégorie syntaxique à laquelle la tête du constituant subordonné appartient : verbe (V), nom (N), adjectif (A), adverbe (Adv) ou préposition (P).

Voici une liste des tags de signes de subordination que l'on peut rencontrer, à des fréquences variées, avec des exemples de phrases taguées :

1. SSCCV / ESCCV :
— *Matthew Young [, who]_{SSCCV} was three weeks old [,]_{ESCCV} died of heart failure after he was given peppermint water [which]_{SSCCV} contained 20 times the correct dose of chloroform.*
2. SSMV / ESMV :
— *He said he had not realised the significance of notes at the side of the page [,]_{SSMV} made by another chemist, which clarified the method.*
3. SSMN / ESMN :
— *His supervisor [,]_{SSMN} 27-year-old Lisa Taylor Lloyd [,]_{ESMN} then failed to notice his mistake.*
4. SSMA / ESMA :
— *Dad Neil [,]_{SSMA} 24 [,]_{ESMA} said : « We want them to go to jail and to be made an example of. »*
5. SSMAdv / ESMAdv :
— *Incredibly [,]_{ESMAdv} council chiefs in Nottingham [...] never suspected he was a double-dealer.*
6. SSMP / ESMP :
— *The item [,]_{SSMP} in 1997 [,]_{ESMP} referred to a forthcoming memorial concert.*

A.4 Modifications apportées au corpus De Loupy *et al*

Nous avons modifié 5 phrases du corpus de De Loupy *et al.* (2010), présenté au chapitre 7. Voici les modifications apportées :

- phrase 514 : La phrase contenait en réalité 2 phrases. Comme la deuxième phrase n'a pas toujours été conservée par les annotateurs, nous l'avons supprimée.
- phrase 1421 : Un espace a été rajouté entre "sens" et le point final, pour éviter une erreur de tokenisation.
- phrase 2142 : Il manquait un espace entre "réformes" et "de" pour l'annotateur 4.
- phrase 2544 : Il manquait une lettre au premier mot pour l'annotateur 4.
- phrase 4483 : Le token "17e" a été réécrit en "dix-septième" car cela causait une erreur de tokenisation.

A.5 Script pour le compresseur symbolique

Nous retranscrivons ici le code permettant de compresser une phrase selon l'approche décrite au chapitre 5. Il s'agit d'un module écrit en Perl5 contenant une méthode principale qui, pour une liste de phrases taguées avec le tagueur de signes et parsées, renvoie une liste de phrases compressées. Il fonctionne avec un ensemble de scripts Tsurgeon (plus de détails à ce sujet en section 5.1.3). Ce travail étant le fruit d'une collaboration avec le RGCL, centre de recherche situé au Royaume-Uni, et le compresseur ne pouvant que compresser du texte en anglais, les commentaires du code sont en anglais.

```

1 #####
# This is a Perl5 module used to compress parsed and tagged English sentences.
#
# The compression is done using the free software Tsurgeon, which is a
# tool from the Stanford University for modifying trees that match a
6 # particular Tregex pattern. For more information, see README files in
# the tregex folder.
#####
# @author Jeremy Chelala : j.n.chelala@wlv.ac.uk or jeremy.chelala@student.uclouvain.be
# @version project begun on 06/07/2017, last modified on 01/08/2017
11 #####
# IMPORTANT : For the following functions to be used, your system must
# have the following directories :
#   - 'user/stanford-tregex-2017-06-09': the stanford tregex and tsurgeon
#     scripts and documents, downloadable at https://nlp.stanford.edu/software/tregex.shtml
16 #   - '../my_tsurgeon_scripts' : the scripts used in the
#     functions (see 'Common Variables' section)
#####

package Compress;

21 use strict;
use warnings;
use feature 'say';

26 ##### Common variables #####

my $go_to_tsurgeon_dir = "cd stanford-tregex-2017-06-09";
my $basic_command = "./tsurgeon.sh -treeFile";
my $prune_commas = "../my_tsurgeon_scripts/prune_commas";
31 my $prune_end_sub = "../my_tsurgeon_scripts/prune_end_sub";
my $prune_end_sub_2 = "../my_tsurgeon_scripts/prune_end_sub_2";
my $delete_beginning_sub = "../my_tsurgeon_scripts/delete_beginning_sub";
my $prune_appositive = "../my_tsurgeon_scripts/prune_appositive";
my $prune_appositive_2 = "../my_tsurgeon_scripts/prune_appositive_2";
36 my $delete_end_appositive = "../my_tsurgeon_scripts/delete_end_appositive";
my $prune_np_sub = "../my_tsurgeon_scripts/prune_np_sub";
my $prune_dashes = "../my_tsurgeon_scripts/prune_dashes";
my $delete_dashes_2 = "../my_tsurgeon_scripts/delete_dashes_2";
my $delete_end_dash = "../my_tsurgeon_scripts/delete_end_dash";
41 my $prune_pp = "../my_tsurgeon_scripts/prune_pp";
my $delete_beginning_pp = "../my_tsurgeon_scripts/delete_beginning_pp";
my $prune_ending_pp = "../my_tsurgeon_scripts/prune_ending_pp";
my $prune_as_if = "../my_tsurgeon_scripts/prune_as_if";

46 my $script_queue = [1,2,3,4,5,6,7,8,9,10,11,12,13,14];
my $temp_file = "temporary.txt";

##### Main functions #####

51 # Compress a text from an array reference of tagged and parsed sentences
# without constraint on compression
# @arg an array reference of tagged and parsed sentences
# @post an array reference of compressed tagged and parsed sentences
sub main_text {
56   my $sentences = shift;
   my @sentence_array = ();

   foreach my $sentence (@{$sentences}) {
61     my $compressed_sent = main($sentence);
     push(@sentence_array, $compressed_sent);
   }

   return \@sentence_array;
66 }

# Main function to compress a sentence

```

```

# @arg a tagged and parsed sentence
# @post a compressed tagged and parsed sentence
71 sub main {
    my $sentence = shift;
    my $scripts = $script_queue;
76
    # Make a copy of the script queue
    my @copy = ();
    foreach (@{$scripts}) {
        push(@copy, $_);
81    }

    return main_aux($sentence, \@copy);
}

86 # Recursive function that applies all the rules from the script queue
# on the parameter sentence
# @arg a tagged and parsed sentence
# @post a compressed tagged and parsed sentence
sub main_aux {
91
    my $sentence = shift;
    my $queue = shift;

    # Base case
96    if (@{$queue} == 0) {
        return $sentence;
    }

    my $current_script = shift(@{$queue});
101    $sentence = execute_script($sentence, $current_script, $temp_file);
    return main_aux($sentence, $queue);
}

# Execute the indicated Tsurgeon script on the current instance
# @arg a tagged and parsed sentence, a script reference (nb), a file
# @post the result of the execution of Tsurgeon script on the sentence.
# The file is removed during the process!
sub execute_script {
111
    my $sentence = shift;
    my $script_nb = shift;
    my $file = shift;

    # Write the sentence in $file
116    open my $fh, '>', $file or die("Cannot open temporary file!");
    print $fh $sentence;
    close $fh;

    # If $sentence contains a dash
121    if ($script_nb <= 3 and $sentence =~ /: -/) {

        if ($script_nb == 1) {
            $sentence = dashes($file);
        }
126    elsif ($script_nb == 2) {
            $sentence = dashes_2($file);
        }
        elsif ($script_nb == 3) {
            $sentence = end_dash($file);
131    }

    } # if ($script_nb <= 3 and $sentence =~ /: -/)

    elsif ($script_nb <= 14 and $sentence =~ /-ES|-SS/) {
136
        # If $sentence contains a tagged comma
        if ($script_nb <= 13 and $sentence =~ /-ES|-SS/) {

            if ($script_nb == 4) {
141                $sentence = appositive($file);
            }
            elsif ($script_nb == 5) {
                $sentence = appositive_2($file);
            }
146    elsif ($script_nb == 6) {
                $sentence = end_appositive($file);
            }
            elsif ($script_nb == 7) {
                $sentence = comma_sub($file);
151    }
            elsif ($script_nb == 8) {
                $sentence = end_sub($file);
            }
        }
    }
}

```

```

156     elif($script_nb == 9) {
        $sentence = end_sub_2($file);
    }
    elif($script_nb == 10) {
        $sentence = beginning_sub($file);
    }
161 # If $sentence contains a prepositional phrase
    elif($sentence =~ /PP /) {

        if($script_nb == 11) {
            $sentence = pp($file);
166        }
        elif($script_nb == 12) {
            $sentence = beginning_pp($file);
        }
        elif($script_nb == 13) {
            $sentence = ending_pp($file);
171        }
    } # elif($sentence =~ /PP /)

    } # if($script_nb <=13 and $sentence =~/, -ES|, -SS/)

176 elif($script_nb == 14) {
    $sentence = np_sub($file);
    }
    } # elif($script_nb <= 14 and $sentence =~ /-ES|-SS/)

181 # Remove the temporary $file
    'rm $file';
    return $sentence;
186 }

# Do one modification using the additional function
# @arg a treeFile containing an English sentence and the original
# sentence it contains
191 # @post a shorter and grammatically correct version of the sentence
sub one_modif_no_constraints {

    my $file = shift;
    my $original = shift;
196 my $result = '';

    $result = dashes($file);
    if(length($result) < length($original)) {return $result;}

201 $result = dashes_2($file);
    if(length($result) < length($original)) {return $result;}

    $result = end_dash($file);
    if(length($result) < length($original)) {return $result;}
206 $result = appositive($file);
    if(length($result) < length($original)) {return $result;}

    $result = appositive_2($file);
    if(length($result) < length($original)) {return $result;}
211 $result = end_appositive($file);
    if(length($result) < length($original)) {return $result;}

216 $result = comma_sub($file);
    if(length($result) < length($original)) {return $result;}

    $result = end_sub($file);
    if(length($result) < length($original)) {return $result;}
221 $result = end_sub_2($file);
    if(length($result) < length($original)) {return $result;}

    $result = beginning_sub($file);
    if(length($result) < length($original)) {return $result;}
226 $result = np_sub($file);
    if(length($result) < length($original)) {return $result;}

231 $result = pp($file);
    if(length($result) < length($original)) {return $result;}

    $result = beginning_pp($file);
    if(length($result) < length($original)) {return $result;}
236 $result = ending_pp($file);
    if(length($result) < length($original)) {return $result;}

```

```

241     return undef;
}

##### Functions using tsurgeon scripts #####
#
# The following functions all take a file containing one sentence as
# parameter and use bash commands to run predefined Tsurgeon scripts.
# They return a compressed tree if the tregex expression used in the
# tsurgeon script is matched, the same argument tree otherwise

251 # @arg and @post statements are the same for all the below functions. That is :
#     @arg a tagged and parsed sentence
#     @post a shorter and grammatically correct version of the sentence

256 # Prune a syntactic tree containing a subordinate or commentary clause
# delimited by two tagged commas
sub comma_sub {
    my $file = shift;
    $file = '../' . $file;

261     my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_commas);";

    my $result = `$command`;
    return $result;
266 }

# Delete a subordinate clause that begins with a tagged comma and is
# situated at the end of the sentence. Compress only clauses introduced
# by which/who/that, possibly preceded by a preposition.
271 sub end_sub {
    my $file = shift;
    $file = '../' . $file;

    my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_end_sub);";

276     my $result = `$command`;
    return $result;
}

281 # Delete a subordinate clause that begins with a tagged comma and is
# situated at the end of the sentence. Compress only clauses introduced
# by which/who/that, possibly preceded by a preposition.
sub end_sub_2 {
286     my $file = shift;
    $file = '../' . $file;

    my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_end_sub_2);";

    my $result = `$command`;
291     return $result;
}

# Delete a subordinate clause at the beginning of the sentence if ended
# ended by a tagged comma
296 sub beginning_sub {
    my $file = shift;
    $file = '../' . $file;

    my $command = "($go_to_tsurgeon_dir; $basic_command $file $delete_beginning_sub);";

301     my $result = `$command`;
    return $result;
}

306 # Delete an appositive delimited by two tagged commas from a file
# containing a parsed and tagged sentence
sub appositive {
    my $file = shift;
    $file = '../' . $file;

311     my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_appositive);";

    my $result = `$command`;
    return $result;
316 }

# Delete an appositive delimited by two commas tagged as <SSMP> and
# <ESMP> from a file containing a parsed and tagged sentence.
sub appositive_2 {
321     my $file = shift;
    $file = '../' . $file;

    my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_appositive_2);";

```

```

326   my $result = '$command';
      return $result;
    }

    # Delete an appositive delimited by a tagged comma as <SSM(A|N|P)> and
331   # a final dot from a file containing a parsed and tagged sentence.
    sub end_appositive {
        my $file = shift;
        $file = '../' . $file;

336   my $command = "($go_to_tsurgeon_dir; $basic_command $file $delete_end_appositive;);";

        my $result = '$command';
        return $result;
    }
341

    # Delete a clause subordinating a NP and introduced by a wh-word, except
    # if the head of the NP is a singular common noun
    sub np_sub {
346   my $file = shift;
        $file = '../' . $file;

        my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_np_sub;);";

351   my $result = '$command';
        return $result;
    }

356   # Delete a commentary clause surrounded by dashes
    sub dashes {
        my $file = shift;
        $file = '../' . $file;

361   my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_dashes;);";

        my $result = '$command';
        return $result;
    }
366

    # Delete a commentary clause surrounded by dashes
    sub dashes_2 {
        my $file = shift;
        $file = '../' . $file;

371   my $command = "($go_to_tsurgeon_dir; $basic_command $file $delete_dashes_2;);";

        my $result = '$command';
        return $result;
    }
376

    # Delete a commentary clause introduced by a dash and situated at the
    # end of the sentence
    sub end_dash {
381   my $file = shift;
        $file = '../' . $file;

        my $command = "($go_to_tsurgeon_dir; $basic_command $file $delete_end_dash;);";

386   my $result = '$command';
        return $result;
    }

    # Delete an independent 'as-if' clause in a parsed sentence
391   sub as_if {
        my $file = shift;
        $file = '../' . $file;

        my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_as_if;);";

396   my $result = '$command';
        return $result;
    }

401   # Delete a prepositional phrase between two commas
    sub pp {
        my $file = shift;
        $file = '../' . $file;

406   my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_pp;);";

        my $result = '$command';
        return $result;
    }

```

```

}
411 # Delete a prepositional phrase at the beginning of the sentence and
# ended by a tagged comma
sub beginning_pp {
416   my $file = shift;
   $file = '../' . $file;

   my $command = "($go_to_tsurgeon_dir; $basic_command $file $delete_beginning_pp);";

   my $result = '$command';
421   return $result;
}

# Delete a prepositional phrase at the end of the sentence and
# introduced by a tagged comma
426 sub ending_pp {
   my $file = shift;
   $file = '../' . $file;

   my $command = "($go_to_tsurgeon_dir; $basic_command $file $prune_ending_pp);";
431   my $result = '$command';
   return $result;
}

436 ##### Additional function #####

# Get the syntactic tree of a parsed and tagged sentence
# @arg a tagged and parsed sentence
441 # @post a character string containing its syntactic tree

sub get_sent_from_file {
446   my $file = shift;
   my $sentence = '';

   open(my $fh, '<', $file) or die;

   while(my $line = <$fh>) {
451     $sentence .= $line;
   }

   close $fh;
   return $sentence;
456 }

1;

```

A.6 Scripts pour le compresseur statistique

Nous donnons les quatre scripts que nous avons conçus dans le cadre de l'implémentation de notre compresseur probabiliste (cf. chapitre 6). Ils sont écrits en Python 3 et remplissent les fonctions suivantes :

- `sent_compressor.py` : Script principal. Son exécution permet, selon les options, d'entraîner un modèle CRF pour la compression, de compresser des phrases nouvelles (prédiction) et d'évaluer un modèle de compression.
- `crf.py` : Script contenant la classe implémentant le CRF.
- `sentence_parser.py` : Script offrant des méthodes permettant d'annoter (tokenisation, *pos tagging*, etc.) une phrase pour la compression.
- `utils.py` : Script offrant des méthodes utilitaires de lecture et d'écriture dans des fichiers aux formats divers.

Les outils et packages nommés dans ces scripts sont décrits en sections 6.4.1 et 6.3 (sauf les packages standards en Python).

A.6.1 `sent_compressor.py`

```

2  #! /etinfo/users/2018/chelala/env_memoire/bin/python
  # -*- coding: utf-8 -*-

  """
  Script exécutable utilisé pour l'entraînement et l'évaluation de modèles CRF de
  compression de phrases, à partir de corpus d'entraînement et de test, annotés ou
7  non annotés

  @author Jeremy Chelala
  @version juin 2018

12 #####
  Conditions d'usage :

      - Interpréteur : Python 3 (application développée en Python 3.4)
      - Modules externes installés :
17     - nltk (aucun modèle à télécharger, uniquement utilisé pour la classe Tree)
      - stanfordcorenlp
      - sklearn-crfsuite
      - sklearn
      - scipy

22 #####
  Utilisation :

  Le script comprend 3 fonctions principales, lancée avec les commandes suivantes :
27  python sent_compressor.py train
  python sent_compressor.py predict
  python sent_compressor.py evaluate

  La commande "python sent_compressor.py train_and_eval" combine la première et la
32 troisième fonction.

  1) Entraînement :

  La fonction "train" entraîne un modèle CRF à partir d'un jeu d'entraînement et sauvegarde
37 le résultat dans un fichier pickle.
  Commande : $ python sent_compressor.py train corpus -nb nb_training_ex [-o out]

      - corpus : chemin vers le fichier contenant le corpus d'entraînement. Le fichier est
        considéré comme un fichier pickle contenant les données au format attendu.
42 - "-nb nb_training_ex" : un entier > 0 et <= taille du corpus qui donne le nombre
        d'exemples à prendre en compte pour l'entraînement. Si non précisé, tous les exemples
        sont utilisés.
      - "-o out" : chemin vers le fichier dans lequel stocker le résultat. Si non précisé,
        le modèle est sauvegardé dans le répertoire courant dans un fichier
47 nommé "crf_compressor.pickle".

  2) Prédiction(s)

52 La fonction "predict" compresse un/des phrase(s) contenues dans un fichier texte selon un

```

```

modèle crf déjà entraîné.
Commande : $ python sent_compressor.py predict file [--model "path/to/model"] [--o out]

-- file : un fichier pickle contenant une liste de phrases déjà annotées
-- "--model path/to/model" : le modèle crf est chargé à partir du fichier mentionné.
Il doit s'agir d'un fichier compressé avec pickle. Si non précisé, cherche le fichier
nommé "crf_compressor.pickle" dans le répertoire courant
-- "--o out" : chemin vers le fichier dans lequel stocker le résultat. Si non précisé,
affiche le résultat sur l'entrée standard

3) Évaluation

La fonction "evaluate" évalue un modèle entraîné à partir d'un jeu de test. Elle calcule
la précision, le rappel et la mesure F1 obtenus.
Commande : $ python sent_compressor.py evaluate test_set --nb index [--model "path/to/mdl"]
[--o out]

-- test_set : chemin vers le fichier contenant le corpus de test. Le fichier est
considéré comme un fichier pickle contenant les données au format attendu
-- "--nb index" : un entier > 0 et <= taille du corpus qui donne la position du premier
exemple à inclure dans le jeu de test. Si non mentionné, = 0
-- "--model path/to/model" : le modèle crf est chargé à partir du fichier mentionné.
Il doit s'agir d'un fichier compressé avec pickle. Si non précisé, cherche le fichier
nommé "crf_compressor.pickle" dans le répertoire courant
-- "--o out" : chemin vers le fichier dans lequel stocker le résultat. Si non précisé,
affiche le résultat sur l'entrée standard

4) Entraînement et évaluation consécutives

Il est possible d'entraîner un modèle et de l'évaluer directement à partir d'un corpus
annoté. Le corpus est divisé en deux jeux de données, l'un dédié à l'entraînement, l'autre
au test. Lorsque l'entraînement est fini, le programme propose à l'utilisateur de
sauvegarder le modèle, de même pour l'évaluation.

Commande : $ python sent_compressor.py train_and_evaluate corpus --nb nb_training_ex

-- corpus : chemin vers le fichier contenant le corpus d'entraînement. Le fichier est
considéré comme un fichier pickle contenant les données au format attendu.
-- "--nb nb_training_ex" ou "--n nb" : un entier > 0 et <= taille du corpus qui donne le
nombre d'exemples à prendre en compte pour l'entraînement. Le reste des exemples est
utilisé pour l'évaluation.
"""

import argparse
import time

import utils as u
from sentence_parser import SentenceParser
from corpus_parser import CorpusParser
from crf import CRFCompressor

# Variable globale
default_model = 'crf_compressor.pickle'

"""
Fonction principale. Gère et les arguments et appelle la sous-fonction adéquate
"""
def main():

    # Crée objet argparse
    argparser = argparse.ArgumentParser(description="Entraîne et/ou évalue " \
        "un compresseur automatique de phrases pour le français")
    argparser.add_argument('quoi', help='train/predict/evaluate/train_and_evaluate' \
        '(seules ces 4 valeurs sont acceptées)')
    argparser.add_argument('fichier')
    argparser.add_argument('--o', dest='out')
    argparser.add_argument('--model', nargs='?', default=default_model)
    argparser.add_argument('--nb', type=int)

    args = argparser.parse_args()

    # Option "train"
    if(args.quoi == 'train'):

        corpus = args.fichier

        train_nb = None
        if(args.nb is not None):
            train_nb = args.nb

        data = u.load_data_pickle(corpus)

        if(train_nb is not None):
            data = data[:train_nb]

```

```

train(data, args.out)

# Option "predict"
142 elif(args.quoi == 'predict'):

    file = args.fichier

    model = args.model
    data = u.load_data_pickle(file)
    index = args.nb
    data = data[index:]

    print("En train de compresser...")

152 sentences, compressed = predict(model, data)

    print("Terminé !")

157 text = "\nRésultats des prédictions : \n\n"
    for sent, comp in zip(sentences, compressed):
        if(sent == comp):
            text += "Originale : {}\nCompressée : {} [Aucun token supprimé]\n" \
                "\n".format(sent, comp)
162         else:
            text += "Originale : {}\nCompressée : {}\n\n".format(sent, comp)

    if(args.out is None):
        print(text)
167     else:
        u.write_txt(text, args.out)

# Option "evaluate"
172 elif(args.quoi == 'evaluate'):

    test_set = args.fichier

    model = args.model
    test_data = u.load_data_pickle(test_set)
177

    print("Évaluation en cours...")
    if(args.nb is None):
        res = evaluate(model, test_data)
    else:
182         res = evaluate(model, test_data, index=args.nb)

    print("Terminée !")

    if(args.out is None):
        print(res)
187     else:
        u.write_txt(res, args.out)

elif(args.quoi == "train_and_evaluate" or args.quoi == "train_and_eval"):

192     if(args.nb is None):
        print("Un nombre d'exemples pour l'entraînement doit être précisé avec " \
            "l'option '-nb'. Pour plus d'informations, tapez 'python sent_compressor -h'")
        exit()

197     train_nb = args.nb
    data = u.load_data_pickle(args.fichier)

    train_data = data[:train_nb]
    test_data = data[train_nb:]
202

    train_and_evaluate(train_data, test_data)

207 """
Entraîne un modèle CRF à partir de données d'entraînement. Sauve le modèle dans "out".
"""
def train(data, out):

212     crf = CRFCompressor(train_set=data, test_set=None)

    time1 = time.time()
    print("Entraînement débuté...")
    crf.train()
217     total_time = (time.time() - time1) / 60
    print("Entraînement terminé. Cela a pris {} min".format( "%.2f" % total_time))

    crf.save_model(out)

222 """

```

```

Donne les résultats des prédictions selon le modèle CRF donné.
:returns tuple(phrases originales, phrases compressées)
"""
def predict(model, sentences):
227     crf = CRFCompressor(None)
        crf.set_model_from_pickle(model)

        sp = SentenceParser()

232     or_sent = []
        compressed = []

        for sent in sentences:
237         res = crf.make_one_prediction(sent)
            tokens = crf.sentence_to_tokens(sent)
            or_sent.append(sp.untokenize(tokens))
            comp_tokens = sp.labels_to_tokens(res, tokens)
            compressed.append(sp.untokenize(comp_tokens))

242     return or_sent, compressed

"""
247 Évalue un modèle donné avec des données test. Reporte le rappel, la précision,
    la mesure F1 et le temps mis pour compresser
:returns str
"""
def evaluate(model, test_data, index=0):
252     data_crf = test_data[index:]

        crf = CRFCompressor(None)
        crf.set_model_from_pickle(model)

257     return crf.evaluate(test_set=data_crf, verbose=False)

"""
262 Entraîne et évalue un modèle CRF de compression à partir de données annotées
"""
def train_and_evaluate(data_train, data_test):

    crf = CRFCompressor(train_set=data_train, test_set=None)

267     # Entraînement
        time1 = time.time()
        print("Entraînement débuté...")
        crf.train()
        total_time = (time.time() - time1) / 60
        print("Entraînement terminé. Cela a pris {} min".format("%.2f" % total_time))

        rep = input("\nVoulez-vous sauvegarder le modèle entraîné ?\n(oui/non) : ")
        if(rep == "oui" or rep == "o"):
277             file = input("Veuillez indiquer un nom de fichier (une réponse vide sauve le " \
                            "modèle dans 'crf_compressor.pickle'"
                            ") : ")
                if(file == ''):
                    crf.save_model(default_model)
                else:
282                     crf.save_model(file)

        # Évaluation
        print("\nÉvaluation en cours...")
        result = crf.evaluate(test_set=data_test, verbose=True)
        result = "Nombre d'exemples pour l'entraînement : {}\n".format(len(data_train), len(data_test)) +
                result
        print("Terminé !\n\n")
        rep = input("Voulez-vous sauvegarder les résultats dans un fichier ?\n(oui/non) : ")
        if(rep == "oui" or rep == "o"):
292             file = input(
                "Veuillez indiquer un nom de fichier (une réponse vide sauve le modèle dans " \
                "'evaluation.txt') : ")
                if(file == ''):
                    u.write_txt(result, 'evaluation.txt')
                else:
297                     u.write_txt(result, file)
            else:
                print(result)

302 if __name__ == '__main__':
    main()

```

A.6.2 crf.py

```

1  #! /etinfo/users/2018/chelala/env_memoire/bin/python
   # -*- coding: utf-8 -*-
   """
   Classe implémentant un modèle d'entraînement de type CRF pour la compression de phrase en
6  français. L'implémentation est en partie inspirée du tutoriel de sklearn_crfsuite :
   https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html.

   Dans la fonction 'word_to_features', nous avons volontairement laissé les facteurs qui
   n'ont pas été retenus pour le modèle final, mais qui avaient été envisagés dans nos
11  expériences. Un '#' ou trois guillemets précèdent ce type de facteur.

   @auteur Jeremy Chelala
   @version juin 2018
   """
16  import sklearn_crfsuite
   from sklearn_crfsuite import metrics

   import utils as u

21  class CRFCompressor():

       'Classe CRFCompressor'

26  stopwords = u.load_data_pickle('data/stopwords.pickle')
       index = u.load_data_pickle('data/word_index.pickle')

       """
       Constructeur. Attend les jeux de données dans un format dépendant des facteurs
31  envisagés dans la méthode 'word_to_features'
       """

       def __init__(self, train_set, test_set=None):
           self.train_set = train_set
           self.test_set = test_set

36           if (train_set is not None):
               self.check_dataset(train_set)
           if (test_set is not None):
               self.check_dataset(test_set)

41           self.x_train = None
           self.y_train = None
           self.x_test = None
           self.y_test = None
46           self.crf = None

       """
       Vérifie le format des données
       :raise TypeError
51       """

       @staticmethod
       def check_dataset(dataset):

           if (not isinstance(dataset, list)):
               raise TypeError("Le jeu de données n'a pas le bon format. " \
31               "Need list, found {}".format(type(dataset)))
           if (not isinstance(dataset[0], list)):
               raise TypeError("Le jeu de données n'a pas le bon format. (Need list, found " \
36               " {}".format(type(dataset[0])))
           if (not isinstance(dataset[0][0], list) and not isinstance(dataset[0][0], tuple)):
               raise TypeError("Le jeu de données n'a pas le bon format. (Need list or " \
41               "tuple, found {}".format(type(dataset[0][0])))

       """
66  À partir d'une phrases annotée, produit les facteurs relatifs à chaque mot
       :return list(features)
       """

       def sentence_to_features(self, sentence):
           result = []
71           for i in range(len(sentence)):
               result.append(self.word_to_features(sentence, i))
           return result

       """
76  Obtient le vecteur de labels pour une phrase annotée
       :list(['0','1'])
       """

       def sentence_to_labels(self, sentence):
           result = [str(item[len(item)-1]) for item in sentence]
81           return result

       """

```

```

Obtient les tokens d'une phrase annotée
: list(str)
86 """
def sentence_to_tokens(self, sentence):
    result = [item[0] for item in sentence]
    return result

91 """
Vérifie les jeux de données d'entraînement passés dans le constructeur et formate
les données pour l'entraînement
"""
def get_training_features(self):
96
    if (self.train_set is None):
        raise ValueError("Aucune donnée d'entraînement n'a été donnée au modèle...")

    self.x_train = [self.sentence_to_features(sent) for sent in self.train_set]
101 self.y_train = [self.sentence_to_labels(sent) for sent in self.train_set]

    return self.x_train, self.y_train

106 """
Vérifie les jeux de données de test passés dans le constructeur et formate les
données pour l'évaluation
"""
def get_test_features(self):
111
    if (self.test_set is None):
        raise ValueError("Aucune donnée de test n'a été donnée au modèle...")

    self.x_test = [self.sentence_to_features(sent) for sent in self.test_set]
116 self.y_test = [self.sentence_to_labels(sent) for sent in self.test_set]

    return self.x_test, self.y_test

121 """
Entraîne le modèle
:return objet crf
"""
def train(self):
126
    self.get_training_features()

    crf = sklearn_crfsuite.CRF(algorithm='lbfgs', min_freq=4, c1=0.1, c2=0.2,
    max_iterations=8000, all_possible_transitions=True, all_possible_states=False)
    crf.fit(self.x_train, self.y_train)
    self.crf = crf
131 return self.crf

136 """
Avec un modèle entraîné, prédit la suite de labels pour une phrase annotée
"""
def make_one_prediction(self, sentence):
141
    if (self.crf is None):
        raise ValueError("Le modèle n'est pas entraîné. Veuillez utiliser la " \
        " fonction train() pour avoir un résultat.")

    features = self.sentence_to_features(sentence)
    return self.crf.predict_single(features)

146 """
Évalue le modèle avec un jeu de données test
:return str
"""
def evaluate(self, test_set=None, verbose=False):
151
    if (test_set is not None):
        self.check_dataset(test_set)
        self.test_set=test_set

    self.get_test_features()

156 labels = list(self.crf.classes_)
    if 'O' in labels:
        labels.remove('O')

161 predicted = self.crf.predict(self.x_test)

import time
results = "Résultats d'évaluation (faite le {}) : \n\n".format(time.strftime("%d/%m/%y"))
results += "Nombre de phrases évaluées : {} \n\n".format(len(self.test_set))
166 results += metrics.flat_classification_report(self.y_test, predicted, labels=labels, digits=3)

mean_cr = 0.0

```

```

171         for vect in predicted:
            mean_cr += self.get_cr(vect)
        mean_cr = mean_cr / len(predicted)

        real_cr = 0.0
        for vect in self.y_test:
            real_cr += self.get_cr(vect)
176        real_cr = real_cr / len(self.y_test)

        results += "\nTaux de compression moyen : {} \t (taux moyen réel : {}) \n".format(
            "%.3f" % mean_cr, "%.3f" % real_cr)

181        if(verbose):

            results += "\nTransitions (de la plus probable à la moins probable) :\n"
            transitions = self.crf.transition_features_

186            transitions_print = []
            for (label_to, label_from), weight in transitions.items():
                elem = label_to, label_from, weight
                transitions_print.append(elem)
            transitions_print.sort(key=lambda item:item[2], reverse=True)
191            for label_to, label_from, weight in transitions_print:
                results += "{} -> {} \t {} \n".format(label_to, label_from, "%.5f" % weight)

            states = self.crf.state_features_
            results += "\nNombre de variables du modèle : {} \n".format(len(states))
196            results += "\nPoids des variables du modèle :\n \n"
            states_print = []
            for (attr, label), weight in states.items():
                elem = attr, label, weight
                states_print.append(elem)

201            states_print.sort(key=lambda item:item[2], reverse=True)
            for attr, label, weight in states_print:
                results += "{} {} \t {} \n".format("%.5f" % weight, label, attr)

206        return results

    """
    Obtient le taux de compression d'une phrase
    :return float
    """
211    @classmethod
    def get_cr(cls, vect):
        nb_keep = 0
        for tok in vect:
176            label = tok[len(tok) - 1]
            if label == '1' or label == 1:
                nb_keep += 1

        return nb_keep / len(vect)

221    """
    Associe un modèle crf stocké dans un fichier pickle compressé au modèle courant
    :param chemin vers un fichier compressé contenant un objet crf
    :return l'objet crf décompressé
    """
226    def set_model_from_pickle(self, pick_file):
        self.crf = u.load_data_pickle(pick_file)
        return self.crf

231    """
    Sauvegarde le modèle crf entraîné
    :param un chemin vers un fichier
    """
236    def save_model(self, out):

        if(self.crf is None):
            raise ValueError("Vous essayez de sauvegarder un modèle crf non entraîné. " \
                "Veuillez d'abord utiliser la fonction train() avant de " \
                "sauvegarder le modèle.")

241        u.write_pickle(self.crf, out)

    """
    Fonctions relatives aux facteurs (features) du modèle
    """
246    def word_to_features(self, sentence, i):

        word = sentence[i][0]
        pos_tag = sentence[i][1]
251        """ ner_tag = sentence[i][2]
        parent_node = sentence[i][3]
        depth = sentence[i][4]

```

```

256     cf = sentence[i][5]
        len_const = sentence[i][6]
        is_sbar = sentence[i][7] """

# Traits (features) sur lesquels le modèle base ses décisions
features = {
261     'position relative': self.get_rel_position(i, len(sentence)),
        'commence par majuscule': word[0].isupper(),
        'est mot vide': self.is_stopword(word),
        '#est mot inconnu (hors de l\'index)': self.get_lemma(word) == 'UNK',
        'pos_tag': pos_tag,
266     '#ner': ner_tag,
        '#noeud parent': parent_node,
        '#profondeur dans arbre synt.': depth,
        'longueur de la phrase > 30': len(sentence) >= 30,
        'longueur de la phrase >= 15 < 30': len(sentence) >= 15 and len(sentence) < 30,
271     'longueur de la phrase < 15': len(sentence) < 15,
        '#mot très fréquent': self.get_freq(word) > 4.0,
        '#mot peu fréquent': self.get_freq(word) < 0.01,
        '#mot à fréquence moyenne': self.get_freq(word) >= 0.01 and
            #self.get_freq(word) < 4,
276     '#fréquence de référence': str(self.get_freq(word)),
        '#longueur constituant': len_const,
        '#est dans P2rel ou sub': is_sbar
        'lemme': self.get_lemma(word)
}

281     """ if (len(cf) != 0):
            features.update(cf) """

# S'il y a un mot précédent
286     if i > 0 :

        previous_word = sentence[i-1][0]
        previous_tag = sentence[i-1][1]
        #previous_ner_tag = sentence[i-1][2]
        #previous_pa = sentence[i-1][3]
291     #previous_depth = sentence[i-1][4]
        #previous_cf = sentence[i-1][5]
        #previous_len_const = sentence[i-1][6]
        #previous_is_sbar = sentence[i-1][7]

296     new_features = {
        'commence par majuscule i-1': previous_word[0].isupper(),
        'pos_tag i-1': previous_tag,
        '#ner i-1': previous_ner_tag,
        '#noeud parent i-1': previous_pa,
301     '#noeud parent i-1 idem mot courant': previous_pa == parent_node,
        '#profondeur arbre synt. précédent': previous_depth,
        '#longueur constituant i-1': previous_len_const,
        '#est dans P2 i-1': previous_is_sbar
        'lemme i-1': self.get_lemma(previous_word)
306     }
        features.update(new_features)

        """ if len(previous_cf) > 0:
            feat = {}
311         for key, val in previous_cf.items():
                new_key = key + ' i - 1'
                feat[new_key] = val
                features.update(feat) """

316     else:
        features['premier token'] = True

# S'il y a un mot suivant
321     if i < len(sentence) - 1:

        next_word = sentence[i+1][0]
        next_tag = sentence[i+1][1]
        """ next_ner_tag = sentence[i+1][2]
        next_pa = sentence[i+1][3]
326     next_depth = sentence[i+1][4]
        next_cf = sentence[i+1][5]
        next_len_const = sentence[i+1][6]
        next_is_sbar = sentence[i+1][7] """

331     new_features = {
        'commence par majuscule i+1': next_word[0].isupper(),
        'pos tag i+1': next_tag,
        '#ner i+1': next_ner_tag,
        '#noeud parent i+1': next_pa,
336     '#noeud parent i+1 idem mot courant': next_pa == parent_node,
        '#profondeur abre synt. i+1': next_depth,
        '#longueur constituant i+1': next_len_const,

```

```

    # 'est dans P2 i+1': next_is_sbar
    'lemme i + 1': self.get_lemma(next_word)
341 }
    features.update(new_features)

    """ if len(next_cf) > 0:
        feat = {}
346         for key, val in next_cf.items():
            new_key = key + ' i + 1'
            feat[new_key] = val
            features.update(feat) """

351 else:
    features['dernier token'] = True

# Tri-grams
if i > 1:
356     previous_word = sentence[i-2][0]
    previous_tag = sentence[i-2][1]
    """ previous_ner_tag = sentence[i-2][2]
    previous_pa = sentence[i-2][3]
    previous_depth = sentence[i-2][4]
361     previous_cf = sentence[i-2][5]
    previous_len_const = sentence[i-2][6]
    previous_is_sbar = sentence[i-2][7] """

    new_features = {
366         'commence par majuscule i-2': previous_word[0].isupper(),
        'pos_tag i-2': previous_tag,
        #'ner mot i-2': previous_ner_tag,
        #'noeud parent i-2': previous_pa,
        #'profondeur arbre synt. i-2': previous_depth,
371         #'longueur constituant i-2': previous_len_const,
        #'est dans P2 i-2': previous_is_sbar
        'lemme i-2': self.get_lemma(previous_word)
    }
    features.update(new_features)

376     """ if len(previous_cf) > 0:
        feat = {}
        for key, val in previous_cf.items():
            new_key = key + ' i - 1'
381             feat[new_key] = val
            features.update(feat) """

    if i < len(sentence) - 2:

386         next_word = sentence[i + 2][0]
        next_tag = sentence[i + 2][1]
        """ next_ner_tag = sentence[i + 2][2]
        next_pa = sentence[i + 2][3]
        next_depth = sentence[i + 2][4]
391         next_cf = sentence[i + 2][5]
        next_len_const = sentence[i + 2][6]
        next_is_sbar = sentence[i + 2][7] """

        new_features = {
396             'commence par majuscule mot i+2': next_word[0].isupper(),
            'post_tag i+2': next_tag,
            #'ner mot i+2': next_ner_tag,
            #'noeud parent i+2': next_pa,
            #'profondeur arbre synt. i+2': next_depth,
401             #'longueur constituant i+2': next_len_const,
            #'est dans P2 i+2': next_is_sbar
            'lemme i+2': self.get_lemma(next_word)
        }

406         features.update(new_features)

        """ if len(next_cf) > 0:
            feat = {}
            for key, val in next_cf.items():
                new_key = key + ' i + 2'
411                feat[new_key] = val
                features.update(feat) """

# 4-grams
if i > 2:
416     previous_word = sentence[i - 3][0]
    previous_tag = sentence[i - 3][1]
    """ previous_ner_tag = sentence[i - 3][2]
    previous_pa = sentence[i - 3][3]
    previous_depth = sentence[i - 3][4]
421     previous_cf = sentence[i - 3][5]
    previous_len_const = sentence[i - 3][6]

```

```

previous_is_sbar = sentence[i - 3][7]"""

426 new_features = {
    'commence par majuscule i-3': previous_word[0].isupper(),
    'pos_tag i-3': previous_tag,
    #'ner mot i-3': previous_ner_tag,
    #'noeud parent i-3': previous_pa,
431 #'profondeur arbre synt. i-3': previous_depth,
    #'longueur constituant i-3': previous_len_const,
    #'est dans P2 i-3': previous_is_sbar
    'lemme i-3': self.get_lemma(previous_word)
}
436 features.update(new_features)

    """if len(previous_cf) > 0:
        feat = {}
        for key, val in previous_cf.items():
441             new_key = key + ' i - 3'
            feat[new_key] = val
        features.update(feat)"""

if i < len(sentence) - 3:
446     next_word = sentence[i + 3][0]
    next_tag = sentence[i + 3][1]
    """next_ner_tag = sentence[i + 3][2]
    next_pa = sentence[i + 3][3]
    next_depth = sentence[i + 3][4]
451 next_cf = sentence[i + 3][5]
    next_len_const = sentence[i + 3][6]
    next_is_sbar = sentence[i + 3][7]"""

    new_features = {
456     'commence par majuscule mot i+3': next_word[0].isupper(),
    'pos_tag i+3': next_tag,
    #'ner mot i+3': next_ner_tag,
    #'noeud parent i+3': next_pa,
    #'profondeur arbre synt. i+3': next_depth,
461 #'longueur constituant i+3': next_len_const,
    #'est dans P2 i+3': next_is_sbar
    'lemme i+3': next_tag
    }
    features.update(new_features)

    """if len(next_cf) > 0:
        feat = {}
        for key, val in next_cf.items():
471             new_key = key + ' i + 2'
            feat[new_key] = val
        features.update(feat)"""

# 5-grams
if i > 4:
476     previous_word = sentence[i - 4][0]
    previous_tag = sentence[i - 4][1]

    new_features = {
    'commence par majuscule i-4': previous_word[0].isupper(),
481 'pos_tag i-4': previous_tag,
    'lemme i-4': self.get_lemma(previous_word)
    }
    features.update(new_features)

if i < len(sentence) - 4:
486     next_word = sentence[i + 4][0]
    next_tag = sentence[i + 4][1]

    new_features = {
491     'commence par majuscule i+4': next_word[0].isupper(),
    'pos_tag i+4': next_tag,
    'lemme i+4': self.get_lemma(next_word)
    }
    features.update(new_features)
496

return features

@staticmethod
501 def get_rel_position(index, length):
    decimal = index / length
    return decimal

@classmethod
506 def is_stopword(cls, word):
    return word.lower() in cls.stopwords

```

```

511     @staticmethod
    def is_ner(ner_tag):
        return ner_tag != 'O'

    @classmethod
    def get_freq(cls, word):
        key = word.lower()
        if key not in cls.index:
            return 0.0
        freq = cls.index[key][1]
        return freq

521     @classmethod
    def get_lemma(cls, word):
        key = word.lower()
        if key not in cls.index:
            return 'UNK'
526         return cls.index[key][0]

```

A.6.3 sentence_parser.py

```

#!/ etinfo/users/2018/chelala/env_memoire/bin/python
# -*- coding: utf-8 -*-

4 """
Classe statique qui permet d'annoter (tokenisation, parsing, ner...) des phrases.
Utilise les fonctionnalités de Stanford Core NLP pour tokeniser, taguer et étiqueter,
via le module stanfordcorenlp. Utilise également une version locale du parser de Stanford
(via le module nltk), plus performante pour les longues phrases en français.

9 @author Jeremy Chelala
@date juin 2018

"""

14 import os.path
import re

from stanfordcorenlp import StanfordCoreNLP
19 import nltk.tree
import nltk.parse.stanford as SFparser
import utils as u

class SentenceParser():
24     'Classe SentenceParser'

    # Variables de classes, utilisées comme valeur par défaut lors de l'initialisation
    stanfordcorenlp_directory = '~/StanfordNLP/stanford-corenlp-full-2018-02-27'
29     lang_default = 'fr'
    parser_file = 'edu/stanford/nlp/models/lexparser/frenchFactored.ser.gz' # pour le
                                     # parsing en local uniquement

    def __init__(self, stanford_path=os.path.expanduser(stanfordcorenlp_directory),
34         lang=lang_default):
        self.stanford_dir = stanford_path
        self.lang = lang

    """
39     Sépare une phrase donnée en une liste de mots (tokenisation)
    :returns list
    """
    def tokenize(self, sentence):
44         with StanfordCoreNLP(self.stanford_dir, lang=self.lang, memory='1g') as nlp:
            tokens = nlp.word_tokenize(sentence)

            return tokens

49     """
    Tokenise et tague une phrase données
    :returns list( tuple(token, tag) )
    """
    def pos_tag(self, sentence):
54         with StanfordCoreNLP(self.stanford_dir, lang=self.lang, memory='3g') as nlp:
            result = nlp.pos_tag(sentence)
            return result

59     """
    Parse une phrase passée en argument
    :returns l'arbre syntaxique de la phrase sous forme de string

```

```

"""
64 def parse(self, sentence):
    with StanfordCoreNLP(self.stanford_dir, lang=self.lang, timeout=10000,
        memory='6g') as nlp:
        result = nlp.parse(sentence)
69     return result

"""
Tokenise, tague et parse une phrase donnée en argument
:returns tuple( list(tokens), list(tags), tree )
"""
74 def full_parse(self, sentence):

    tree_str = self.parse(sentence)

79     # Transforme le string tree en objet arbre
    tree = nltk.tree.Tree.fromstring(tree_str)
    tokens_tags = tree.pos()

    tokens = [token for token, pos in tokens_tags]
    tags = [pos for token, pos in tokens_tags]
84     return tokens, tags, tree_str

"""
Ibid., mais avec le parser de Stanford local (pas "Stanford CoreNLP").
"""
89 def local_full_parse(self, sentence):

    parser = SFPARSER.StanfordParser(model_path=SentenceParser.parser_file)
    tokens_tags = self.pos_tag(sentence)
94     tree = parser.tagged_parse(tokens_tags)

    tokens = [token for token, tag in tokens_tags]
    tags = [pos for token, pos in tokens_tags]
99     return tokens, tags, str(tree.__next__())

"""
Donne les entités nommées présentes dans une phrase et les catégorise
:returns
"""
104 def ner_parse(self, sentence):

    with StanfordCoreNLP(self.stanford_dir, lang=self.lang) as nlp:
        result = nlp.ner(sentence)
109     return result

"""
Donne le tag du noeud père étant donné la position d'un mot dans un arbre syntaxique.
La position 0 correspond au premier mot de la phrase, n-1 au dernier.
:param un arbre syntaxique au format 'str' (string) tel que donné par le parser de
Stanford et la position du mot
:returns le tag du noeud parent (str)
"""
114 @classmethod
def get_parent_node(cls, tree, token_nb):

    # Transforme l'arbre en objet tree avec parenté
124     ptree = cls.__get_parented_tree(tree)

    # Compteur de mots
    current_tok_nb = -1

    for sbt in ptree.subtrees():
129         if (sbt.height() == 2):
            current_tok_nb += 1
            if (token_nb == current_tok_nb):

                if (sbt.parent() is None):
134                     return None

                return sbt.parent().label()

"""
139 Renvoie True si le mot à la position i a un noeud nommé 'node' dans sa parenté
:param un arbre syntaxique au format 'str' (string) tel que donné par le parser
de Stanford, la position du mot et le label du tag à repérer
:returns bool
"""
144 @classmethod
def is_child_of(cls, tree, index, node):

```

```

# Transforme l'arbre en objet tree avec parenté
ptree = cls.__get_parented_tree(tree)

149 for i, sbt in enumerate(ptree.subtrees(lambda sbt:sbt.height() == 2)):
    if i == index:
        while sbt.parent().label() != 'SENT':
            if sbt.parent().label() == node:
154                 return True
            sbt = sbt.parent()

        return False

159 """
Donne le nombre de tokens faisant partie du même constituant syntaxique que
le mot à la position i
:param un arbre syntaxique et la position du token
:return int
164 """
@classmethod
def get_len_constituent(cls, tree, index):

    ptree = cls.__get_parented_tree(tree)

    count_pos = -1
    for sbt in ptree.subtrees():

        if sbt.height() == 2:
174             count_pos += 1

        if count_pos == index:
            if sbt.parent().label() == 'SENT':
                return 1
179             return len(sbt.parent().leaves())

    """
Obtient la hauteur de l'arbre depuis la racine jusqu'au mot à la position donnée
:param un arbre syntaxique au format 'str' et la position du mot
184 :returns int
    """
@classmethod
def get_depth(cls, tree, index):

    ptree = cls.__get_parented_tree(tree)
    count_token = -1

    for sbt in ptree.subtrees():

        if (sbt.height() == 2):
194             count_token += 1

        if (count_token == index):
            count = 1
            current = sbt.parent()
            while (current.label() != 'SENT'):
                count += 1
                current = current.parent()
199             return count

    """
Obtient les features "chunk" pour un mot à l'index donné dans un arbre syntaxique
:param un arbres syntaxique et la positin du mot
209 :returns dict( keys:nom_feature, val: [0,1]) (vide si aucun feature)
    """
@classmethod
def get_chunk_features(cls, tree, index):

    features = {}
    ptree = cls.__get_parented_tree(tree)

    current_tok_nb = -1
    for sbt in ptree.subtrees():
        if (sbt.height() == 2):
            current_tok_nb += 1
            sbt[0] = sbt[0] + "-{}".format(current_tok_nb)
219             if (current_tok_nb == index):

                current_tree = sbt.parent()
                while (current_tree.parent() is not None and current_tree.label() != 'SENT'):
224                     for i, child in enumerate(current_tree.subtrees()):
                         if (child.height() == 2 and child == sbt):
                             if (i == 1):
229                                 key = "B-{}".format(current_tree.label())
                             else:
                                 key = "I-{}".format(current_tree.label())

```

```

234         if key in features.keys():
            features[key] += 1
        else:
            features[key] = 1
        break

239     current_tree = current_tree.parent()

    return features

"""
244 Annote une phrase donnée seule (sans les labels donc)
:returns list( tuple(token, tag, parent_node) )
"""
def annotate_sent(self, sent):

249     return self.pos_tag(sent)

"""
254 Pour un couple de phrases (phrase originale-compression), attribue un label [1,0] à
chaque mot de la phrase originale. 1 indique que le mot a été conservé dans la
compression. Par défaut, la méthode considère que les phrases sont tokenisées.
:returns list( [0,1] )
"""
def labelize(self, sent, comp, from_str=False):

259     vector = []

    if (from_str):
        tokens_or = self.tokenize(sent)
        tokens_comp = self.tokenize(comp)
264     else:
        tokens_or = sent
        tokens_comp = comp

    if (len(tokens_or) == len(tokens_comp)):
        for i in range(len(tokens_or)):
            vector.append(1)
        return vector

    index_comparison = 0
    for token in tokens_or:

274         if (index_comparison >= len(tokens_comp)):
            vector.append(0)
        else:
            current_or = token.lower()
            current_comp = tokens_comp[index_comparison].lower()

            if (current_or == current_comp):
                vector.append(1)
                index_comparison += 1
            else:
                vector.append(0)

    return vector

"""
289 Transforme un vecteur de labels en une phrase tokenisée ne reprenant que les mots indiqués par le
vecteur
:param vecteur de labels, tokens phrase originale
:returns list(tokens)
"""
294 def labels_to_tokens(self, vector, tokens_or):
    res = []
    for label, token in zip(vector, tokens_or):
        if str(label) == '0' or str(label) == '1':
            res.append(token)
299     return res

"""
304 Reconstitue une phrase à partir d'une liste de tokens. Peut parfois donner un résultat
légèrement de l'originale.
:returns str
"""
def untokenize(self, tokens):

309     res = u.array_to_txt(tokens, join_elem=' ')

    res = re.sub("à le ", "au ", res)
    res = re.sub("à les ", "aux ", res)
    res = re.sub("de le ", "du ", res)
    res = re.sub("de les ", "des ", res)
314

```

```

319     res = re.sub(" ?- ?", "-", res)
    res = re.sub("\\" (.+?) \\", "\g<1>", res)
    res = re.sub(" ' (.+?) '", "\g<1>", res)
    res = re.sub("\. ", ".", res)
    res = re.sub(", ", ",", res)
    res = re.sub("([ldujtsmcnLDJTSMCNU])' ", "\g<1>", res)

324     for i in range(len(res)):
        if(res[i].isalpha()):
            if(i < len(res) - 1):
329                 res = res[:i] + res[i].upper() + res[i+1:]
            else:
                res = res[:i] + res[i].upper()
                break

    return res

334 """
    Transforme un arbre syntaxique de type str en objet ParentedTree (nltk)
    :param str
    :returns ParentedTree
339 """
    @staticmethod
    def __get_parented_tree(tree):
        # Change les parenthèse de noeuds en accolades ('[]'), pour éviter les ambiguïtés avec les
344         tokens '(' et ')'
        new_tree = re.sub("\([([ ^ ])\)", "{\g<1>", tree)
        new_tree = re.sub("\)", "}", new_tree)
        new_tree = re.sub("PUNCT \\", "PUNCT )", new_tree)

        # Transforme l'arbre en objet tree avec parenté
349         return nltk.tree.ParentedTree.fromstring(new_tree, brackets='{ }')

```

A.6.4 utils.py

```

1  #! /etinfo/users/2018/chelala/env_memoire/bin/python
   # -*- coding: utf-8 -*-

   """
   Script contenant des fonctions utilitaires d'écriture et de lectures de données dans des
6   fichiers txt, pickle et gunzip.

   @author Jeremy Chelala
   @date juin 2018

11  """
   import os
   import gzip
   import pickle

16  def write_txt(string, out):
       os.makedirs(os.path.dirname(out), exist_ok=True)
       with open(out, "w+") as fh:
           fh.write(string)

21  def append_txt(string, out):
       os.makedirs(os.path.dirname(out), exist_ok=True)
       with open(out, "a") as fh:
           fh.write(string)

26  def write_gzip(string, out):
       os.makedirs(os.path.dirname(out), exist_ok=True)
       with gzip.open(out, "wb+") as fh:
           content = string.encode('utf-8')
           fh.write(content)

31  def write_pickle(donnees, out):
       os.makedirs(os.path.dirname(out), exist_ok=True)
       with open(out, "wb") as fh:
           pickle.dump(donnees, fh)

36  def get_lines(file):
       with open(file, "r") as fh:
           lines = fh.readlines()
       return lines

41  def array_to_txt(array, join_elem="\n"):
       result = ""
       for elem in array:

```

```
        result += str(elem) + join_elem
46     return result

def list_equals(cand, ref):
    if len(cand) != len(ref):
51         return False

    for i, j in zip(cand, ref):
        if i != j:
            return False
56     return True

def load_data_pickle(pic_file):
    with open(pic_file, "rb") as fh:
        data = pickle.load(fh)
61     return data
```