Normalisation orthographique de corpus bruités

Marion Baranes

Université Paris Diderot – Sorbonne Paris Cité
École doctorale de Sciences du Langage n° 132
UFR Linguistique
Laboratoire Alpage (Inria – Université Paris Diderot)
Entreprise viavoo

Doctorat de Linguistique Théorique, Descriptive et Automatique

Marion Baranes

Normalisation orthographique de corpus bruités

Thèse dirigée par : **Laurence Danlos et Benoît Sagot**

Soutenue publiquement le 23 Octobre 2015

Composition du jury :

Dr	Delphine Bernhard	Université de Strasbourg	(Examinatrice)
Pr.	Laurence Danlos	Université Paris-Diderot (Paris 7)	(Directrice)
Pr.	Cédrick Fairon	Université Catholique de Louvain	(Rapporteur)
Pr.	Philippe Langlais	Université de Montréal (UdeM)	(Rapporteur)
Dr	Benoît Sagot	Inria	(Co-Directeur)

Remerciements

Je tiens tout d'abord à remercier Thierry Desforges, directeur de l'entreprise viavoo, d'avoir rendu possible la réalisation de ce travail et de m'avoir accordé sa confiance.

Un immense merci à Benoît Sagot qui a clairement enrichi la définition du mot *encadrer* dans le contexte de cette thèse. Je salue notamment l'intérêt constant qu'il a porté à ces travaux, sa disponibilité et ses conseils méticuleux.

Je remercie Cédrick Fairon et Philippe Langlais qui m'ont fait l'honneur d'être rapporteurs de cette thèse et qui m'ont fait part de leurs commentaires précieux. Merci aussi à Delphine Bernhard qui a accepté de faire partie du Jury de cette soutenance.

J'adresse une pensée particulière à Geoffrey Doucy qui m'a patiemment encadré au sein de viavoo pendant mes premières années de thèse. Sans lui cette thèse n'aurait jamais eu lieu. Merci aussi à Sébastien Louvet qui a su reprendre ce suivi avec justesse et qui a eu foi en mes délais parfois légèrement utopiques.
J'ai eu la chance chez viavoo d'être entourée et encouragée, merci donc à mes collègues et ex-collègues et notamment à Anne-Charlotte, Aude, Goshia, Jeanne, Laurent, Stav. et Taoufik qui m'ont chacun prêté une oreille attentive et aidé, à leur manière, quand j'en avais besoin.

Merci à Laurence Danlos et à tous les membres d'Alpage qui ont représenté pour moi de véritables bouffées d'oxygène. Travailler dans ce laboratoire m'a notamment permis de rencontrer Virginie Mouilleron, Damien Nouvel et Djamé Seddah dont les conseils m'ont été précieux.
Ces mois passés sur cette thèse n'auraient jamais été les mêmes sans les doctorants et ex-doctorants du labo : Charlotte, Chloé, Corentin, Emmanuel, Enrique, Juliette, Luc, Marianne, Maximin, Pierre, Rosa et Sarah. Des personnes rares qui ont su ensoleiller mes journées.

Je remercie Valérie d'avoir été aux petits soins avec moi quand je me consumais à petit feu et pour son admirable talent à me rendre le sourire en toutes circonstances.

De manière plus générale, merci à mes amis et à ma famille. J'ai une chance incroyable de vous avoir et d'être si bien entourée.

Je remercie enfin mes parents et mon frère pour être ce qu'ils sont et pour leur soutien sans faille. Merci à ma mère qui a eu le courage de lire l'intégralité de cette thèse dans le seul but d'en évincer les coquilles.

Table des matières

Résumé

L'information contenue dans les messages publiés par les internautes (forums, réseaux sociaux, sites d'avis, etc.) comporte un intérêt stratégique pour de nombreuses entreprises. Néanmoins, peu d'outils ont été conçus pour faciliter l'analyse de ces messages, dont l'orthographe, la typographie et la syntaxe sont souvent bruitées.
Cette thèse industrielle a été réalisée au sein de l'entreprise viavoo afin d'améliorer les résultats d'un outil d'extraction d'information qui fait abstraction de la variabilité flexionnelle. Nous avons ainsi développé une chaîne de traitements pour la normalisation orthographique de textes bruités. Son objectif est donc de transformer ces textes pour faire en sorte que tous les mots qui les composent obtiennent une orthographe standard, à la flexion près.
L'approche présentée ici consiste tout d'abord à déterminer automatiquement, parmi les tokens du corpus traité qui sont inconnus d'un lexique de référence, ceux qui résultent d'*altérations* et qu'il conviendrait donc de normaliser, par opposition aux autres (néologismes, emprunts, etc.). Des candidats de normalisation sont alors proposés pour ces tokens à l'aide de règles pondérées obtenues par des techniques d'apprentissage par analogie. Nous identifions ensuite des tokens connus du lexique de référence mais qui résultent néanmoins d'une altération (*fautes grammaticales*), et proposons des candidats de normalisation pour ces tokens. Enfin, des modèles de langue permettent de prendre en compte le contexte dans lequel apparaissent les différents types d'altérations pour lesquels des candidats de normalisation ont été proposés afin de choisir les plus probables. Différentes expériences et évaluations sont réalisées sur le français à chaque étape et sur la chaîne complète. Une attention particulière a été portée au caractère faiblement dépendant de la langue des modules développés, ce qui permet d'envisager son adaptation à d'autres langues européennes.

Mots-clefs : Normalisation, correction orthographique, mots inconnus, altérations, données produites par l'utilisateur.

Abstract

The information contained in messages posted on the Internet (forums, social networks, review sites...) is of strategic importance for many companies. However, few tools have been designed for analysing such messages, the spelling, typography and syntax of which are often noisy.
This industrial PhD thesis has been carried out within the viavoo company with the aim of improving the results of a lemma-based information retrieval tool. We have developed a processing pipeline for the normalisation of noisy texts. Its aim is to ensure that each word is assigned the standard spelling corresponding to one of its lemma's inflected forms.
First, among all tokens of the corpus that are unknown to a reference lexicon, we automatically determine which ones result from *alterations* — and therefore should be normalised — as opposed to those that do not (neologisms, loanwords...). Normalisation candidates are then generated for these tokens using weighted rules obtained by analogy-based machine learning techniques. Next we identify tokens that are known to the reference lexicon but are nevertheless the result of an alteration (*grammatical errors*), and generate normalisation candidates for each of them. Finally, language models allow us to perform a context-sensitive disambiguation of the normalisation candidates generated for all types of alterations. Numerous experiments and evaluations are carried out on French data for each module and for the overall pipeline. Special attention has been paid to keep all modules as language-independent as possible, which paves the way for future adaptations of our pipeline to other European languages.

Keywords : Normalization, Spell-checking, unknown-word, spelling mistake, User-Generated Content, Natural Language Processing.

Introduction générale

Problématique

Les tâches liées au traitement automatique des langues comme la classification, la traduction automatique ou l'extraction d'informations intéressent actuellement de plus en plus les entreprises. La majorité de ces outils s'appuient sur des corpus relativement propres. Si une personne choisit de travailler sur des données plus altérées, rédigées sur internet, sa tâche se complexifie. C'est pourquoi il est important de nettoyer ces textes afin de pouvoir appliquer par la suite les traitements souhaités. Être capable de normaliser ou de corriger automatiquement devient ainsi un réel besoin.

La correction et la normalisation de textes ne sont pas des sujets nouveaux (à titre d'exemple nous pouvons citer Damerau (1964) et Levenshtein (1966)) et ont déjà fait l'objet de nombreuses recherches comme nous le verrons au chapitre 3.3. Toutefois, les outils existants à l'heure actuelle ne répondent que partiellement à ce problème. La grande majorité des travaux réalisés sur la normalisation se concentrent sur les messages texto. Parallèlement, les études sur la correction traitent principalement les textes de qualité journalistique. Ces derniers sont donc rarement adaptés aux types de textes que l'on peut retrouver sur internet, de qualité parfois très dégradée, et qui sont pourtant ceux que l'on veut analyser lorsque l'on traite des données réelles dites « produites par l'utilisateur » (*User-Generated Content*). Ces données peuvent provenir de différents canaux tels que les blogs, les forums ou les réseaux sociaux.

Notre objectif est de combler ce manque, en développant un système dédié à la normalisation orthographique automatisée de ces données réelles. Le principal intérêt d'une tâche comme celle-ci est de simplifier et d'améliorer l'analyse d'un texte bruité par des outils de traitement automatique de la langue conçu avant tout pour le traitement des textes édités.

Contexte

L'analyse des données produites par l'utilisateur est une problématique qui suscite un intérêt croissant auprès des entreprises. En effet, de telles analyses leur permettraient par exemple de connaître l'avis de leurs consommateurs sur leurs prestations. La quantité importante de ces messages rend cette tâche difficilement réalisable manuellement. Afin de pallier ce problème, certaines entreprises proposent d'analyser automatiquement ces messages, bien que la qualité orthographique de ces derniers pose toujours problème.

La thèse décrite ici est une thèse industrielle. Elle a été réalisée au sein de viavoo, une société spécialisée dans l'analyse de l'expérience client. Cette société propose aux entreprises d'analyser leurs retours clients et d'en extraire les principales informations.

Ces retours client se présentent sous la forme de verbatim produits par des consommateurs par le biais d'internet. Ils peuvent porter sur de multiples sujets. En effet, en fonction de l'entreprise concernée, un consommateur pourra aussi bien parler d'un produit de beauté, du confort d'un de ses voyages, d'un problème de réseau immédiat ou encore de la prestation d'un agent commercial lors d'un de ses achats. En outre, ces messages peuvent provenir de divers canaux tels que les réseaux sociaux, les mails, les forums, les sites d'avis ou les enquêtes de satisfaction. La diversité de ces corpus reflète l'hétérogénéité de leur contenu. Ainsi, un locuteur pourra aussi bien parler à la première personne qu'à la troisième en s'adressant ou non à quelqu'un. Le message sera par ailleurs plus ou moins soigné en fonction de la personne à laquelle il s'adresse et du canal utilisé. Dans un forum par exemple, un message sera probablement plus court et écrit de manière plus familière que dans un mail s'adressant à un service client. L'entreprise est ainsi amenée à étudier des textes de taille, de nature, de thème et de qualité très variables.

Le système d'extraction d'informations mis en place par viavoo s'appuie sur des grammaires locales et intègre à différents niveaux une analyse lexicale, syntaxique et sémantique. Cette analyse du texte, bien que réalisée à la flexion près, est peu permissive. Si elle peut passer outre des fautes d'accord dans un texte, elle sera gênée par tout autre type d'altérations [1] que peut subir un mot. Les informations contenues dans ce message seront alors plus complexes à détecter et à analyser. Ainsi comme nous pouvons le constater dans l'énoncé « ***ya*** *pas un problème de* ***rayso ver*** *Paris ?* », la présence d'une altération peut empêcher la détection des informations. En effet, un système s'arrêtant sur l'orthographe ne pourra comprendre et catégoriser la nature du problème soulevé par l'auteur de cet énoncé. La présence de mots altérés peut ainsi avoir de sérieuses conséquences sur les résultats obtenus par les solutions proposées par viavoo. Être capable de détecter

1. Nous faisons dans ce travail la distinction entre un mot altéré et une faute d'orthographe. Une faute d'orthographe étant un type d'altération tout comme peut l'être par exemple une abréviation. Nous reviendrons sur ces notions dans les chapitres à venir.

ces altérations afin de les remplacer par leur forme normalisée est ainsi nécessaire.

Une approche intuitive serait de mettre en place un système de correction adapté. Toutefois, la solution intégrée par viavoo travaillant à la flexion près, un système moins *strict* semble être un meilleur compromis. Par ailleurs, les systèmes de correction se concentrent usuellement sur les fautes lexicales ou grammaticales, ce qui est légèrement restrictif si on s'intéresse aux altérations de manière plus générale.

Nous proposons ainsi dans cette thèse d'implémenter un système capable de traiter tous les mots altérés, que ces derniers correspondent à des mots connus ou non des lexiques utilisés. Actuellement, rares sont les systèmes qui tentent de traiter à la fois les altérations donnant lieu à des mots connus et celles produisant des mots inconnus (voir toutefois Carlson et Fette (2007)). La majorité des études se concentrent souvent sur un seul type d'altération (fautes lexicales, fautes grammaticales ou encore langage texto). Dans la mesure où il est primordial que notre système fonctionne quel que soit le canal, il est nécessaire qu'il ne se limite pas à un type de texte ou à une qualité orthographique donnée. En effet, un message provenant d'un réseau social tel que Facebook aura de très fortes chances d'être moins bien orthographié qu'un autre posté sur un site d'avis. Les altérations qu'il contiendra ne seront par ailleurs pas forcément de la même nature.

Dans la mesure où ce système sera appliqué à terme aux textes traités par viavoo, il est important qu'il soit très peu bruyant. En effet, un système qui réaliserait une sur-normalisation du texte traité risquerait, en plus d'altérer le texte, d'induire en erreur l'analyse proposée par viavoo. Par exemple, une normalisation inutile et erronée pourrait permettre la détection d'informations initialement absentes du texte et ainsi faussées, ou entraîner la perte d'une information qui était initialement détectée. Bien que la sous-détection puisse être gênante, il est moins grave d'omettre une information à cause d'un mot altéré que d'en inventer ou d'en supprimer une. Nous favoriserons ainsi la précision au rappel tout au long de ce travail.

Enfin, précisons que le système mis en place se doit d'être indépendant de la langue. Pour l'instant, viavoo ne traite pleinement que le français mais analyse déjà, de façon plus limitée, de nombreuses langues européennes. Nous souhaitons par conséquent concevoir un système de normalisation automatique qui, en plus d'être apte à traiter les textes en français, serait aussi facilement adaptable à d'autres langues. Le caractère multilingue que peut avoir un correcteur n'apparaît que très peu dans la littérature (Reynaert, 2004).

Structure de la thèse

Cette thèse est structurée en deux parties. La première, composée de trois chapitres, propose un état de l'art des différents domaines que nous abordons dans notre travail. La seconde partie, dotée elle aussi de 3 chapitres, détaille la chaîne de traitement mise en place afin de répondre à notre problématique. Elle est suivie d'un chapitre concluant cette thèse et présentant ses perspectives.

Avant de rentrer dans le cœur de notre sujet, il est important de redéfinir et de comprendre les notions que nous voulons manipuler. L'altération d'un mot peut aussi bien donner lieu à un mot connu qu'inconnu d'un lexique de référence. Le **chapitre 1** a donc pour principal objectif d'appréhender la notion de *mot inconnu*. On commencera pour cela par s'interroger sur la notion de mot et de lexique avant de nous concentrer sur les mots inconnus. Une classification de ces derniers sera proposée au sein de ce chapitre.

Les mots inconnus correspondant ainsi à plusieurs catégories de mots, leur détection au sein d'un corpus peut être réalisée au cas par cas. Différentes façons de détecter automatiquement ces inconnus peuvent être mises en place quand ces derniers ne correspondent pas à des mots altérés. Le **chapitre 2** introduira ces méthodes avant de se concentrer plus particulièrement sur une technique d'apprentissage qui a particulièrement retenue notre attention : l'apprentissage par analogie. Nous reviendrons par conséquent sur cette notion et sur ses applications en traitement automatique de la langue.

Les altérations, mises de côté jusqu'ici puisqu'elles peuvent aussi bien produire des mots connus qu'inconnus, font l'objet du **chapitre 3**. Ce dernier se focalise sur l'opération même d'altération d'un mot. Il s'interroge notamment sur ses origines, son mécanisme ou encore sur les différents tokens pouvant être obtenus à l'issue d'une altération. Puis il décrit les différentes méthodes pouvant être utilisées pour traiter une altération en s'attardant particulièrement sur les systèmes de normalisation et de correction automatique.

Suite à cette première partie qui donne un aperçu de l'existant, une introduction globale de la seconde partie propose une rapide introduction du système mis en place et de son architecture.

La description de ce système commence avec le **chapitre 4** qui détaille la chaîne de traitement initiale sur laquelle nous nous sommes appuyée. Il décrit ensuite plus précisément les divers prétraitements que nous avons élaboré et auxquels nous soumettons le texte que nous voulons traiter. Ces prétraitements ont deux objectifs : écarter les mots inconnus qui ne correspondent pas à des altérations afin de ne pas les considérer comme telles par la suite et normaliser les mots altérés aisément détectables. Chacun d'entre eux est suivi d'une évaluation afin de vérifier leur utilité au sein de notre chaîne.

Une fois le texte prétraité, une première recherche des altérations détectables hors contexte est alors réalisable. Le **chapitre 5** décrit ainsi la mise en place et l'évaluation d'un système se concentrant sur les mots inconnus encore non traités par notre chaîne. On suppose ici qu'il ne peut alors s'agir que de mots altérés. Ce système propose ainsi de produire des candidats de normalisation dotés chacun d'un poids de confiance. Pour cela, il s'appuie sur un système de règles pondérées induites automatiquement par analogie.

Le **chapitre 6** regroupe deux tâches qui ont en commun qu'elles nécessitent, contrairement au chapitre précédent, des informations contextuelles. La première de ces tâches est la mise en place et l'évaluation d'un système se concentrant sur les mots dont l'altération aura produit un mot connu des lexiques. La seconde est un système de désambiguïsation qui reprend toutes les analyses obtenues précédemment (avec et sans contexte) afin de ne conserver que les plus pertinentes.

Enfin, cette thèse s'achèvera sur une conclusion générale (**chapitre 7**) qui la récapitulera et présentera les perspectives envisageables pour une amélioration et un prolongement de ces travaux.

Première partie

État de l'art

CHAPITRE 1

Les mots inconnus

Sommaire

En traitement automatique des langues, il peut s'avérer complexe d'analyser un texte si certaines des unités qui le composent ne figurent pas dans nos lexiques. Ce problème a par exemple été soulevé en traduction automatique (Gdaniec et al., 2001; Cartoni, 2006; Denoual, 2007). En effet, il est difficile de traduire un mot dans une langue cible si ce mot est déjà inconnu de la langue source. De même, les mots inconnus doivent être pris en considération dans des tâches telles que l'annotation en catégories grammaticales (Mikheev, 1997) ou en classes morphologiques (Nakov et al., 2003). Comme nous le verrons par la suite dans ce chapitre (section 1.3), la probabilité qu'un inconnu corresponde à un néologisme est assez forte. C'est pour cette raison que certaines études se sont intéressées aux questions liées à la classification des mots inconnus (Blancafort et al., 2010) et à la détection de néologismes (Issac, 2011; Sagot et al., 2013; Falk et al., 2014), la détection de néologismes ayant souvent pour objectif d'étendre des lexiques de langue (Dister et Fairon, 2004; Sagot et al., 2013; Falk et al., 2014).

Dans le cadre de ce travail, la présence d'un inconnu peut être problématique. Si un mot est considéré comme inconnu de notre lexique de référence, il nous est impossible de déterminer sans analyse supplémentaire la raison de cette absence. Ce mot peut ainsi être considéré comme tel pour plusieurs raisons : cesdits lexiques sont incomplets, le mot en question vient d'être créé ou encore parce qu'il s'agit d'un mot mal orthographié. Ainsi, il serait très maladroit de tenter de traiter les fautes lexicales par exemple en partant du principe que tout mot inconnu ne peut qu'être le résultat d'une erreur. Cela pourrait notamment donner lieu à de nombreuses surcorrections.

1.1 Du mot connu au mot inconnu

Avant de nous intéresser à la classification des inconnus, il est primordial de définir ce qu'est un mot inconnu et comment le détecter au sein d'un texte. Nous commencerons donc ici par nous pencher sur les unités linguistiques sur lesquelles s'appuie cette étude. Ce choix déterminera comment nos corpus seront par la suite segmentés et comment nous distinguerons les *inconnus* des *non-inconnus*.

Nous avons pour l'instant utilisé l'unité linguistique MOT sans pour autant la définir. Cela a été réalisé sciemment. En effet, bien que cette notion soit à première vue compréhensible et interprétable par tous, sa définition ne fait pas l'objet d'un consensus (Tournier, 2004). Une première solution serait de s'appuyer sur un critère formel et de considérer qu'un mot correspond à l'ensemble des caractères compris entre deux blancs ou ponctuations. Néanmoins, cette proposition est généralement jugée insatisfaisante car trop simpliste. Elle ne permet ni de maîtriser le cas des mots agglutinés (*desquels*) ni celui des mots composés (*bandes dessinées*) et ne gère pas les problèmes que peuvent poser la ponctuation au sein d'une chaîne de caractères (*cahin-caha*, *porte-plume* ou *est-ce que* vs. *scénario-*

catastrophe ou *(ce) livre-là*). Par ailleurs, elle se révèle particulièrement inefficace pour les langues qui ne délimitent pas leurs mots par la typographie (c'est par exemple le cas du mandarin comme l'explique Magistry (2013)). Cette notion purement typographique a toutefois son utilité, et nous la dénoterons comme il est d'usage par le terme « token ». Bien que la notion de mot soit utilisée dans plusieurs travaux du domaine de manière imprécise, de nombreux linguistes ont tenté de la définir (Lachachi, 2011). Bloomfield (1933, p. 178), par exemple, propose de considérer un mot comme « une forme libre minimale » [1]. Cette définition, bien que brève, est difficile à appliquer et trop restrictive. Nous pouvons aussi citer Meillet (1921, p. 30) qui a cherché à déterminer les éléments constitutifs d'un mot. Il explique ainsi qu'« un mot résulte de l'association d'un sens donné à un ensemble de sons donnés susceptible d'un emploi grammatical donné. ». Néanmoins, ces éléments demeurent trop ambigus, comme le montrent Lehmann et Martin-Berthet (2013).

Bien que beaucoup de personnes aient suggéré des définitions de la notion de mot, aucun auteur n'est parvenu à en proposer une simple, précise, applicable de façon reproductible et, *a fortiori*, admise par tous. Cela pousse de nombreux chercheurs à considérer cette notion de mot comme non-universelle (Haspelmath, 2011). Certains vont ainsi abandonner cette unité au profit de termes plus techniques. Baudouin de Courtenay (1895) introduit donc la notion de MORPHÈME qui fut ensuite repris par Bloomfield (1926) afin de désigner le plus petit élément significatif. Saussure (1916), quant à lui, utilise la notion de SIGNE LINGUISTIQUE qui se compose d'un SIGNIFIANT et d'un SIGNIFIÉ. Ces derniers représentent respectivement une *image acoustique* et un *concept*. Les termes de LEXÈME et de MOT-FORME sont eux aussi introduits (cf. par ex. Polguère, 2008) : un mot-forme correspond à un signe linguistique détenant une autonomie de fonctionnement et une cohésion, alors qu'un lexème est défini comme un ensemble de mots-formes qui ne se distinguent que par la flexion. En lexicologie, Pottier (1962) propose le terme de LEXIE qui constitue une unité linguistique représentant une « unité lexicale mémorisée ». Enfin, nous pouvons citer Martinet (1985) qui propose une nouvelle unité minimale de signification : le MONÈME. Cette unité détient un signifiant et un signifié et est composée de deux autres types d'unités : le lexème et le morphème qui représentent respectivement sa partie lexicale et grammaticale. Tous ces termes, listés ici de manière non exhaustive, ont un usage bien plus limité que le « mot », mais ont tous l'avantage de pouvoir être définis plus précisément.

Ces débats parfois plus théoriques qu'applicatifs sur la notion de mot dépassent en grande partie le cadre de ce travail. En effet, les définitions proposées peuvent s'étendre sur de nombreux niveaux d'analyse linguistique et sont souvent très, voire trop, spécifiques étant donnés les corpus que nous souhaitons traiter. Le travail présenté ici se concentre sur des langues européennes occidentales telles que l'anglais, le français ou encore l'allemand. Nous pourrons par conséquent

1. En anglais: « a minimum free form. ».

nous appuyer au moins en partie sur les séparateurs présents dans nos corpus. L'unité que nous choisirons servira de base pour notre travail. Cette base doit être solide et non sujette à de multiples ré-interprétations. Les notions de TOKEN et de MOT-FORME proposées par la norme ISO MAF (Morpho-syntactic Annotation Framework ; Clément et Villemonte de la Clergerie, 2005) semblent toutes deux, au premier abord, susceptibles de répondre à ces besoins.

La norme ISO MAF définit un token comme une séquence ininterrompue de caractères isolée des autres à l'aide de séparateurs. Ces séparateurs peuvent correspondre à une marque de ponctuation [2] ou encore à un espace. Un mot-forme ou FORME est ici considéré comme une unité syntaxiquement atomique. Il détient des propriétés morphosyntaxiques et lexicales et peut correspondre à plusieurs entités contiguës, insécables, qui partagent une même catégorie grammaticale [3]. Pour simplifier, un token se rapproche de la notion de *mot typographique* tandis qu'un mot-forme pourrait être associé à une unité lexicale capable de constituer une entrée de dictionnaire. Ainsi ces deux termes ne se chevauchent pas nécessairement. Par exemple, la séquence « *chaise longue* » correspondra à deux tokens et un mot-forme (ou FORME COMPOSÉE) tandis que la séquence « *au* » correspondra à un token, mais à deux mots-formes amalgamés (*à*+*le*) [4].

Dans le présent travail, nous emploierons les notions de token et de mot-forme. Le terme de « mot » ne sera donc utilisé ici que de manière utilitaire au sens de mot graphique (ou token) et non dans son sens linguistique. Notre étude s'appuiera ainsi sur des lexiques composés des tokens de chaque langue traitée. Ces tokens peuvent correspondre aussi bien à un amalgame (*duquel*), à un mot-forme simple (*sapin*) ou à un token appartenant à mot-forme composé (*facto* par exemple pour la forme *de facto*) [5]. Un token ne figurant pas dans ce lexique sera considéré comme un TOKEN INCONNU ou plus simplement comme un INCONNU. La détection de ces tokens dépendant fortement du lexique utilisé, il est primordial de bien choisir celui avec lequel nous réaliserons notre analyse lexicale.

2. Certains caractères de ponctuation (tel que les points ou les tirets par exemple) peuvent ne pas jouer le rôle de séparateurs si on décide de ne pas les considérer comme tels. Ce détail est notamment utile si on est confronté à des tokens composés de symboles de ponctuation comme les *smileys* ou les adresses e-mail.

3. Bien que cette définition soit discutable, nous nous en contenterons ici puisqu'elle répond et suffit à nos besoins pour la réalisation de cette étude.

4. Ces deux segmentations du texte peuvent parfois être complexes à aligner. C'est par exemple le cas de « à l'instar du » qui se découpera en quatre tokens (*à + l' + instar + du*), mais à deux mots-formes (*à l'instar de + le*) (Sagot et Boullier, 2008).

5. Ajouter les différents tokens d'une forme composée dans nos lexiques peut être critiquable. En effet, nous avons conscience qu'il peut être risqué de considérer un élément de forme composée (non existant dans un lexique de langue de manière indépendante) comme valide sans être sûr qu'il n'est pas accompagné des autres éléments qui compose cette forme.

1.2 Les lexiques

Polguère (2008, p. 70) définit le lexique d'une langue comme « l'entité théorique correspondant à l'ensemble des lexies de cette langue ». Le terme *théorique* figurant dans cette définition permet de souligner le caractère discutable de la présence de certaines de ces lexies dans un lexique (cette question sera abordée à la section 1.2.1). En traitement automatique de la langue, on pallie l'absence de connaissances des systèmes mis en place en utilisant les informations contenues dans les lexiques. Ces derniers doivent donc être les plus complets possible et respecter un certain formalisme. En fonction de la tâche mise en place, on peut attendre d'eux qu'ils possèdent certaines informations sur chacun des tokens ou formes fléchies qu'ils contiennent. Cette partie détaillant principalement des lexiques de formes, nous utiliserons cette dernière unité plutôt que les tokens pour décrire leur contenu. Le lexique de tokens que nous souhaitons utiliser pour la suite de notre travail sera par ailleurs extrait de ces lexiques de formes.

Dans cette section, nous discuterons des points qui doivent être pris en compte afin de constituer un lexique ou d'en choisir un pour réaliser une analyse lexicale (section 1.2.1). Nous nous intéresserons ensuite aux lexiques morphologiques existants (section 1.2.2).

1.2.1 Quel est le contenu attendu d'un lexique ?

Avant de pouvoir créer ou choisir un lexique pour réaliser une tâche en traitement automatique des langues, il nous semble primordial de nous interroger sur le contenu attendu de ce lexique. Nos réflexions peuvent se résumer en deux questions :

1. Un lexique peut-il réellement être représentatif des formes d'une langue ?
2. Un lexique doit-il nécessairement être exhaustif ?

1.2.1.1 Un lexique peut-il être représentatif ?

Il est complexe de déterminer quelles sont les formes qui appartiennent à une langue. À partir de quel moment une forme peut-elle prétendre faire partie d'une langue ? Affirmer que c'est le cas à condition qu'elle soit dans un dictionnaire ou un lexique de cette langue n'est pas réellement satisfaisant puisque cette dernière pourrait figurer dans un lexique et être absente d'un autre (Sablayrolles, 2008). La forme *dédiaboliser*, par exemple, se retrouve dans le Robert, mais pas dans le Larousse ni dans le *Trésor de la langue française*. Peut-on alors considérer cette forme comme appartenant à la langue française ? Un dictionnaire ne prévalant pas sur un autre, nous pouvons supposer que c'est l'usage d'une forme qui la fait entrer dans une langue. Les différents lexiques d'une langue n'évaluent pas forcément

l'usage d'un mot de la même manière et peuvent par conséquent différer dans leurs contenus. Nous pouvons ainsi nous interroger sur la légitimité pour un mot d'appartenir au lexique d'une langue. Plusieurs cas de figure peuvent d'ailleurs être problématiques pour les lexiques :

1. Les formes non partagées par tous :
   On peut s'interroger sur la place des formes qui ne sont pas partagées par l'ensemble des locuteurs d'une langue, mais uniquement par certaines communautés de personnes. C'est par exemple le cas des québécismes ou régionalismes comme le mot *wassingue* qui signifie « serpillère » dans le nord de la France. Ces mots, bien qu'utiles dans le cas où l'on traite les corpus concernés, ne sont que très peu utilisés et alourdissent les dictionnaires. Les formes scientifiques peuvent poser le même problème. Aurons-nous vraiment envie d'avoir des formes telles que *corynébactérie* ou *choriocentèse* dans notre lexique ?

2. Les formes oubliées ou non usitées :
   On peut s'interroger sur la présence dans les lexiques de certaines formes qui ont disparu de la langue standard (Zumthor, 1967). La présence de ces archaïsmes ou formes vieillies dans les lexiques peut être problématique, voire gênante, en traitement automatique car elle ajoute des ambiguïtés qui n'auraient lieu d'être (le nom féminin *couverte* (*couverture de lit*) partage les formes fléchies avec l'adjectif *couverte*).

3. Les nouvelles formes :
   La langue ne cessant d'évoluer, de nouvelles formes sont constamment créées (Renouf, 1993). Cela ne signifie pas pour autant que chacune de ces nouvelles formes doive figurer dans le lexique. On peut en effet distinguer plusieurs types de nouvelles formes : (i) celles qui ne seront utilisées qu'une seule et unique fois, (ii) celles qui n'apparaissent que de manière temporaire et (iii) celles qui viennent remplir un vide lexical dans la langue et qui perdureront probablement. Le cas (i) correspond aux nouvelles formes créées, mais utilisées une seule fois (Pruvost et Sablayrolles, 2012, p. 59). Ces nouvelles formes, les hapax, peuvent par exemple être créées afin de pallier un trou de mémoire (ex. : la création du nom *casserolée* pour de parler d'une *poêlée*). Son caractère unique montre qu'elle n'a pas forcément été créée pour enrichir la langue de façon pérenne. Il serait plus aisé ici de considérer cette nouvelle forme comme une forme temporaire de substitution. En ce qui concerne les deux derniers cas (ii) et (iii), les distinguer se révèle plus complexe. Nous ne pouvons pas clairement savoir quelle nouvelle forme se maintiendra dans le temps. Par conséquent, il est difficile de déterminer auquel de ces deux cas appartient une nouvelle forme qui apparaît à plusieurs reprises. Ainsi, comme le précisent Pruvost et Sablayrolles (2012, p. 34-36), le statut d'une nouvelle forme évolue très rapidement, elle peut aussi bien disparaître de la langue comme s'y intégrer parfaitement en l'espace de quelques années.

Reste à noter que cette difficulté se retrouve pour l'intégration d'un mot étranger emprunté au lexique d'une autre langue.

Enfin, la graphie de certaines formes peut varier d'un lexique à l'autre en fonction de variations orthographiques. Ces variations peuvent être aussi bien l'œuvre d'une réforme de l'orthographe (suppression de l'accent circonflexe sur la lettre *i* par exemple : *paraître* → *paraitre*) que résulter de l'évolution d'une forme dans une langue. Le nom *soprane* par exemple est ainsi parfois accepté en parallèle à la graphie *soprano* qui correspond à la forme empruntée de l'italien. Nous pouvons par ailleurs noter qu'il arrive que la norme graphique d'un mot ne soit pas celle adoptée par la majorité des locuteurs d'une langue. C'est le cas de la forme normée *boguer*, qui apparaît dans des textes, écrite de multiples manières comme : *buguer*, *bugger*, *bugguer*, etc. (Walther et Sagot, 2011). On peut s'interroger sur la pertinence de voir figurer toutes ces formes dans un lexique. Y ont-elles leur place ou ne vaudrait-il mieux pas les considérer comme des fautes et imposer de manière systématique leur forme normée ? Intégrer ces formes dans un lexique n'est intéressant qu'en fonction de l'emploi que l'on veut faire de ce lexique, de la manière dont on veut l'utiliser.

1.2.1.2 Un lexique doit-il être exhaustif ?

L'objectif principal d'un lexique est de contenir, au mieux, tous les éléments (tokens/formes) qui composent la langue. On pourrait donc penser dans un premier temps que pour analyser automatiquement la langue, il est préférable de détenir un lexique aussi complet que possible. Néanmoins, la langue ne cessant d'évoluer, notre lexique ne peut être exhaustif (Dister et Fairon, 2004). Cartoni (2006) a d'ailleurs montré que l'incomplétude lexicale est bien un problème constant qui dépend en grande partie de l'apparition continue de nouvelles formes. Pour cela, il a sectionné un corpus en plusieurs sous-corpus. Afin de réduire le taux d'inconnus à 0% pour chacun d'entre eux, il a tenté d'étendre son lexique entre la lecture de chaque sous-corpus à l'aide des nouvelles formes trouvées précédemment. Néanmoins, bien que l'écart entre le pourcentage d'inconnus avant et après alimentation du lexique se soit rapidement amenuisé après la lecture des premiers sous-corpus [6], il a, par la suite, beaucoup moins évolué. Cette expérience montre notamment qu'alimenter un lexique ne suffit pas à résoudre les problèmes posés par l'incomplétude lexicale et souligne une caractéristique que partagent de nombreux inconnus, à savoir cette faculté de pouvoir apparaître une seule et unique fois. Nous avons nous même remarqué ce phénomène dans un flux textuel constitué de dépêches d'agence de presse (Sagot et al., 2013).

6. Cette première progression s'explique par l'ajout de formes qui étaient manquantes, lacunaires dans le lexique et l'ajout de nouvelles formes utilisées à plusieurs reprises dans plusieurs parties du corpus.

On peut s'interroger sur le caractère bénéfique qu'aurait un lexique représentant réellement une langue dans sa totalité. Serait-il réellement judicieux pour notre système de s'appuyer sur un lexique dans lequel figureraient, en plus des formes courantes d'une langue, ses formes plus techniques ou encore ses archaïsmes ? En effet comme nous l'avons vu précédemment, la présence de ces derniers dans un lexique alourdirait de manière conséquente les traitements linguistiques, tout simplement de par la taille de ce lexique, mais aussi de par les ambiguïtés qu'il provoquerait. Par ailleurs dans le cadre de la correction orthographique, certaines fautes courantes pourraient ne plus être détectées parce qu'elles correspondraient à des formes rares. Par exemple, c'est le cas de l'archaïsme *affin* qui a plus de chance de correspondre à la forme erronée de *afin* ou *affine*. Nous pourrions aussi être tentée de corriger une faute initialement simple par une forme à présent inusitée. Si l'on considère que ce même archaïsme est dans notre lexique et qu'on rencontre la forme erronée *afiin*, un système hésitera entre les formes *afin* et *affin* pour la corriger.

Si utiliser un lexique complet est important, il est indispensable que ce dernier sache se limiter aux formes usuelles et courantes. Un lexique spécifique pouvant être ajouté en cas de nécessité (si on tente par exemple de traiter un corpus technique). Chercher à tout prix à étendre la couverture d'un lexique n'est donc pas indispensable.

1.2.2 Lexiques morphologiques existants

L'accessibilité des corpus et le besoin réel de ressources linguistiques pour le traitement automatique des langues ont poussé plusieurs chercheurs à s'intéresser au développement de lexiques. En France, ces travaux ont notamment été encouragés par l'apparition de dictionnaires informatisés. On peut notamment citer Le TLFi (*Trésor de la Langue Française informatisé*) et les travaux réalisés au sein du laboratoire LADL sur la constitution de dictionnaires électroniques DELA [7] (cf. Courtois (1990) et Silberztein (1990, 1993)). On définit un lexique informatisé comme un ensemble d'entrées structurées, à l'aide d'un support informatique, associées à des informations. Ces informations varient ensuite en fonction du besoin qui motive la construction de ce lexique, elles peuvent aussi bien être d'ordre syntaxique, morphologique, lexical, sémantique ou encore phonétique. Chacune peut avoir son utilité en fonction de l'application que l'on a de ce lexique.

Pour réaliser notre travail, une liste de tokens ou de formes simples, sans informations complémentaires concernant leur lemme, leur catégorie grammaticale ou encore leur flexion, ne nous suffirait pas. En effet, nous avons besoin pour certaines tâches (telle que la détection de néologismes dérivationnels) de disposer d'informa-

7. Cet ensemble de dictionnaires réunit le dictionnaire des mots simples DELAS, le dictionnaire de transcription phonétique DELAP, le dictionnaire des formes fléchies DELAF et le dictionnaire des formes composées DELAC.

tions morphologiques. Par ailleurs, nous souhaitons utiliser des ressources libres de sorte que l'outil présenté dans cette thèse puisse être utilisable aussi bien par des chercheurs que par des entreprises.

Nous donnerons ici un aperçu des lexiques morphologiques pour le français qui ont retenu notre attention en commençant par les décrire brièvement avant de les comparer.

1.2.2.1 Lexiques existants

Quatre lexiques distribués librement se démarquent particulièrement :

1. Morphalou (Romary et al., 2004)
2. Lexique 3 (New, 2006)
3. Le Le*fff* (Lexique des Formes Fléchies du Français) (Sagot, 2010)
4. Le GLÀFF (Gros Lexique À tout Faire du Français) (Hathout et al., 2014)

1.2.2.1.1 Morphalou est un lexique morphologique de formes fléchies. Il correspond à une version XML du lexique TLFnome (qui est lui-même issu de la nomenclature du TLF (*Trésor de la Langue Française*). Il est disponible sur le site du CNRTL (`www.cnrtl.fr/lexiques/morphalou/`).

Ce lexique contient 539 413 entrées (pour 68 075 lemmes). Chacune de ces entrées est associée à son lemme, à sa forme phonétique, à sa catégorie grammaticale et à ses traits flexionnels. Néanmoins, il est important de préciser que Morphalou contient beaucoup d'entrées qui correspondent à des archaïsmes.

1.2.2.1.2 Lexique 3 est une version améliorée des lexiques *Lexique* 1 et 2 (New et al., 2001, 2004). Ces derniers avaient été mis en place afin de pallier les défauts de *Brulex*, la première base de données lexicale informatisée mise à disposition des psycholinguistes, à savoir : (i) l'absence de formes fléchies, (ii) les fréquences contenues non représentatives de la langue actuelle écrite et parlée et (iii) l'absence de mise à jour du lexique.

Les entrées du lexique proposé par New (2006) proviennent du corpus *Frantext,* mais aussi de dialogues de films et de séries afin de contenir un vocabulaire plus représentatif du langage parlé actuel. Néanmoins, on peut noter que tout comme Brulex seules les formes les plus usuelles sont couvertes.

Lexique 3 détient près de 135 000 entrées (pour environ 55 000 lemmes). En plus d'être associée à des informations morphosyntaxiques, chaque entrée se voit aussi rattachée à sa transcription phonémique, une segmentation syllabique, des

informations sur les homophones, les homographes, les voisins phonologiques et orthographiques, la fréquence des formes dans des corpus écrits, etc.

1.2.2.1.3 Le Le*fff* Ce lexique morphologique et syntaxique du français (Sagot, 2010) a été construit sur le formalisme Alexina [8] afin de pallier l'absence d'un tel lexique pour le français, à la fois libre de droits, à large couverture et de qualité. Ce lexique est notamment utilisé pour des tâches de traitement automatique de la langue.

Le Le*fff* possède près de 120 000 lexèmes définis par un lemme, une catégorie grammaticale, une classe flexionnelle, ainsi que des informations syntaxiques (cadre de sous-catégorisation notamment). À partir du lemme et de la classe flexionnelle, une grammaire morphologique associée permet de produire l'ensemble des formes fléchies, rattachées à des informations issues de celles associées au lexème. Ceci produit plus de 500 000 entrées fléchies.

De nombreux lexiques flexionnels ont été construits en conformité avec l'architecture Alexina (Sagot, 2010) dans d'autres langues. On peut notamment noter l'existence des lexiques Enlex pour l'anglais, DeLex pour l'allemand (Sagot, 2014) et Le*ffe* pour l'espagnol (Molinero et al., 2009) qui correspondent aux trois autres langues pour lesquelles nous aimerions adapter notre système à terme.

1.2.2.1.4 Le GLÀFF Ce lexique morphologique du français a été conçu à partir des entrées du Wiktionnaire. Il a été construit afin de pallier l'absence de lexique électronique libre, à large couverture qui contiendrait en plus des informations morphosyntaxiques des transcriptions phonémiques.

Il contient près de 1,4 million d'entrées et, sont associés à chacune d'entre elles un lemme, une description morphosyntaxique et, dans 90% des cas, une ou plusieurs transcriptions phonémiques. Ce grand nombre d'entrées s'explique par le fait que le Wiktionnaire (et par conséquent le GLÀFF) possède beaucoup de formes rares, argotiques ou encore régionales ainsi que de nombreuses variantes orthographiques qui correspondent pour la plupart à des archaïsmes [9].

8. Site internet `http://alexina.gforge.inria.fr/`

9. En effet comme il est expliqué dans les critères d'acceptation des articles du Wiktionnaire, « La rareté ou la notoriété d'un mot n'est pas un critère, contrairement à Wikipédia, pourvu que l'on soit sûr qu'il soit bien attesté. Les mots récents, les mots familiers, les mots argotiques sont considérés comme faisant partie de la langue. Il est d'autant plus utile qu'ils sont souvent absents des autres dictionnaires. Les mots et les définitions désuets peuvent figurer sur le Wiktionnaire, du moment qu'ils sont attestés. Par exemple, *loix*, l'ancien pluriel de *loi* n'est jamais utilisé en français contemporain, mais existe toujours dans des documents écrits. Les variantes orthographiques d'un même mot peuvent être toutes présentes, sauf s'il s'agit clairement de fautes d'orthographe. » (Sajous et al., 2014).

1.2.2.2 Comparatif des lexiques

Tous les lexiques présentés ci-dessus diffèrent les uns des autres. La table 1.1 permet notamment d'avoir une vue d'ensemble du nombre d'entrées contenues dans chaque lexique.

Lexique	nb entrées
Morphalou	540 000
Lexique 3	135 000
Le*fff*	500 000
GLÀFF	1 400 000

Table 1.1 – Nombre d'entrées contenues par lexique

Comme nous pouvons le constater, Lexique 3 semble avoir relativement peu de mots-formes si on le compare aux autres. Cela s'explique par le fait qu'il ne possède que les formes fléchies les plus usuelles (Hathout et al., 2014). On peut par ailleurs noter le nombre très élevé d'entrées du GLÀFF. Ce dernier n'est toutefois pas surprenant puisque ce lexique prend aussi en compte des formes rares, voire erronées, de la langue comme expliqué ci-dessus.

Hathout et al. (2014) se sont essayés à comparer la couverture de chacun de ces lexiques sur le vocabulaire utilisé dans quatre types de corpus de taille et de nature différentes. Ces corpus sont : Frantext (un corpus composé de romans datant du XXème siècle), LM10 (un corpus composé des articles du journal *Le Monde* parus entre les années 1990 et 2000), la Wikipédia française et frWaC (un corpus composé de pages web). Les résultats obtenus sont visibles dans la table 1.2 extraite de Sajous et al. (2014).

Dans ce tableau, les corpus sont triés (de haut en bas) en fonction de leur qualité et de leur style. Ainsi, on commence avec Frantext qui est doté de textes relativement soutenus et bien orthographiés pour finir avec frWaC qui est composé de textes souvent « bruités ». Les scores représentés dans ce tableau décroissent ainsi en conséquence. Bien que sur des tokens fréquents (avec une occurrence égale ou supérieure à 100) ces lexiques restent très similaires (excepté dans le cas de frWaC où l'on note plus de différences), on peut noter qu'ils se distinguent sur des tokens moins courants. Ces différences s'expliquent de par les choix qui ont présidé au développement de chaque lexique. Ainsi, un lexique qui a choisi de conserver les archaïsmes aura de fortes chances d'avoir de meilleurs scores sur Frantext. De même, un lexique qui intégrera rapidement les néologismes ou certaines formes d'argot se fera plus remarquer sur un corpus bruité représentant en partie le langage parlé tel que frWaC. Il est par ailleurs important de noter que, dans ce dernier cas, obtenir une meilleure couverture n'est pas systématiquement ce qui sera le plus recherché par un lexique. En effet, le frWaC étant un corpus bruité, un lexique ne souhaitera pas couvrir les formes mal orthographiées. Enfin, il semble

important de préciser que le fait que chaque lexique contienne ou non des entités nommées aura automatiquement un impact sur les résultats présentés ici.

Seuil : fréquence ≥		**1**	**2**	**5**	**10**	**100**	**1000**
Frantext	Nb formes	145 437	95 189	61 813	43 919	10 767	1 376
	Lexique	66,76	84,35	94,00	**96,91**	**99,15**	**99,27**
	Le*fff*	71,89	85,63	93,21	96,21	99,08	98,90
	Morphalou	73,93	86,66	93,29	96,00	98,48	97,09
	GLÀFF	**76,92**	**88,57**	**94,54**	96,72	98,77	98,76
LM10	Nb formes	300 606	172 036	106 470	77 936	29 388	7 838
	Lexique	29,59	47,28	65,23	76,31	93,81	98,58
	Le*fff*	39,64	58,22	74,33	83,20	95,99	**98,90**
	Morphalou	39,06	56,82	71,92	80,32	93,27	97,48
	GLÀFF	**45,24**	**63,83**	**78,63**	**86,23**	**96,46**	98,68
Wikipédia	Nb formes	953 920	435 031	216 210	136 531	35 621	7 956
	Lexique	9,13	18,27	31,52	43,03	78,58	95,72
	Le*fff*	12,88	23,94	38,26	49,65	80,57	95,71
	Morphalou	13,05	23,96	37,87	48,87	78,74	94,16
	GLÀFF	**16,42**	**29,00**	**44,13**	**55,45**	**83,21**	**96,10**
FrWaC	Nb formes	1 624 620	846 019	410 382	255 718	74 745	22 100
	Lexique	5,83	10,85	20,84	30,81	66,00	89,47
	Le*fff*	9,85	16,67	28,57	39,16	71,61	91,16
	Morphalou	10,09	16,89	28,53	38,68	69,36	88,51
	GLÀFF	**13,13**	**21,13**	**34,29**	**45,35**	**76,39**	**92,76**

TABLE 1.2 – Couverture des lexiques/corpus (en %) de Sajous et al. (2014)

1.3 Les tokens inconnus

1.3.1 Catégorisation des inconnus

La présence dans les textes de tokens inconnus des lexiques étant problématique, de nombreuses études ont tenté de déterminer ce à quoi correspondaient ces inconnus et ont proposé plusieurs classifications (Ren et Perrault, 1992; Maurel, 2004; Dister et Fairon, 2004; Cartoni, 2008; Blancafort et al., 2010; Dutrey et al., 2012; Falk et al., 2014). Si ces classifications diffèrent, c'est notamment car ces études ne cherchent pas à analyser les mêmes types de textes. En effet, les inconnus rencontrés dans un corpus journalistique ne seront pas les mêmes que ceux présents dans des corpus issus du web. Par ailleurs, beaucoup de ces études se distinguent les unes des autres en fonction des différentes classes qu'elles ont choisies

afin de répartir leurs tokens inconnus. Les principales catégories proposées sont les suivantes [10] :

1. Les **créations lexicales**, souvent caractérisées comme des formes construites selon un ou des procédés identifiables (Cartoni, 2008), ne sont finalement que très peu définies. Néanmoins, en observant les classifications proposées par les différentes études, on peut constater que la définition de cette classe n'est pas commune à tous les auteurs. En effet même si tous englobent dans la notion de création lexicale les abréviations et les néologismes construits par dérivation et par composition, certains ajoutent d'autres types de formes. Ainsi, on peut citer Maurel (2004) qui inclut les chiffres romains, les dérivés de noms propres et les sigles dans les créations lexicales. Cartoni (2008) reprend par ailleurs cette liste et y ajoute les onomatopées. En outre, on constate aussi dans les créations lexicales proposées par Blancafort et al. (2010) que ces derniers prennent aussi en compte les constructions *ad hoc* (ex. : *autodétermination*). Nous considérerons dans cette étude les créations lexicales comme l'ensemble des nouvelles formes construites suite à l'application de règles morphologiques propre à la langue traitée sur des formes déjà existantes. Cette catégorie est considérée par de nombreux auteurs comme celle qui contient le nombre le plus important d'inconnus (Ren et Perrault, 1992; Dister et Fairon, 2004; Maurel, 2004; Cartoni, 2006) [11].

2. Les **emprunts** apparaissent dans toutes les typologies réalisées. Ces derniers correspondent à des tokens provenant de langues étrangères. Ils sont souvent considérés comme inconnus car ils ne figurent que très rarement dans le lexique de la langue cible. Néanmoins, si un emprunt est très présent dans une langue, il peut par la suite s'intégrer à cette dernière et être modifié en appliquant certains mécanismes de formation de la langue, c'est à dire à l'aide de règles lexicogéniques. Dans ce cas, on parlera d'un **emprunt adapté**. C'est par exemple le cas du verbe *liker* qui a été créé sur la base du verbe anglais *like*.

3. Les **formes** considérées comme **fautives**, ou **erreurs**, figurent elles aussi de manière systématique dans les classifications des mots inconnus et ce, de manière plus ou moins précise. Le chapitre suivant de cette thèse traitant en grande partie la notion de forme fautive, nous ne nous attarderons pas sur cette catégorie ici.

4. Les **entités nommées** (dans leur sens étendu ou non) figurent très fréquemment dans les études faites sur les inconnus (Ren et Perrault, 1992; Maurel,

10. Les catégorisations et sous-catégorisations des inconnus proposées dans ces différentes études n'étant pas systématiquement les mêmes, nous nous sommes permis d'en lisser certaines afin de pouvoir les comparer plus aisément.

11. Le pourcentage de mots inconnus dans une catégorie dépendant fortement de l'étude qui propose ce résultat, nous ne ferons aucune généralisation ici. Le lecteur intéressé peut se référer aux références indiquées.

2004; Blancafort et al., 2010). En effet, cette catégorie de formes est problématique en traitement de la langue car elles sont complexes à lister et leur détection automatique n'est pas toujours aisée. Elles se retrouvent ainsi très souvent considérées comme des tokens inconnus. Toutefois, nous pouvons noter que certaines études ont préféré exclure les noms propres, considérant que ces derniers représentent une problématique relativement différente des autres inconnus (Cartoni, 2008) ou, afin d'optimiser la tâche à réaliser sur les inconnus (Dister et Fairon, 2004; Falk et al., 2014).

5. On peut par ailleurs, constater que quelques travaux citent parmi les différents types d'inconnus les **formes rares, régionales ou spécialisées** (Ren et Perrault, 1992; Dister et Fairon, 2004). Ren et Perrault (1992) évoquent plus particulièrement les régionalismes et les mots scientifiques et Dister et Fairon (2004) mentionnent quant à eux la présence de québécismes [12] et de noms féminisés parmi leurs inconnus.

6. Dister et Fairon (2004) et Blancafort et al. (2010) ajoutent par ailleurs une classe pour représenter les inconnus correspondant à des tokens concernés par la **réforme de l'orthographe**.

7. Certaines études ont aussi intégré à leur classification une ou des catégories représentant plus particulièrement des **tokens propres au langage web** [13]. Ces catégories traitent ainsi des formes correspondant à des procédés expressifs comme les smileys, à de l'écriture phonétisante ou encore à des mails ou à des URL (Dutrey et al., 2012; Blancafort et al., 2010).

8. Enfin, à toutes ces catégories d'inconnus s'ajoutent les tokens qui auraient dû figurer dans le lexique informatisé de référence. En effet comme Cartoni (2008) le précise, il peut arriver qu'un token, bien que courant et répertorié dans beaucoup de lexiques, soit absent de celui qu'on utilise, et ce sans raison valide.

1.3.2 Représentation du lexique proposée par Tournier

Avant d'aller plus loin dans la classification des inconnus, il est intéressant de comprendre, linguistiquement parlant, comment s'agencent les différentes unités qui constituent le lexique d'une langue et les mécanismes qui leur permettent d'évoluer au sein de cette langue. Tournier (2004) explique que les éléments qui composent une langue se divisent en quatre catégories : Le lexique réel, le lexique potentiel, le non-lexique et le xénolexique (comme illustré dans la figure 1.1). Le LEXIQUE RÉEL correspond à l'ensemble des formes réalisées dans la langue

12. Leur étude a pour but de faire de l'extension de ressources lexicales en s'appuyant sur un corpus dynamique de presse québécoise.

13. Ce style rédactionnel a fait l'objet de nombreuses études telles que (Seddah et al., 2012), nous y reviendrons au chapitre 3.

à un instant t. Tournier distingue dans ce lexique les formes répertoriées (qui appartiennent à une ZONE SÛRE) des formes qui ne le sont pas (et qui figurent par conséquent dans une ZONE FLOUE). Les nouvelles formes apparues récemment, par exemple, demeurent ainsi dans la zone floue du lexique réel. Le LEXIQUE POTENTIEL représente l'ensemble des formes qui n'ont jusqu'ici jamais été créées, mais qui pourraient l'être à l'aide des RÈGLES LEXICOGÉNIQUES [14] existantes dans la langue à cet instant t. Le NON-LEXIQUE désigne toutes les formes non réalisées et non réalisables à l'aide de règles lexicogéniques. Enfin, le terme XÉNOLEXIQUE illustre l'ensemble des formes qui appartiennent aux lexiques réels d'autres langues que celle traitée.

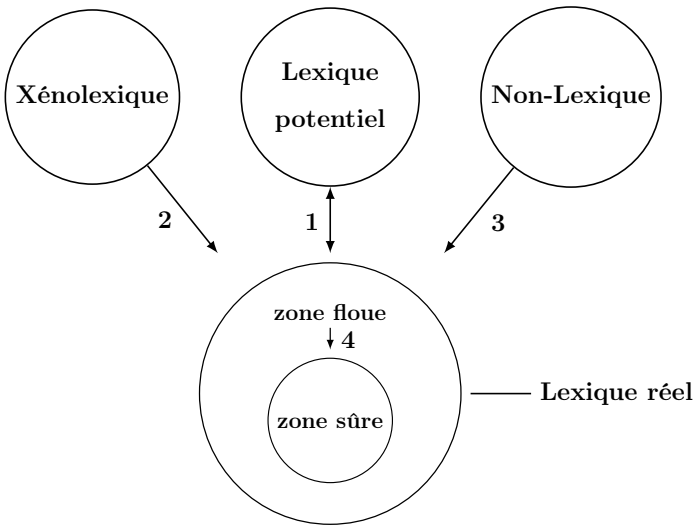

FIGURE 1.1 – Représentation de la dynamique lexicale par Tournier

Cette vision du lexique relativement simple nous permet de mieux appréhender les formes produites dans la langue. Le concept de DYNAMIQUE LEXICALE, défini par Tournier (1985), permet notamment de comprendre les différents phénomènes qui induisent la création d'un mot inconnu. Les flèches présentes dans la figure 1.1 représentent les différents transferts possibles entre ces quatre ensembles. La flèche 1 permet ainsi la réalisation de nouvelles formes (construites en combinant des formes du lexique réel avec des règles lexicogéniques) qui demeuraient jusque-là potentielles (ex. : *démondialisation*). La flèche 2 explique la présence de mots empruntés dans notre langue (ex. : *vegan*) [15]. La 3^{e} flèche représente enfin le cas, rare, d'apparition de mots dans le lexique réel réalisés sans l'aide de règles existantes dans la langue (ex : *Zumba*). Enfin, la 4ème flèche illustre l'entrée dans la zone sûre d'une forme qui n'était jusque-là pas répertoriée. C'est par exemple le cas du verbe *procrastiner* qui a longtemps été utilisé sans être répertorié.

14. Les règles lexicogéniques sont décrites comme des règles décrivant les mécanismes de formation des mots de la langue (Tournier, 2004).

15. Les emprunts adaptés résultant ainsi de la combinaison de la flèche 2 puis de la flèche à double sens 1 (ex. : *végane* ou *véganisme*).

1.3.3 Les inconnus dans la dynamique lexicale

Nous aimerions modéliser cette représentation de la langue afin de mieux appréhender le problème que provoque la présence d'inconnus dans les corpus écrits que nous serons amenée à traiter. Cette opération nous oblige ainsi à fixer des notions qui restaient assez théoriques dans la représentation de Tournier. La zone sûre (« zone constituée d'éléments réalisés et répertoriés ») contenue dans le « lexique réel » soulève par exemple quelques questions. Que signifie ici « répertorié » ? Doit-on s'appuyer sur la connaissance du locuteur qui a produit cet élément ou sur une ressource linguistique concrète ? Et dans ce dernier cas, nous devons nous demander quelle ressource il convient d'utiliser. Un système ne pouvant pas, dans notre cadre, se reposer sur les connaissances du locuteur, il s'appuie sur un lexique de référence. C'est à l'aide de ce dernier que nous représentons la zone sûre du lexique réel d'une langue. Lorsqu'un token appartenant à la zone floue de notre lexique réel apparaît dans un texte, il est annoté comme inconnu.

Dans cette section, nous commencerons par nous concentrer sur les différentes manières pour un inconnu d'intégrer la zone floue du lexique réel. Nous nous intéresserons ensuite plus en détail aux différents éléments qui composent le lexique réel.

1.3.3.1 Intégration des tokens dans le lexique réel

Pour prendre en compte les inconnus présents dans nos corpus et les diverses classifications observées, nous proposons de reprendre et d'adapter la représentation de Tournier afin de comprendre et d'expliciter la présence de ces différents tokens inconnus dans la zone floue.

Comme dit précédemment, la dynamique lexicale permet, entre autres, d'expliquer l'apparition de nouvelles formes dans la zone floue du lexique réel. Ainsi en s'appuyant uniquement sur son schéma de la dynamique lexicale, on peut d'ores et déjà expliquer la présence de nombreux inconnus. Ainsi, comme nous l'avons dit plus haut :

- La flèche 1 permet de justifier la construction de nouvelles formes créées à l'aide de règles telles que les emprunts adaptés (ex : *buguer*), les entités nommées dérivées (ex : *sarkozisme*), les abréviations (ex : *abrev*), les néologismes (ex : *réseautage* qui est produit par dérivation ou *dermographe* construit par composition) ou encore *ad hoc* (ex : *cyberattaque*).
- La flèche 2 reliant le xénolexique au lexique réel justifie ainsi l'apparition d'emprunts (ex. : *bug*).
- Enfin la 3^{e} flèche peut aussi représenter l'introduction dans notre langue d'inconnus correspondant à des métadonnées (c'est par exemple le cas de lien URL comme *www.monsite.com* ou d'adresse e-mail), à des pseudonymes ou

encore à des hashtags. On peut par ailleurs supposer que les entités nommées appartiendraient à cette même catégorie. En effet, ces dernières possèdent un mécanisme de création non formalisable, on ne peut pas les rattacher au lexique potentiel. Les lister intégralement dans un lexique n'est pas non plus envisageable dans la mesure où elles appartiennent à une classe ouverte et ne sont pas listables.

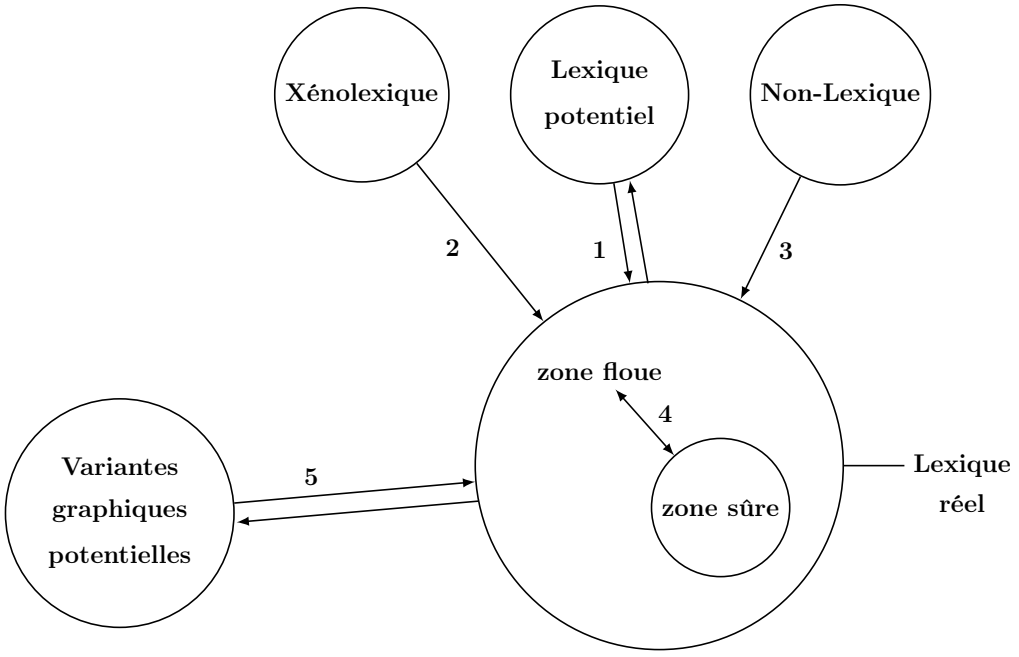

FIGURE 1.2 – Représentation adaptée de la dynamique lexicale par Tournier

Si nous nous limitons à la représentation de Tournier, il est difficile d'expliquer la présence d'inconnus construits pour des raisons orthographiques, ce problème étant inhérent à l'écrit. Afin de pallier ce manque, nous proposons d'ajouter un nouvel ensemble nommé VARIANTES GRAPHIQUES POTENTIELLES dans cette représentation. La figure 1.2 illustre cet ajout. Ce nouvel ensemble correspond à tous les tokens, jusqu'ici non créés, qui pourraient être produits suite à la réécriture d'un token présent dans le lexique réel. Il permettrait de justifier l'apparition dans le lexique réel (via la flèche 5) d'inconnus provoqués par une simplification graphique volontaire (*2m1* pour *demain*), une graphie expressive (*nooooon*), une instabilité graphique (*clé* vs *clef*), une variation orthographique due à une réforme de l'orthographe (*connaitre* vs *connaître*) ou simplement une faute d'orthographe (*ereur*) [16]. Les tokens issus de cet ensemble seront appelés dans la suite de ce travail ALTÉRATIONS.

16. Nous distinguons ici les fautes d'orthographe des simplifications graphiques, ces dernières étant réalisées volontairement.

Nous prenons par ailleurs le parti de considérer les inconnus correspondant à des smileys et à des formes nouvelles d'onomatopées et interjections [17] comme des tokens issus d'un ensemble dérivé du xénolexique. Nous pensons qu'ils proviennent non pas d'une langue étrangère à proprement parler, mais de deux ensembles représentant le monde qui nous entoure. Les onomatopées appartiendraient ainsi à un ensemble regroupant toutes les retranscriptions possibles des bruits ou sons qui nous entourent (ex. : *bouhouhou* ou *gné*). Les smileys, quant à eux, seraient issus d'un ensemble composé de toutes les représentations possibles de l'aspect visuel du monde et des objets de ce monde au moyen de signes graphiques usuels (ex : :D, <3 voire ♡).

1.3.3.2 Tokens constituant le lexique réel

La représentation proposée ci-dessus permet de modéliser l'intégration de nouveaux tokens dans la langue (dans le lexique réel). Néanmoins, elle n'explique pas les raisons qui font que des tokens, présents dans nos corpus écrits, qui peuvent nous sembler connus et admis par tous, demeurent inconnus de notre lexique de référence. En effet, elle n'explique pas clairement leur passage de la zone floue du lexique réel vers sa zone sûre. Nous tâcherons ainsi dans cette partie de détailler la diversité des tokens absents de notre lexique informatisé afin de mieux comprendre ce qui va permettre à un token d'entrer dans un lexique.

Bien qu'en théorie l'on puisse tracer une frontière nette entre les tokens répertoriés et ceux qui ne le sont pas, en pratique cette frontière est moins aisée à représenter. Nous proposons donc de considérer chacun des tokens qui composent le lexique réel comme une unité qui répond à 3 questions : (i) ce token est-il répertorié dans notre lexique informatisé de référence ? (ii) est-il intégré dans l'usage de la langue [18] ? (iii) appartient-il à la norme de cette langue [19] ? Nous pouvons par conséquent distinguer 8 types de tokens. Nous faisons le postulat que si un token est dans un lexique alors il fait au moins partie de l'usage ou de la norme. Cela restreint le nombre de types de tokens à 7. Ces derniers sont représentés dans la figure 1.3 présente ci-dessous. Le cercle gris foncé représente ici les tokens qui rentrent dans l'usage de la langue, le gris clair ceux qui font partie de la norme. Enfin, la zone rouge superposée à ces deux cercles représente l'ensemble des tokens qui figurent dans notre lexique informatisé (soit la zone sûre de Tournier). Les chiffres présents dans cette figure représentent enfin les différents types de tokens introduits précédemment et détaillés ci-dessous :

17. Les formes dérivées d'onomatopées interjections telles que *hahha* ou *hahahaahaaa*) proviennent quant à elles du lexique potentiel.

18. Nous considérons qu'un token rentre dans cette catégorie s'il est compris et connu par une majorité de locuteurs de cette langue

19. Cette dernière question présuppose naturellement l'existence d'une telle norme, ce qui est le cas pour toutes les langues traitées dans cette thèse.

1. Les tokens créés, mais non partagés par une communauté, qui ne figurent ainsi ni dans l'usage ni dans la norme d'une langue (ex : *2mandD*, *troller*) ;
2. les tokens appartenant à l'usage, mais qui ne font pas encore partie de la norme (*kiffer*, *psychoter*) et qui ne sont pas répertoriés (donc absents de notre lexique de référence) ;
3. les tokens non répertoriés appartenant à l'usage qui peuvent aussi bien avoir été intégrés dans la norme depuis peu (ex : *lol, googliser* ou *selfie*) ou y figurer depuis longtemps (l'absence dans notre lexique de référence des tokens concernés par ce dernier cas ne peut pas être expliquée de manière logique puisqu'il s'agit vraiment ici d'incomplétude lexicale) ;
4. les tokens qui appartiennent encore à la norme, mais qui, n'étant plus d'usage, sortent du lexique (ex. : *havir* ou *avertin*) ;
5. les tokens répertoriés appartenant uniquement à la norme (ex. : *hâlâmes* ou *tintinnabuler*) ;
6. les tokens présents dans notre lexique et qui figurent dans l'usage de la langue, mais pas dans la norme (ex. : *kéké* ou *chelou*) ;
7. Les tokens répertoriés qui apparaissent aussi bien dans l'usage que dans la norme de la langue (ex : *faire*, *pomme* ou *penser*).

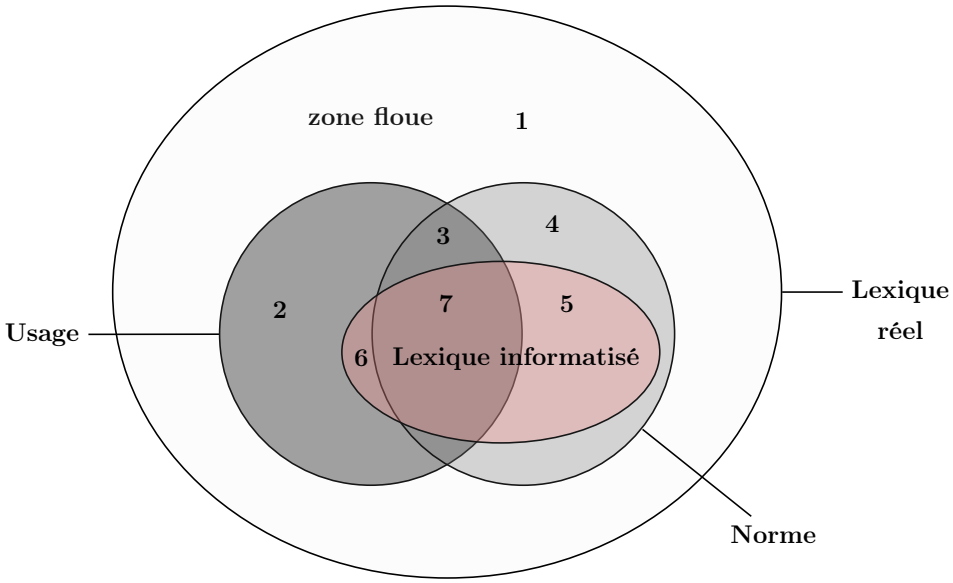

FIGURE 1.3 – Adaptation du lexique réel

Contrairement aux tokens de type 4, ceux des types 2 et 3 ont des chances d'intégrer notre lexique informatisé de référence. Les tokens de type 1 quant à eux ne seront ajoutés que s'ils rentrent dans l'usage (soit lorsqu'ils passeront en type 2 ou 3).

Ce schéma nous permet enfin de mieux comprendre comment se répartissent les tokens appartenant à des domaines spécifiques (médecine, informatique...), au langage SMS ou encore aux régionalismes. En effet, ces derniers ont une caractéristique commune, ils ne sont utilisés que par un nombre restreint de locuteurs d'une langue. Ils ne figurent donc pas systématiquement dans l'usage et/ou dans la norme et par conséquent ne sont que très rarement dans notre lexique de référence. La majorité de ces tokens se divise ainsi principalement entre les différents types de tokens ne figurant pas dans le lexique de référence (soit les types 1, 2, 3 et 4). La répartition de ces tokens dans le lexique réel est particulièrement problématique en traitement de la langue puisqu'elle ne permet pas à un système de considérer ces tokens comme appartenant à la langue. Généralement, ce problème est pallié en associant le lexique de référence à des lexiques plus spécifiques des domaines ou du type de tokens que l'on veut couvrir. Ainsi, Ringlstetter et al. (2006) proposent de prendre en compte des dictionnaires spécifiques (argotiques, archaïques) afin de couvrir l'incomplétude de leur lexique de référence. De même, Dister et Fairon (2004) qui travaillent sur des corpus québécois se sont adaptés en intégrant un lexique plus spécifique au québécois et ont par ailleurs ajouté un lexique des professions et de l'orthographe réformée afin de couvrir au mieux les différents tokens considérés comme inconnus par leur système. On peut enfin citer Ren et Perrault (1992) qui proposent d'enrichir leur lexique.

CHAPITRE 2

Détection automatique des inconnus

Sommaire

2.1 Classification des inconnus

La classification d'inconnus a toute sa place dans un travail comme le nôtre. Cela peut, entre autres, nous éviter de modifier des formes qui avaient toute leur place dans nos corpus. C'est pourquoi nous nous sommes plus particulièrement intéressée aux techniques de détection automatique des principaux types d'inconnus que nous avons étudié au chapitre précédent. Nous nous concentrerons ici sur la détection automatique des entités nommées, des emprunts non adaptés et celle des créations lexicales. Nous évoquerons par ailleurs rapidement le cas des emprunts adaptés. Les tokens altérés constituant l'objet de notre étude, nous nous concentrerons sur les travaux s'y référant dans le troisième chapitre.

2.1.1 Détection des entités nommées

Les entités nommées représentent une forte part des inconnus dans les corpus journalistiques (Maurel, 2008). Souvent employées pour désigner les noms propres (noms de personnes, d'entreprises, de lieux ou encore sigles), le terme d'entité nommée peut néanmoins être utilisé de manière étendue, notamment en y incluant les entités numériques (ex. : les dates, les montants...) et les entités temporelles (ex. : événements) ainsi que Sagot et Boullier (2008) le font par exemple.

Maurel (2008), qui s'est particulièrement intéressé à la place des noms propres parmi les inconnus, constate qu'en étiquetant tous les inconnus [1] commençant par une majuscule comme des noms propres, il n'obtient qu'un très faible taux d'erreurs (env. 1%). Cette technique n'est néanmoins vérifiée que sur des textes de type journalistique, pour des noms propres simples et pour le français. En allemand par exemple, la capitalisation des noms mettrait à mal cette méthode.

En détection automatique des entités nommées, les travaux sont séparables en deux catégories distinctes (Nouvel, 2012) : ceux qui sont orientés connaissances (qui s'appuient notamment sur des transducteurs) et ceux qui sont orientés données (qui utilisent plutôt des méthodes d'apprentissage automatique).

Parmi les travaux orientés connaissances, nous pouvons citer ceux qui s'appuient principalement sur des connaissances encyclopédiques. Bunescu et Pasca (2006), Kazama et Torisawa (2007), Balasuriya et al. (2009) ou encore Nothman et al. (2009) utilisent par exemple la Wikipédia. Toutefois, ces approches, qui s'appuient sur des bases ou des inventaires d'entités nommées, sont parfois jugées insatisfaisantes. En effet, les entités nommées appartenant à une classe ouverte, elles ne peuvent pas être listées de manière exhaustive. Pour pallier ce manque, certains

1. Dans son travail, Maurel détecte les inconnus en s'appuyant sur le logiciel Intex Silberztein (1993). Il est à noter que cet outil ne reconnaît pas les entités nommées dans son système ce qui explique la forte proportion de noms propres considérés comme inconnus dans ce travail.

travaux s'aident de transducteurs. C'est ce que font par exemple, Stern et Sagot (2010) qui combinent leur base d'entités nommées (conçue à l'aide de la base GeoNames couplée à la Wikipédia) à des transducteurs pour détecter ces entités nommées en corpus ou Friburger (2002) qui détecte ces entités en utilisant des cascades de transducteurs. On peut aussi mentionner l'approche de McDonald (1996) qui se repose sur deux types d'indices pour les repérer : les indices internes et externes à une entité nommée. Les indices internes correspondent ainsi à des formes qui figurent fréquemment dans des entités nommées (*organisation des Nations Unies, place Stanislas,...*). Les indices externes désignent le contexte d'apparition dans lequel peuvent apparaître les entités nommées.

Les travaux orientés données, quant à eux, s'appuient sur des méthodes d'apprentissage automatique afin d'obtenir des résultats semblables à ceux qu'on pourrait trouver dans des corpus de références. Ces systèmes peuvent ainsi reposer sur différentes techniques telles que le *clustering* (Miller et al., 2004) ou d'autres modèles discriminant comme des modèles de maximum d'entropie (MaxEnt) (Borthwick et al., 1998; Mikheev et al., 1999; Ekbal et al., 2010), des Modèles de Markov Cachés (HMM) (Bikel et al., 1999), des modèles markoviens à maximum d'entropie (McCallum et al., 2000) ou encore des Champs de Markov Aléatoires (CRF) (Lafferty et al., 2001; McCallum et Li, 2003; Raymond et Fayolle, 2010).

2.1.2 Détection des emprunts non adaptés

Nous ne traiterons dans cette section que des emprunts non adaptés. Les emprunts adaptés étant des emprunts dont l'incorporation au sein de la langue cible résulte d'une adaptation morphologique et qui sont souvent sujets à des processus dérivationnels. Nous y reviendrons par conséquent plus tard en section 2.1.4.

Une solution simple pour détecter un emprunt (non adapté), c'est-à-dire un token appartenant à une langue étrangère, serait de s'appuyer sur un lexique de cette langue étrangère. Néanmoins, cette solution comporte quelques désavantages. La prise en compte de tous les lexiques des langues autres que celle traitée risque d'être coûteuse et de générer de nombreuses ambiguïtés. Par ailleurs, rien ne garantit que ces lexiques auront une couverture assez large pour couvrir les différents types de tokens pouvant être empruntés. En effet, le token emprunté peut aussi bien correspondre à un terme courant, à un terme technique non répertorié dans un lexique général, ou encore à une création lexicale. Une autre solution serait de détecter automatiquement la langue d'un inconnu. En effet, s'il apparaît qu'un inconnu correspond à un token d'une autre langue, sa classification devient évidente. Nous nous sommes par conséquent intéressée aux travaux réalisés dans le domaine de la détection automatique de langue. Bien que ces derniers soient ceux qui correspondent au mieux à notre problème, il nous faut préciser qu'ils ne sont généralement pas effectués dans le même contexte que le nôtre. En effet, les mé-

thodes proposées dans ces travaux supposent que l'on dispose de longs textes à traiter et non d'un seul et unique mot.

Ainsi, plusieurs approches proposent d'exploiter une phrase dans son ensemble. C'est par exemple le cas de Johnson (1993) qui exploite la présence des mots grammaticaux dans une phrase ou de Ingle (1976) et Beesley (1988) qui se limitent aux tokens les plus courts d'une langue. Zampieri (2013) propose, quant à lui, d'utiliser un classifieur combiné à des sacs de mots afin de déterminer la langue d'un énoncé. Par ailleurs, on constate que les recherches en identification automatique de la langue ont récemment pris un nouvel essor avec l'analyse de textes plus courts tels que ceux qu'on peut trouver sur Twitter Lui et Baldwin (2014) [2]. Toutefois, ces approches ne permettent pas de déterminer la langue d'un inconnu pris isolément. Le contexte d'un emprunt étant généralement rédigé dans la langue cible, il ne peut être pris en compte dans notre cas.

Il est donc plus pertinent dans notre contexte de se concentrer sur des études qui se limitent aux tokens. De nombreuses méthodes, plus statistiques, peuvent être citées. Il est par exemple possible de s'appuyer sur des n-grammes simples comme Cavnar et Trenkle (1994) et Martins et Silva (2005). Par ailleurs, on constate que sur des données issues du Web, qui sont plus proche de nos données que des textes plus littéraires, Martins et Silva (2005) rapportent une précision variant de 80% à 100% pour la classification de pages parmi 12 langues en utilisant les n-grammes ainsi que quelques heuristiques complémentaires. Ces n-grammes peuvent aussi être associés de l'entropie relative (Sibun et Reynar, 1996) ou combinés à des modèles de Markov (Dunning, 1994).

2.1.3 Détection des créations lexicales

De même que les entités nommées, les créations lexicales représentent une grande partie des inconnus. Il est donc primordial de savoir les distinguer dans un texte. Comme nous l'expliquons dans (Sagot et al., 2013), la majorité des créations lexicales suivent des règles internes à la langue. Ces règles peuvent mettre en jeu soit une unité lexicale et un ou plusieurs affixes (dans ce cas il s'agira de dérivation), soit deux unités lexicales (composition).

De nombreuses études se sont penchées sur la détection de ces créations lexicales. La plupart de celles décrites dans la littérature mettent notamment l'accent sur l'analyse des éléments qui les constituent, en proposant des systèmes aussi bien semi-supervisés que non supervisés. *Dérif* (Hathout et Namer, 2011), un système à base de règles (créées manuellement), détermine par exemple les éléments à partir desquels sont construites des unités lexicales, par dérivation ou par composition. Lovins (1968) propose d'analyser les néologismes en utilisant des algorithmes de racinisation. D'autres encore réalisent cette tâche de manière non supervisée,

2. Un lecteur intéressé peut se référer à la bibliographie proposée par Lui et Baldwin (2014).

en passant par exemple par la notion d'analogie formelle (Lavallée et Langlais, 2011) ou en utilisant un système de segmentation (Goldsmith, 2001; Creutz et Lagus, 2005). Certains travaux montrent par ailleurs qu'il est possible d'ajouter à l'analyse de la construction d'un mot la prédiction de son lemme et de ses traits morphologiques en s'appuyant sur un système d'apprentissage supervisé couplé à de l'analogie (comme le montrent Stroppa et Yvon (2006) et comme nous le verrons en section 4.5.1.3.1), ou, si l'on a préalablement fait usage d'un étiqueteur morphosyntaxique, à l'aide de règles de réécriture (Namer, 2000). Disposer de ces informations supplémentaires a par exemple permis à Dal et Namer (2000) (avec *GéDéRif*, un système à base de règles), ainsi qu'à Tanguy et Hathout (2002) (avec *Webaffix*), de proposer une approche qui, pour chaque forme nouvelle rencontrée, calcule ses dérivés à partir d'un générateur de flexion et vérifie leur validité sur Internet.

À l'inverse des études précédemment citées, on note l'existence de travaux s'appuyant sur d'autres éléments que ceux qui composent une création lexicale. Ainsi on peut par exemple citer Cabré et Nazar (2011) qui proposent d'extraire les néologismes d'un ensemble de corpus journalistiques en se référant à la diachronie de ce dernier (soit à l'évolution de la langue en fonction de la date de publication de chaque article) ou Issac (2011) qui tente de détecter les néologismes en procédant par élimination en écartant de la sorte des tokens qu'il ne veut pas reconnaître comme tels (ex : fautes ou concaténation de tokens). Falk et al. (2014) reprennent d'ailleurs cette dernière idée et ne conservent que les néologismes les plus probables en s'appuyant sur un système d'apprentissage automatique après avoir éliminé à l'aide d'un classifieur les inconnus ne correspondant pas à des néologismes.

En ce qui concerne les mécanismes compositionnels, on peut notamment citer Mathieu-Colas (2010) qui se penche sur la création lexicale par trait d'union, toutefois nous n'avons pas connaissance de systèmes implémentés correspondant à ce travail.

Enfin, on peut noter quelques travaux qui, sans traiter directement de la création lexicale, s'approchent du sujet. En morphologie, plusieurs études se sont ainsi intéressées au regroupement de mots en familles morphologiques [3]. C'est par exemple le cas de Bernhard (2010), qui a mis en place deux systèmes d'apprentissage non supervisés (*MorphoClust* et *MorphoNet*) s'appuyant pour l'un sur de la classification ascendante hiérarchique et pour l'autre sur des algorithmes fondés sur les graphes afin de détecter des relations entre mots de même famille dans des réseaux lexicaux. C'est aussi le cas de Hathout (2010) dont le système *Morphonette* mêle analogie et informations sémantiques extraites de définitions lexicographiques. Nous pouvons encore citer le cas de Baranes et Sagot (2014) qui s'appuient sur des règles morphologiques apprises par analogie afin de rattacher entre elles les entrées dérivationnellement liées d'un lexique pour le français, l'anglais, l'allemand

3. Une famille morphologique est composée de mots partageant une base lexicale commune (*écrit, écrire, écrivain...*).

et l'espagnol. D'autres travaux tentent de prédire les formes potentielles de la langue (Neuvel et Fulop, 2002) ou de compléter automatiquement des quadruplets d'analogie (Lepage, 1998).

2.1.4 Détection des emprunts adaptés

Les emprunts adaptés ne peuvent pas être assimilés à ceux non adaptés. Comme nous l'avons expliqué ci-dessus, ces derniers correspondent à des tokens étrangers adaptés morphologiquement à la langue traitée. Cette *adaptation* leur fait perdre certaines caractéristiques propres aux emprunts adaptés qui auraient pu être nécessaires à leur détection. Pour cette raison, nous avons fait le postulat qu'ils pouvaient être assimilés aux créations lexicales. Ce choix peut néanmoins limiter notre analyse. En effet, un token dérivé trouvé en corpus ne pourra être analysé qu'à deux conditions : (i) notre lexique de référence doit contenir le terme de base dont il est dérivé, (ii) la modification morphologique qu'il a subi doit avoir été réalisée à partir de ce même terme. C'est par exemple le cas de l'emprunt adapté *débugage* qui a été construit à l'aide : (i) du token *débuguer* qui est, à présent, intégré dans quelques lexiques (bien que la version *déboguer* soit préférée) et (ii) de la règle suffixale morphologique « er → age » qui permet désigner, à partir d'un verbe, le résultat de son action — on notera que ces étapes sont suivies de la suppression du « -u- », dispositif graphémique permettant de noter la prononciation correcte de la finale « -guer », désormais inutile. Si l'une de ces deux conditions n'est pas respectée, la détection d'un emprunt adapté à l'aide de méthodes de reconnaissance de création lexicale ne sera plus complètement possible. Par exemple, la reconnaissance du token *hashage* dans *table de hashage* sera plus complexe à réaliser (Walther et Sagot, 2011). Sa forme d'origine *hash*, étant trop spécifique au domaine de l'informatique et peu utilisé seul, ne figure pas dans la langue cible [4]. C'est ainsi malgré cette limite que les emprunts adaptés seront considérés dans cette étude comme des créations lexicales.

2.2 Systèmes par analogie

Parmi les travaux cités ci-dessus, nous nous sommes particulièrement intéressée à ceux qui utilisent l'apprentissage par analogie. Notre système s'appuyant lui aussi sur ce type d'apprentissage, il nous semble important de le détailler ici. Nous définirons par conséquent dans un premier temps les notions d'analogie et d'analogie formelle. Puis dans un second temps, nous nous intéresserons en particulier à ses applications possibles dans des tâches de traitement automatique des langues.

4. Notons que le terme anglais *hash table* peut être repris en français en étant adapté (*table de hashage*.) mais aussi utilisant une traduction phonétiquement proche de la version adaptée (*table de hachage*)

2.2.1 L'analogie

2.2.1.1 Définition et propriétés

Le terme ANALOGIE, provenant du grec ἀναλογία, signifie *proportion* (au sens mathématique). Il désigne ainsi le rapport proportionnel existant entre deux paires d'objets. Pour cette raison, nous emploierons indistinctement dans cette thèse les termes *rapport proportionnel* et *rapport analogique*. Un rapport analogique entre quatre objets est considéré comme valide si le premier de ces objets est au second ce que le troisième est au dernier. Ainsi, nous considérerons que les quatre objets A, B, C et D partagent un rapport de proportionnalité si *A est à B ce que C est à D*. Ce rapport se note :

$$A : B :: C : D$$

Un rapport proportionnel est doublement articulé. Les « : » illustrent ici sa première notion articulatoire : le RATIO. Sa seconde, la CONFORMITÉ, est par ailleurs représentée par « :: ». Ainsi un rapport est analogique entre deux paires d'objets si leurs ratios respectifs sont conformes l'un à l'autre (Lepage, 2015).

Une même analogie peut être écrite de différentes façons en appliquant les propriétés suivantes :

1. La conformité d'une analogie est réflexive. Si on choisit une paire d'objets, son ratio est nécessairement conforme à lui même. $[A : B :: A : B]$ correspond ainsi bien à une analogie valide.

2. Le second et le troisième objet d'une analogie sont interchangeables. Si on a par exemple l'analogie $[A : B :: C : D]$, alors l'analogie $[A : C :: B : D]$ est valide puisque les objets B et C sont substituables l'un à l'autre.

3. La conformité d'une analogie peut être inversée. Ainsi $[A : B :: C : D]$ peut aussi s'écrire $[C : D :: A : B]$. Le symbole « : : » sert ainsi ici de pivot pour réaliser l'inversion.

Ainsi comme l'explique Lepage (2015), ces propriétés permettent de représenter une même analogie de huit manières différentes (cf. table 2.1). Toutefois, une dernière propriété peut être ajoutée : l'inversion des ratios d'une analogie. Cette quatrième propriété nous permettrait ainsi de passer de la forme $A : B :: C : D$, à la forme $B : A :: D : C$. Elle ne permettrait pas de produire d'autres représentations d'une même analogie (elle provoquerait plutôt une redondance dans la création de ces formes), mais positionnerait la propriété de l'inversion à une place centrale (voir la table 4 de Lepage, 2015).

Forme	propriété(s) appliquée(s)
$A : B :: C : D$	_
$A : C :: B : D$	Propriété 2
$B : D :: A : C$	Propriété 2 + Propriété 3
$B : A :: D : C$	Propriété 2 + Propriété 3 + Propriété 2
$C : D :: A : B$	Propriété 3
$C : A :: D : B$	Propriété 3 + Propriété 2
$D : B :: C : A$	Propriété 3 + Propriété 2 + Propriété 3
$D : B :: C : A$	Propriété 3 + Propriété 2 + Propriété 3 + Propriété 2

Table 2.1 – Représentations possibles pour l'analogie $A : B :: C : D$

2.2.1.2 Définition analogie formelle

L'analogie formelle est un cas spécifique d'analogie que nous définissons ici comme impliquant que la relation entre les 4 objets soit graphémique. C'est par exemple le cas de l'analogie : [*transcrire* : *transcription* :: *inscrire* : *inscription*] qui illustre le mécanisme morphologique qui s'applique lors du passage de la première forme à la seconde et de la troisième à la dernière.

Plusieurs définitions ont jusqu'ici été proposées (Lavallée et Langlais, 2011). Hathout (2002), par exemple, se concentre sur les analogies qui partagent uniquement un préfixe commun tandis que Moreau et al. (2007) permettent qu'une analogie puisse à la fois contenir une opération de préfixation et de suffixation. Nous pouvons aussi citer Lepage (1998), qui propose un système plus riche prenant en compte préfixation, suffixation et infixation.

Nous nous attarderons ici sur la définition proposée par Yvon et al. (2004) et reprise dans Stroppa et Yvon (2005, 2006). Cette dernière a l'avantage de décrire l'analogie formelle de manière moins restrictive que ces précédents travaux. Elle généralise notamment la définition proposée par Lepage (1998) et a été réutilisée dans plusieurs études (Lavallée et Langlais, 2011; Rhouma et Langlais, 2014). L'analogie formelle est ainsi définie en s'appuyant sur la notion de décomposition et d'alternance.

On considère ainsi qu'un token peut être décomposé en plusieurs sous-parties, appelées facteurs. Ces facteurs peuvent correspondre à un ou plusieurs caractères. Soit x une chaîne de caractères sur un alphabet Σ. On définit $f(x)$, une factorisation de cette forme, comme une séquence de n facteurs composant cette forme. On dira ainsi que $f(x) = f_x^1 ... f_x^n$ de sorte que $x \equiv f_x^1 \odot ... \odot f_n^x$, le symbole $\odot$ dénotant ici la concaténation [5]. On notera x_i un facteur composant la forme x.

5. On notera que, dans cette définition, la notion de factorisation est purement formelle. Les différents facteurs constituant la chaîne de caractères x ne correspondent donc pas nécessairement à un caractère simple ou à des morph(ème)es.

À cette notion de factorisation s'ajoute la notion d'alternance. Formellement, on considérera que l'on peut identifier un rapport analogique entre quatre formes si et seulement si elles peuvent chacune être décomposées en n facteurs. Si l'on considère deux des quatre formes, certains de ces facteurs seront communs aux deux formes, alors que les autres parties pourront alterner. Soient les quatre formes x, y, z et t. Calculer le DEGRÉ D'ANALOGIE $(x : y :: z : t)$ reviendra ainsi à s'assurer que ces formes puissent respectivement être factorisées en f_x, f_y, f_z et f_t de telle sorte que pour tout i compris entre 1 et d, $(f_y^i, f_z^i) \in \{(f_x^i, f_t^i), (f_t^i, f_x^i)\}$. Le plus petit d que l'on puisse obtenir en choisissant au mieux la factorisation définit le degré de cette analogie, et c'est celui que l'on retient. Ainsi, si on a les quatre formes $x = transcrire, y = transcription, z = inscrire$ et $t = inscription$. Leur rapport analogique (de degré 2) peut être représentée comme ceci : $x_1 = y_1 = transcri$, $z_1 = t_1 = inscri$, $x_2 = z_2 = ption$ et $y_2 = t_2 = re$

$$[\underset{x1}{\underline{transcri}}\ \underset{x2}{\underline{re}} :: \underset{x1}{\underline{transcri}}\ \underset{y2}{\underline{ption}} :: \underset{z1}{\underline{inscri}}\ \underset{x2}{\underline{re}} : \underset{z1}{\underline{inscri}}\ \underset{y2}{\underline{ption}}]$$

Dans un rapport de proportionnalité, chaque forme est donc constituée de d facteurs. La factorisation retenue est ainsi une conséquence du calcul du degré du rapport analogique. Une forme pourra être constituée d'un nombre de facteurs qui différera en fonction des autres formes avec lesquelles on la met en relation analogique. Comme nous le verrons en section 2.2.2.1, différentes méthodes ont été mises en œuvre pour résoudre des rapports analogiques.

Stroppa et Yvon (2006) s'accordent par ailleurs pour préciser que bien que la définition proposée ci-dessus généralise celle de Lepage (1998), elle conserve néanmoins les deux traits suivants :

1. Il est possible qu'un calcul de proportionnalité n'obtienne aucun résultat ou puisse en générer plusieurs.
2. Si t est la forme attendue pour résoudre l'analogie $[x : y :: z : ?]$, alors t est composé des symboles composants y et z qui ne sont pas contenus dans x, et ce, dans un ordre inchangé. Ainsi, toutes les solutions d'une équation analogique seront d'égale longueur : $|x| + |t| = |y| + |z|$

Enfin, il est à noter que toutes les propriétés de l'analogie (présentée dans la section précédente) s'appliquent à l'analogie formelle.

2.2.2 Apprentissage par analogie en TAL

Ces dernières années, l'apprentissage par analogie a été utilisé afin de résoudre plusieurs problèmes distincts posés en traitement automatique de la langue. La mise en œuvre de ce type d'approche suppose l'élaboration d'un système et donc d'algorithmes permettant la détection de relations par analogie. Nous évoquerons donc brièvement en section 2.2.2.1 les diverses propositions faites à ce propos

avant de donner un aperçu, section 2.2.2.2, des différents domaines qui utilisent l'analogie.

2.2.2.1 Systèmes existants d'apprentissage par analogie

L'application de l'apprentissage par analogie peut permettre deux types d'opérations [6] :

1. Résoudre une équation analogique $[x : y :: z : ?]$ en retrouvant la chaîne de caractères t manquante.
2. Rattacher une forme inconnue t à des formes connues en l'intégrant dans une analogie $[x : y :: z : t]$.

La résolution de la première opération suppose de s'aider des éléments connus du triplet de forme (x, y, z) afin de résoudre une équation analogique. Lepage (1998) propose d'aligner les formes x, y et x, z afin de déduire la valeur de t. Pour cela il met en place un algorithme qui prend les deux paires (x, y) et (x, z). Il tente de parcourir ainsi les formes y et z de trouver leurs correspondances avec x. Ainsi pour chaque caractère de ces chaînes, si une correspondance est trouvée on passe au caractère suivant, sinon, le caractère en question est conservé comme un composant de la solution recherchée. Yvon et al. (2004) (cités par Langlais et al. (2009)) préfèrent, quant à eux, s'appuyer sur un système qui repose sur des transducteurs à états finis. Ils s'appuient pour cela sur un théorème affirmant que t peut être la solution de l'équation $[x : y :: z : ?]$ si et seulement si t correspond à la concaténation des formes y et z à laquelle on aurait extrait les caractères de la chaîne x, soit : $\mathrm{t} \in y \odot z \backslash x$. Ils proposent ainsi de concaténer ou d'intercaler de toutes les façons possibles les chaînes de caractères y et z et d'en extraire les lettres contenues dans la chaîne z. Le résultat apparaissant le plus de fois correspondra ainsi à t.

La résolution de la seconde opération pose certaines difficultés. En effet, intuitivement, elle suppose de constituer l'ensemble de tous les triplets pouvant partager avec t une relation analogique. Néanmoins, la génération de tous les triplets (x, y, z) possibles est très coûteuse à réaliser. En effet, parcourir à trois reprises l'ensemble des éléments connus générerait une complexité cubique.

Afin de contourner ce problème, Stroppa et Yvon (2006) proposent de s'appuyer non pas sur un lexique entier, mais sur une sous-partie de ce lexique (appelée *paradigme* dans l'étude) qui serait constitué d'entrées partageant un critère commun. Ce critère peut varier en fonction de la tâche qu'on veut réaliser [7]. Lors de

6. Il est à noter que ces opérations ne sont pas systématiquement réalisables. Elles peuvent aussi bien donner lieu à plusieurs résultats qu'à aucun.

7. Les entrées peuvent ainsi être regroupées en fonction de leur classe flexionnelle si on veut procéder à une tâche d'analyse flexionnelle ou en fonction de leur racine si on préfère effectuer une analyse dérivationnelle.

la recherche de triplets, les deux premières formes (constituant la première paire d'une analogie) devraient ainsi appartenir à un même paradigme. Pour la sélection de la troisième forme du triplet, les auteurs proposent de limiter la recherche de cette forme à un nombre n de paradigmes choisis aléatoirement, l'idée sous-jacente étant que les paradigmes sont « très redondants et interchangeables ».

Langlais et Yvon (2008) se sont aussi penchés sur ce problème de complexité. Ils proposent ainsi de considérer tour à tour chaque entrée x d'un lexique et d'associer cette forme x à notre forme t connue. Cela permet d'introduire une contrainte sur le nombre de symboles que les formes y et z doivent contenir pour que le quadruplet $[x : y :: z : t]$ corresponde à une analogie [8].

Langlais (2009) utilise par ailleurs cette dernière méthode afin d'illustrer l'utilité de l'apprentissage par analogie pour la segmentation de formes en morphèmes. Il s'appuie pour cela sur la définition d'analogie formelle proposée par Yvon et al. (2004) (cf 2.2.1.2) et part du principe que les factorisations permettant la détection d'une analogie correspondent la majorité du temps aux frontières morphologiques d'une forme. Ainsi, il tente pour chaque forme d'extraire toutes ses factorisations possibles. Il compte ensuite le nombre d'occurrences de toutes les factorisations qui lui permettent de créer une relation par analogie. Celle qui obtient le nombre d'occurrences le plus élevé est ensuite considérée comme celle correspondant au découpage morphologique le plus probable. Cet algorithme a ensuite été repris dans plusieurs travaux (Lavallée et Langlais, 2011; Rhouma et Langlais, 2014).

Hathout (2008), quant à lui, propose un système d'apprentissage qui n'utiliserait que les lexèmes d'un lexique [9] afin d'apprendre automatiquement les structures morphologiques des entrées d'un lexique. Il propose pour cela de procéder en deux temps. Dans un premier temps, il tente de regrouper des mots morphologiquement reliés (en s'appuyant sur de la similarité lexicale [10], puis, dans un second temps, il valide ces liens à l'aide de l'analogie. Cette vérification analogique est réalisée à l'aide des distances d'éditions de Levenshtein (1966). Il calcule puis compare ainsi les signatures d'édition de deux paires de formes (x, y) et (z, t). Chaque signature illustre ainsi les modifications à effectuer pour passer d'une chaîne à une autre. Hathout n'autorise ici que les 3 opérations : l'insertion (I) d'un caractère, sa suppression (D) et la substitution (S) d'un caractère par un autre. La correspondance de deux caractères ou séquence de caractères (M) n'est pas considérée comme une opération. Ces modifications sont ainsi représentées par des ensembles de trois éléments constitués de la nature de cette

8. Il est à préciser que cette stratégie ne peut être efficace qu'à deux conditions : (1) Le nombre de quadruplets dont il faut vérifier le degré de proportion doit être inférieur au nombre de triplet qui auraient pu être créés, (2) la paire (y, z) doit pouvoir être trouvée efficacement.

9. Il s'autorise ici l'utilisation d'informations sémantiques (des définitions lexicographiques, synonymes, etc.) et formelles (graphémiques, phonologiques, etc.).

10. Pour ce faire, Hathout propose de rattacher les entrées d'un lexique en s'appuyant sur leurs propriétés sémantiques (extraites dans leurs définitions) et formelles (n-gramme de caractères extraits de leurs lemmes respectifs).

modification (D, S, I ou M), de la chaîne prise en entrée de cette modification et de celle prise en sortie. S'il s'avère que ces deux signatures sont identiques, alors ces deux paires sont considérées comme formant une analogie formelle. Par exemple, les distances d'édition des paires *(fructueux :infructueusement)* et *(soucieux :insoucieusement)* sont identiques et peuvent être représentées comme suit : $\sigma(fructueux : infructueusement) = \sigma(soucieux : insoucieusement) =$ ((I,ϵ,i), (I,ϵ,n), (M,@,@), (S,x,s), (I,ϵ,e), (I,ϵ,m), (I,ϵ,e), (I,ϵ,n), (I,ϵ,t)) [11]. Ce constat nous permet d'affirmer que ces deux paires de formes entretiennent bien une relation d'analogie.

2.2.2.2 Domaines exploitant l'analogie

Dans un premier temps, l'apprentissage par analogie a été beaucoup employé en analyse et en génération au niveau morphologique. Ainsi Tanguy et Hathout (2002) et Hathout (2002) ont montré la possibilité de relier des formes qui étaient dérivationnellement liées en combinant analogie et informations sémantiques. Stroppa et Yvon (2005, 2006) prédisent, quant à eux, le lemme et les traits morphosyntaxiques d'une forme connue à l'aide de l'analogie. Ils montrent par ailleurs qu'il est possible d'apprendre automatiquement ces mêmes informations pour une forme inconnue. Langlais (2009), dans son étude, met par ailleurs en relief les points de rencontre entre morphologie constructionnelle et analogie formelle en appuyant son étude sur les entrées du lexique Celex (Burnage, 1990).

De nombreux travaux ont par ailleurs appliqué la notion d'analogie à la traduction automatique. Lepage et Denoual (2005) figurent ainsi parmi les premiers à s'y être essayés. Leur système et les améliorations de ce dernier ont été évalués à plusieurs reprises dans le cadre de la campagne d'évaluation IWSLT (Lepage et Denoual, 2005; Lepage et al., 2008, 2009). D'autres se sont intéressés à la traduction de termes plus spécifiques. Ainsi, nous pouvons par exemple citer Langlais et Patry (2007, 2008) et Denoual (2007) qui se sont concentrés au même moment sur la traduction des mots inconnus d'une langue source vers une langue cible, ou encore Langlais (2013) qui propose de faire la translittération de noms propres anglais en mandarin à l'aide d'analogie formelle. L'apport de l'analogie dans ce domaine a notamment été souligné par Rhouma et Langlais (2014) qui comparent les systèmes de traduction par analogie à ceux s'appuyant uniquement sur la phrase.

Nous pouvons enfin citer quelques travaux qui ont tenté d'appliquer la notion d'analogie à d'autres tâches propres au traitement automatique de la langue. Yvon (1997, 1999) emploie ainsi l'analogie pour convertir de l'écriture orthographique en

11. Dans nos travaux (cf. section 4.5.1.3.1 et chapitre 5), nous avons fait le choix de représenter ces modifications plutôt sous la forme de règles. Cette signature d'édition aurait ainsi pu être représentée sous la forme de deux règles, une de préfixation (ϵ→in) et une seconde de suffixation (x$→sement$).

transcription phonétique. Moreau et al. (2007) et de Claveau et L'Homme (2005) l'appliquent à l'extraction terminologique. Federici et al. (1997) tentent de faire de la désambiguïsation sémantique par analogie. Enfin Gosme et Lepage (2011) l'utilisent pour mettre en place un système de lissage de modèle n-gramme.

L'apprentissage par analogie, permettant de capturer les régularités de la langue, peut ainsi réellement être utile. Il peut par exemple nous permettre d'étudier morphologiquement les inconnus afin de déterminer s'ils respectent le mécanisme de la langue traitée. Si c'est le cas, il pourrait parfaitement s'agir de néologismes ou encore d'erreurs dans la flexion d'une forme connue. Nous reviendrons notamment dessus au chapitre 4.5.3. Par ailleurs, même s'il n'existe pas de travaux à notre connaissance qui appliquent l'apprentissage par analogie à la normalisation ou à la correction de texte, ce type d'apprentissage pourrait parfaitement être adapté à notre problématique.

CHAPITRE 3

Détection et interprétation automatique des altérations

Sommaire

Les tokens dont la graphie a été altérée correspondent, comme nous l'avons souligné dans les premiers chapitres, à l'une des grandes catégories constituant l'ensemble des inconnus (cf. par exemple Dister et Fairon, 2004). Dans notre travail, en plus de les reconnaître, nous voulons retrouver leur graphie initiale. Ces **altérations** ne correspondent pas nécessairement à des fautes d'orthographe ou à un inconnu. Ce chapitre a ainsi pour objectif de définir ce qu'on appelle altération, ce que nous ferons dans la section 3.1. Nous nous pencherons ensuite sur les méthodes existantes permettant la détection de ces tokens altérés et l'association de ces derniers à leurs formes initiales. Parmi tous les systèmes existants se distinguent deux approches : la normalisation et la correction automatique. Nous évoquerons en section 3.2 les travaux existants pour ces deux approches.

3.1 Typologie des altérations

Nous appelons ici ALTÉRATION l'une des variantes graphiques possibles d'un mot appartenant à la zone floue de notre lexique réel [1]. Cette entité peut ainsi correspondre aussi bien à un token mal orthographié (par exemple : *demin*) qu'à une simplification graphique volontaire (par exemple : *2m1*). Cela peut ainsi donner lieu une grande diversité de tokens distincts. De nombreuses classifications des différents types d'altérations ont d'ailleurs été déjà été proposées (voir par exemple Ren et Perrault, 1992; Anis, 2003; Véronis et Guimier de Neef, 2006; Ringlstetter et al., 2006). Cette diversité se ressent plus ou moins en fonction du corpus observé. Ainsi, un message produit sur un réseau social aura, par exemple, plus de chances de contenir des altérations qu'un article journalistique. De même, on utilisera plus aisément des abréviations sur des forums que dans un e-mail ou sur un site d'avis, ces derniers étant des canaux plus formels. C'est pourquoi certains travaux se sont concentrés sur les altérations propres à certains types de corpus tels que les SMS (Anis, 2001; Fairon et al., 2006), les tchats (ou salons de clavardage en français canadien) (Tatossian, 2011) ou encore les forums et les réseaux sociaux (Seddah et al., 2012).

Dans leur travail, Boissière et al. (2007) proposent de décrire les altérations (plus spécifiquement les erreurs) en s'appuyant sur deux critères : (i) la nature de l'altération et (ii) son analyse et sa conséquence au niveau linguistique (phonologique, morphologique, syntaxique). Nous pensons, en effet, que la diversité des altérations ne peut être décrite qu'en prenant en compte ces différents critères. Toutefois, nous nous appuierons non pas sur deux, mais sur trois critères :

1. L'origine de l'altération : Qu'est-ce qui a provoqué cette altération ?
2. Le mécanisme d'altération : Quel procédé a-t-on utilisé pour réaliser cette altération ?

1. On fait ainsi référence ici aux inconnus générés en passant par la cinquième flèche de notre figure 1.3 introduite en section 1.3.3.1.

3. Le résultat de l'altération : Quelles sont les caractéristiques de l'altération produite ?

Nous reprendrons chacun de ces trois critères dans la suite de cette section afin d'affiner la notion d'altération et de justifier leur grande diversité.

3.1.1 Origine des altérations

Le nombre d'altérations réalisables sur des tokens appartenant à notre lexique peut potentiellement être illimité (ex : *maintenant* → *mintenant, mnt, mtn, maintnant, maintenan, m1tenant, m1tnant, maintenaaaaant...*). Bien que les raisons qui ont pu motiver ces altérations expliquent en partie cette grande diversité, les lister et les détailler nous est impossible. Malgré cela, les tokens altérés peuvent être divisés en deux catégories : ceux qui ont été réalisés sciemment par le locuteur et ceux qui ont été produits non volontairement.

Un token altéré de manière non intentionnelle est généralement le fruit d'une faute de frappe [2], d'une erreur d'inattention ou encore d'une méconnaissance du locuteur concernant la norme du token en question dans un contexte donné.

Les tokens modifiés volontairement, quant à eux, sont réalisés à de multiples desseins. Le locuteur peut tout d'abord chercher, à travers cette altération, à produire un jeu de mots. On peut ainsi citer pour exemple le titre d'un colloque sur l'orthographe française : « *le français m'a tuer* » (Didier et al., 2005) ou encore les histoires du prince Motordu écrites par Pef qui contiennent de multiples paronymes [3]. Il peut aussi procéder à cette altération dans un souci d'économie d'espace (lorsque la taille d'un message est limitée [4]) ou dans un souci de rapidité (pour effectuer une prise de notes par exemple). On peut encore citer les altérations réalisées dans le but d'accentuer une émotion ou un sentiment. Ces types d'altérations connaissent depuis quelques années un pic d'utilisation avec l'essor des NOUVELLES FORMES DE COMMUNICATION ÉCRITE (NFCE) telles que les SMS, les e-mails, les messageries instantanées (*chat*), etc.

L'origine d'une altération étant ainsi déterminante dans la création de cette dernière, plusieurs études ont choisi de s'appuyer dessus afin de proposer des classifications. Celle proposée par Jacquet-Pfau (2001), par exemple, se concentre sur les *erreurs*. Elle divise ces dernières en trois catégories : les erreurs de compétence (méconnaissance de la norme), celles de performance (ex : faute typographique) et les erreurs dites « système » (mots inconnus qui ne sont pas pour autant des erreurs

2. Ces fautes sont aussi appelées *fautes de proximité clavier*.

3. La paronymie désigne un rapport lexical existant entre deux tokens dont les sens diffèrent, mais qui ont la particularité de partager une graphie et/ou une phonétique très semblable (ex. : *embrasser/embraser* ou *balade/ballade*).

4. Le réseau social *Twitter* par exemple permet d'envoyer des messages contenant un nombre de caractères limités.

— ils peuvent être provoqués par de l'incomplétude lexicale ou correspondre à des emprunts par exemple). Ainsi, dans son travail, Jacquet-Pfau (2001) s'appuie bien sur la *nature* de ces erreurs pour les classifier. Ce choix nous semble discutable. En effet, à notre niveau, même si nous pouvons émettre certaines suppositions, nous ne pouvons déterminer de façon certaine et dans tous les cas la raison qui a poussé un locuteur à produire une altération. La provenance de cette dernière est difficile à déterminer sans informations complémentaires. Si nous prenons par exemple le token mal orthographié *ereur*, on ne peut affirmer clairement que cette altération ait été provoquée par une méconnaissance de la langue. Il pourrait aussi s'agir uniquement d'une faute de frappe. Par ailleurs, on peut noter qu'un utilisateur réitérera plus aisément des altérations lues et vues. Ainsi, si ce dernier est habitué à lire des forums, sources de multiples altérations, il aura tendance à les reproduire. Il ne s'agit plus, dans ce cas, ni de fautes typographiques ni d'altérations remettant en cause une compétence (l'utilisateur en question peut s'en apercevoir après une simple relecture), mais plutôt d'une modification provoquée par un mécanisme de mimétisme. Ce dernier type d'altération est ainsi souvent considéré comme dû à la compétence du locuteur malgré que ce ne soit pas réellement le cas.

3.1.2 Mécanismes d'altération

En première approche, on peut décrire le mécanisme dont découle l'altération d'un token comme l'application d'une ou plusieurs transformations sur un caractère [5] ou sur un ensemble de caractères composant ce token. Ces transformations peuvent elles-mêmes se décomposer en une ou plusieurs opérations élémentaires (Damerau, 1964) : l'insertion d'une lettre, son inversion avec une autre, sa substitution et sa suppression. Ces opérations élémentaires peuvent ensuite être appliquées sur un caractère, ou un ensemble de caractères. En pratique, ces mécanismes d'altération peuvent être précisés. Nous avons donc repris ci-dessous chacune de ces opérations afin de les détailler.

1. Insertion
   Cette opération consiste à ajouter un caractère au sein d'un token (ex. : *opérations*→*opér**a**ations*). En fonction du contexte dans lequel elle s'applique et du nombre de caractères concernés par cette dernière, elle permettra de réaliser différents types d'opérations. Ainsi, l'ajout d'une lettre identique à une autre de son contexte permettra de réaliser un doublement de cette lettre (ex. : *faim*→*faaim* ou *balade*→*ballade*). L'ajout de cette même lettre à deux reprises ou plus consistera à ÉTIRER cette lettre (ex. : *non*→*nooooooon*). Si toutefois elle n'est pas identique, mais proche sur le clavier d'une de ses lettres voisines, alors on parlera d'un ajout dû

5. Le terme « caractère » peut correspondre ici à une lettre, à un signe diacritique, à un symbole (ex. : signe de ponctuation) ou encore à un espace.

à la PROXIMITÉ CLAVIER (ex : *pause→pausde* ou *gars→gares*). Enfin si la lettre ajoutée ne correspond à aucun de ces deux cas, on parlera simplement d'AJOUT.

2. Inversion
   L'inversion est une opération qui permet d'inverser la place de deux caractères au sein d'un token (ex. : *opérations→opé**ar**tions*). De même que pour l'insertion, on distingue le cas où ce sont deux lettres voisines sur un clavier qui sont inversées (ex. : *plume→lpume* ou *mail→mali*) du cas où elles ne le sont pas (ex. : *blanc→blacn* ou *une→nue*). Par ailleurs, l'opération d'inversion peut permettre d'intervertir deux ensembles de caractères (*louche→chelou*).

3. Substitution
   L'opération de substitution consiste à substituer un caractère avec un autre au sein d'un token donné (ex. : *opérations→opé**t**ations*). Elle est utilisée dans de nombreux cas. On peut ainsi l'utiliser pour remplacer un caractère seul par un autre qui serait :

   - à proximité sur le clavier (ex : *pas→pzs* ou *dalle→salle*) ;
   - identique au premier avec un symbole diacritique ajouté ou supprimé [6] (ex. : *français→francais* ou *élevé→élève*) ;
   - similaire au niveau graphique (ex : *couleur→cou1eur* ou *soi→sol*) ;
   - identique au premier phonétiquement parlant (ex : *bisous→bizous* ou *sa→ça*) ;
   - sans lien avec le premier (ex. : *viande→viafde* ou *pain→vain*).

   Cette opération est aussi applicable sur deux ensembles de caractères qui partageraient une même phonétique. Ainsi, les chaînes de caractères *eau* et *au* peuvent par exemple être remplacées par la lettre *o* (*aussi→ossi* ou *saut→sot*). Il est à noter que ce type de substitution phonétique peut être utilisé volontairement par un locuteur dans l'unique but de réduire la taille d'un mot (comme expliqué précédemment dans la description de la nature d'une altération section 3.1.1). Cette méthode peut ainsi être appliquée en utilisant le son représenté par des lettres ou des combinaisons de lettres (*quoi→koi*), mais aussi en utilisant la valeur phonétique d'un caractère seul pour leurs valeurs phonétiques (*c'est→C*, *demain→2m1* ou encore *à plus→a+*). Par ailleurs, cette méthode permet de procéder à la décomposition phonétique d'un token donné (*moi→moua*), le but de cette opération

6. Une erreur sur un diacritique peut être considérée comme une opération de substitution d'une lettre avec une autre, lorsque ces lettres (avec et sans un signe diacritique) correspondent toutes deux à deux touches distinctes sur le clavier (« é » et « e ») ou, comme une opération d'ajout ou de suppression si la lettre accentuée ne peut être réalisée qu'en combinant la touche de la lettre non accentuée et celle de l'accent concerné(« ê » et « e »).

consistant dans ce cas non plus à réduire un token, mais à décomposer les sons qui le composent.

Enfin, nous pouvons noter que l'opération de substitution est parfois utilisée pour remplacer un ensemble de caractères, porteur des traits morphologiques (genre, nombre...) d'un token donné, avec un autre. C'est par exemple le cas des altérations *chevaux→chevals*, *gare→garent* ou encore de *closez*[7]. Ces dernières substitutions ainsi produites peuvent avoir des conséquences sur la compréhension de la phrase et sur l'interprétation du token.

4. Suppression
Enfin, l'opération de suppression consiste à supprimer un caractère. Considérée comme la contrepartie de l'insertion, elle est, elle aussi, très usitée. Elle permet ainsi l'application de l'opération inverse du doublement d'une lettre (ex. : *rattrapage→ratrapage* ou *ballade→balade*) ou de la suppression simple d'un ou plusieurs caractères[8] (ex : *simple→smple* ou *pain→pin*). Cette opération est notamment utilisée pour réduire un token donné. Pour cela, on peut par exemple procéder des manières suivantes :

   - en supprimant les lettres finales : c'est notamment le cas des *e* (*grav→grave*) ou des consonnes muettes, les mutogrammes (*pa→pas*) ;
   - en supprimant les lettres n'appartenant pas au squelette consonantique d'un token (*tout→tt* ou *toujours→tjs*) ;
   - en supprimant les caractères composant la partie droite ou gauche d'un token (*numéro→num* ou *bouteille→teille*).

   Un ensemble de tokens peut ainsi être réduit en utilisant l'opération de suppression à plusieurs reprises. De nombreux caractères sont parfois supprimés afin de ne conserver que les premières lettres de chacun de ces tokens et de les retranscrire sous forme de sigle (*mort de rire→ mdr*).

Ces quatre opérations peuvent être combinées, ce qui peut générer des altérations très éloignées de leurs formes d'origine. C'est par exemple le cas du token *tkt* (abréviation de *t'inquiète* pour dire *ne t'inquiète pas*) qui peut se construire en utilisant à la fois des opérations de suppression et de substitution phonétique. On peut aussi citer les procédés d'assimilation phonétique (*je suis→chui*), qui mélangent opérations de substitution et de suppression, ou encore ceux d'agglutination, qui se constatent généralement sur un ensemble de tokens, mêlant ainsi plusieurs mécanismes évoqués précédemment (« *s'il te plaît je (ne) vais pas le*

7. Le verbe *clore* est un verbe défectif.

8. Comme nous l'expliquions précédemment, une faute sur un diacritique peut être considérée comme un cas particulier d'insertion ou de suppression de lettre (la lettre « ê » étant tapée au moyen de deux touches « ^ + e » et non d'une seule).

faire » → « *steplé j'vépa l'fair* » [9]). Ces deux derniers procédés offrent notamment au locuteur la possibilité de transcrire la prononciation orale relâchée.

3.1.3 Tokens résultants de ces altérations

Lorsque l'on obtient le résultat simple d'une altération, on ne connaît pas nécessairement sa nature et les mécanismes qui ont permis sa construction. Ces derniers ne sont pas systématiquement déductibles. Analyser le résultat d'une altération consiste donc à observer un token une fois transformé. La première question qu'on peut se poser est donc : ce token altéré est-il connu du lexique de référence ?

Si un token altéré ne figure pas dans un lexique de référence, on ne peut affirmer de manière certaine sa provenance et par conséquent si elle a été réalisée intentionnellement ou non. Toutefois, certaines altérations produites volontairement utilisent des mécanismes suffisamment particuliers pour qu'un système puisse les détecter. C'est par exemple le cas des tokens étirés. La reduplication d'une lettre un nombre important de fois (*heiiiiiiiiin*) peut, par exemple, nous permettre de retracer aisément ses mécanismes de construction et son origine par la même occasion. On peut aussi citer le cas des tokens décomposés (*in-cro-ya-ble*) ou encore celui des tokens abrégés (*tjs*). Leur régularité s'explique notamment par leur usage relativement fréquent dans certains types de corpus. C'est d'ailleurs notamment pour cette raison que de nombreux systèmes se sont concentrés sur les altérations produites dans les NFCE. Une partie de ces dernières s'appuient sur ces normes de construction implicites et utilisent des mécanismes souvent distincts de ceux qui sous-tendent la production d'une faute lexicale. Dans le cas où un système est incapable de retracer l'origine et le mécanisme de construction d'une altération, il aura souvent tendance à considérer cette dernière comme une faute lexicale, c'est à dire comme une erreur réalisée sur un token connu du locuteur et qui produit un token absent de notre lexique de référence.

Lorsqu'une altération figure dans un lexique de référence, cela signifie que la modification de ce token a produit un autre token de ce même lexique. Ces altérations, détectables uniquement en contexte, sont souvent considérées comme des erreurs grammaticales. Néanmoins, sans information complémentaire, il est très complexe de savoir si cette altération était voulue ou non. En effet, un token altéré partageant la même phonétique que sa forme initiale pourra, par exemple, aussi bien correspondre à une erreur d'homophonie qu'à un simple jeu de mots.

9. Il est à préciser dans cet exemple que bien que le *e* final de **faire** ait été supprimé, ce n'est pas le cas des *e* des mots *je* et *le* qui eux ont été élidés.

3.2 Traitement automatique des altérations

La présence d'altérations au sein d'un texte peut avoir de nombreux impacts sur sa compréhension et sur la qualité des analyses en traitement automatique de la langue auxquelles il est susceptible d'être soumis. Ces outils étant très souvent conçus pour analyser la langue sous sa forme standardisée, leurs performances et leurs résultats se retrouvent par conséquent assez dégradés dès qu'il s'agit de traiter des textes plus bruités (voir par exemple Foster, 2010). Dans notre cas, les tokens modifiés peuvent gêner la détection de sentiments ou l'extraction d'informations. Repérer ces altérations et être capable de retrouver leur forme initiale est ainsi essentiel pour assurer le bon fonctionnement de cet outil.

De nombreux systèmes ont été implémentés dans ce but. Parmi eux se démarquent ceux qui ont pour objectif de normaliser un texte et ceux qui cherchent à le corriger. Bien qu'ils tentent tous deux d'améliorer la qualité d'un texte, ces deux types de systèmes ne partagent pas pour autant les mêmes objectifs. La NORMALISATION d'un texte est une approche visant à prétraiter un texte en préparation d'un traitement aval. Elle peut aussi bien être destinée à améliorer la qualité du texte en proposant une orthographe plus normalisée aux tokens qui ne le sont pas, qu'être implémentée dans le but de remplacer des altérations (volontaires ou non) par des tokens plus standards. Selon le traitement aval envisagé, la normalisation peut, dans certains cas, se satisfaire de remplacer un token altéré par un autre qui n'en serait pas la correction *normée* mais partagerait le même lemme que ce dernier. Par exemple, si le traitement aval repose sur l'identification de lemmes, retrouver le token exact correspondant à la correction de l'altération n'est pas nécessaire. La CORRECTION AUTOMATIQUE, quant à elle, se définit comme une tâche qui tend à supprimer toutes les altérations pouvant être désignées comme fautes en les remplaçant par le token *normé* attendu, sans pour autant toucher à ceux réalisés volontairement. Pour ce faire, elle doit dans un premier temps détecter les tokens mal orthographiés, puis les remplacer par leurs pendants bien orthographiés.

Contrairement à une tâche de correction automatique qui se réalisera souvent de la même manière, quel que soit le but de son application, la normalisation d'un texte se fera ainsi plutôt en fonction du système appliqué à sa suite. Si elle est réalisée dans l'optique de procéder à une opération de débruitage, alors ses critères de normalisation seront adaptés à ce système. Dans notre cas, nous avons besoin d'un outil apte à prétraiter un texte bruité avant son passage dans un système qui s'appuierait principalement sur les lemmes des tokens qui composent le corpus à analyser, cas évoqué ci-dessous. En guise de prétraitement, on pourrait donc aussi bien procéder à une correction simple de ce texte qu'à sa normalisation à la morphologie flexionnelle près, ce qui reviendrait à proposer, pour chaque

altération, soit sa version corrigée, soit un token qui lui est flexionnellement lié [10]. Ainsi, comme le montre la table 3.1, les sorties que pourraient avoir des systèmes de correction et de normalisation, dans notre tâche, différeront [11].

	Correction	Normalisation
Vous ***téléphonnez***	*téléphonez*	*téléphonez, téléphoniez, téléphoner,...*
travaile	*travail, travaille*	*travail, travails, travaille, travailler,...*
Ils ***son*** *là*	*sont*	*sont*
Elle est ***joli***	*jolie*	—
le ***04 fév. 88***	*04 fév. 88*	*DATE+04_ février_ 1988*
c'est ***looooong***	— ou *long*	*long*
c'est ***in-cro-ya-ble***	—	*incroyable*
tkt	—	*t'inquiète*

Table 3.1 – Exemples de sorties possibles pour des tâches de correction et de normalisation (un tiret long indique que l'on conserve la forme initiale inchangée.)

Cette distinction entre correction et normalisation peut sembler simpliste. En effet, si l'on prend l'exemple d'un système de conversion de texte en langue dite sms vers une langue plus normalisée, on le considérera ici comme une tâche de normalisation puisqu'il s'agira de traiter principalement des altérations qui seront en grande majorité volontaires. Toutefois, cette tâche pourrait tout aussi bien être prise comme un système hybride de correction et de normalisation.

Il est enfin à noter que, quel que soit le type de système choisi, le traitement automatique des altérations est souvent composé de 3 sous-parties : (i) la détection des altérations, (ii) la génération de candidats de normalisation ou de correction correspondant à ces dernières et (iii) la sélection du ou des candidats les plus probables pour chacune. Au vu des multiples études existantes dans ces deux domaines, nous ne nous attarderons pas successivement sur ces trois étapes lors de notre description de ces travaux. Nous y reviendrons toutefois dans la mise en place de notre système (section 4).

Les sous-sections 3.2.1 et 3.2.2 ci-dessous ont pour principal objectif de donner respectivement un éventail des systèmes existants pour la normalisation puis pour la correction.

3.2.1 Systèmes de normalisation

La normalisation peut être utilisée dans deux buts distincts. Initialement, elle était employée pour mieux appréhender les textes contenants des formes telles que des

10. Une altération peut, ainsi, aussi bien avoir un ou plusieurs candidats de normalisation corrects.

11. Il est à noter dans le cas d'une normalisation réalisée à la morphologie flexionnelle près qu'il est inutile de normaliser des fautes d'accord.

dates, des nombres, des acronymes ou encore des formes capitalisées (Mikheev, 2000; Sproat et al., 2001). En effet, bien que ces dernières ne correspondent pas à des altérations, leur graphie peut varier. Elles demeurent par conséquent parfois complexes à détecter et à prendre en compte. Les normaliser permet, par exemple, d'obtenir des formes analysables automatiquement pour procéder à du référencement. Puis, avec l'essor des NFCE et, plus spécifiquement des messages provenant des réseaux sociaux (Han et Baldwin, 2011) ou de type texto (Choudhury et al., 2007), la normalisation s'est découvert un nouvel emploi. Ainsi, plutôt que de n'être utilisée que pour harmoniser la représentation de certains types d'entités, elle a aussi permis l'amélioration de textes très altérés. Cette tâche peut aussi bien être appliquée dans l'unique but d'obtenir des textes de meilleure qualité qu'afin de passer en prétraitement d'autres tâches peu performantes face à un corpus bruité (voir par exemple Hassan et Menezes, 2013; Seddah et al., 2012).

Avec ce second objectif, la normalisation aspire ainsi à retranscrire un texte généralement bruité en utilisant une graphie plus standardisée. Les systèmes de normalisation s'intéresseront par conséquent plus aux altérations susceptibles de modifier le sens d'un texte. Les fautes d'accord n'étant pas réellement gênantes pour la compréhension, ne sont usuellement que peu traitées par ce type de système [12]. Les altérations concernées demeurent toutefois nombreuses et diverses dans les corpus de type NFCE. Leur graphie peut par ailleurs être très éloignée de leur forme initiale et faire ainsi référence à plusieurs formes distinctes. C'est par exemple, le cas du token *m* dans les énoncés suivants : « *je t'm* » et « *c'est m pa vrai* ».

La problématique posée par la normalisation, faisant l'objet de nombreux travaux et de projets annexes [13], a été abordée de multiples manières. Selon Kobus et al. (2008), la normalisation peut être abordée sous trois angles distincts : en représentant cette tâche comme une tâche de reconnaissance vocale [14], de traduction ou encore de correction automatique. Nous reprendrons ci-dessous cette classification afin de donner un rapide aperçu des différents travaux qui ont été réalisés dans ce domaine.

12. Il est à préciser que ce point est nullement gênant si un système de normalisation est appliqué au préalable comme prétraitement dans le but d'améliorer les résultats d'un outil qui ne s'appuieraient que sur la forme lemmatisée des tokens composants le texte à analyser.

13. Nous pouvons notamment citer le projet *Tweet NLP* (`http://www.ark.cs.cmu.edu/TweetNLP/`) qui se concentre sur les tweets ou encore le projet *sms4science* (`http://www.sms4science.org/`) qui se focalise sur les SMS.

14. Notre emploi ici du terme de « reconnaissance vocale » suit la littérature. À proprement parler, les techniques employées sont plutôt des techniques de phonétisation, certes employées en reconnaissance vocale, mais il n'est naturellement pas question de reconnaître le moindre signal vocal audio.

3.2.1.1 Normaliser via des techniques de reconnaissance vocale

Les messages textos, SMS, sont souvent écrits de manière très oralisée. Aborder ces messages en les phonétisant dans le but de retrouver ensuite leur graphie normée peut ainsi s'avérer plus simple. La normalisation pourrait ainsi être appréhendée comme une tâche de *reconnaissance vocale*. C'est, par exemple, la solution qui a été choisie par Kobus et al. (2008) pour traiter des SMS rédigés en français. Leur système se décompose en plusieurs étapes : ils cherchent dans un premier temps à transformer un message SMS en un treillis de phonèmes qu'ils convertissent ensuite en un treillis de mots pour, enfin, extraire de ce dernier la séquence de mots la plus probable à l'aide d'un modèle de langue.

3.2.1.2 Normaliser via des techniques de traduction

Une seconde intuition que l'on peut avoir lorsque l'on souhaite normaliser un texte très bruité serait de ne plus considérer ce dernier comme altéré, mais comme un texte rédigé dans une autre langue. Le normaliser reviendrait alors à réaliser une opération de *traduction* dans laquelle la langue source serait la langue utilisée par le locuteur sous sa forme altérée et la langue cible sa version normalisée. C'est par exemple le choix qu'ont fait Vienney et Melian (2004) qui considèrent ainsi le langage dit « texto » comme une langue à part entière et qui ont par conséquent choisi de le traduire (vers du français standard) en s'appuyant sur des informations morpho-syntaxiques et sémantiques. Aw et al. (2006) sont eux aussi partis de ce postulat afin de normaliser à l'aide d'un traducteur statistique des SMS en anglais standard. Pennell et Liu (2011) ont, quant à eux, mis en place un système de traduction s'appuyant sur un modèle de traduction pour générer des candidats de normalisation et un modèle de langue trigramme permettant de réaliser une sélection parmi ces candidats.

Par ailleurs, nous pouvons aussi citer Beaufort et al. (2010) qui proposent un système hybride, combinant des méthodes de correction et de traduction, afin de normaliser des messages SMS. Pour ce faire, ils s'appuient notamment sur la détection d'unités de texte non ambiguës qu'ils ne souhaitent pas normaliser (les smileys par exemple) et sur des techniques d'apprentissage par alignement en se reposant sur des corpus parallèles (corpus constitués de messages SMS et de leur retranscription).

3.2.1.3 Normaliser via des techniques de correction automatique

La correction automatique est l'une des tâches qui se rapproche le plus de la normalisation. Elle tend, elle aussi à améliorer la qualité d'un texte, mais pour cela elle se concentre sur les tokens dits erronés. Aborder la normalisation de cette manière permet entre autres de ne plus analyser systématiquement les phrases

dans leur ensemble, mais plutôt en étudiant chaque mot un à un en partant du principe que la majorité des tokens n'ont pas été altérés. Plusieurs approches ont été utilisées dans ce sens en normalisation.

Certaines se contentent d'ajouter des ressources lexicales à des systèmes déjà existants afin d'améliorer leurs performances face à des textes bruités. Nous pouvons par exemple illustrer ce cas avec les travaux réalisés par Guimier de Neef et Fessard (2007) pour adapter le logiciel TiLT [15] aux SMS. Ces derniers ont ainsi choisi d'ajouter à leur système initial des ressources linguistiques spécifiques aux SMS (ex. : lexique d'abréviations). Il est toutefois à noter que l'ajout de ressources est souvent utilisé en combinaison à d'autres systèmes. Seddah et al. (2012) tentent eux aussi de normaliser leur corpus afin de l'annoter morpho-syntaxiquement par la suite. Pour ce faire ils utilisent un ensemble de ressources linguistiques permettant de remplacer une altération (détectée à l'aide d'expressions régulières ou telle quelle) le temps de l'analyse.

De nombreux travaux ont par ailleurs cherché à faire de la normalisation en s'inspirant de techniques de correction plus statistiques. Choudhury et al. (2007) ont implémenté un système de correction en s'appuyant sur un modèle de Markov caché supervisé qui prend en compte la graphie et la phonétique des formes afin de traiter les abréviations et autres altérations typographiques. Cook et Stevenson (2009) ont, par la suite, étendu ce travail en utilisant un modèle d'erreur non supervisé prenant en compte les différents mécanismes de formations d'erreurs. On peut aussi citer les travaux de Contractor et al. (2010) qui utilisent la notion de distance édition afin de repérer les tokens graphiquement voisins d'une altération et ainsi décoder un message. Gouws et al. (2011) reprirent d'ailleurs ce précédent travail pour y ajouter un dictionnaire de paire de mots fortement associés tel que « *u→you* ».

Han et Baldwin (2011) ont développé quant à eux un classifieur afin de détecter et de proposer des candidats de normalisation aux tokens non standard. Ils s'appuient notamment sur des métriques graphiques et phonétiques afin de générer un premier ensemble de candidats de normalisation. Ils classent ensuite ces candidats en prenant en compte de nombreux autres paramètres tels que le contexte, des informations morphologiques (affixes par exemple) ou encore la distance d'édition entre le mot initial et le candidat de normalisation proposé.

Liu et al. (2011) proposent, la même année, d'entraîner un modèle CRF afin de produire des variantes lexicales pour un token donné. Liu et al. (2012) reprennent par la suite ce système et y ajoutent un modèle d'erreur s'appuyant sur plusieurs critères tels que la distance d'édition, la similarité graphique et phonétique ou

15. TiLT est une plateforme modulaire et multilingue de traitement automatique des langues naturelles qui a été mise en place chez Orange Labs. Elle propose plusieurs applications telles que la correction orthographique, l'extraction d'informations, la traduction automatique ou encore le prétraitement linguistique de documents.

encore l'amorçage visuel[16]. Hassan et Menezes (2013) proposent de prendre en compte les contextes pour faire de la normalisation. Pour cela, ils extraient, à l'aide d'une marche aléatoire, les contextes similaires en les associant à des paramètres de distance d'édition.

Nous pouvons encore citer le travail de Chrupała (2014) ou encore celui de Yang et Eisenstein (2013). Chrupała (2014) propose un système qui ne nécessite aucune ressource lexicale et qui s'appuie uniquement sur l'apprentissage des opérations de distance d'édition provenant d'un grand nombre de données non annotées et de très peu de données annotées. Yang et Eisenstein (2013) proposent d'utiliser un modèle statistique non supervisé dans lequel les relations entre mots standards et non standards sont représentées par un modèle log-linéaire.

3.2.1.4 Conclusion

Quel que soit l'angle choisi pour réaliser une tâche de normalisation, on constate très vite qu'il comporte des limites (Beaufort et al., 2010). Ainsi, les systèmes de normalisation utilisant des systèmes de reconnaissance vocale ont souvent le défaut de perdre des informations lors de la phonétisation du texte. Certains tokens bien orthographiés peuvent ainsi être perdus. Par ailleurs, cette méthode aura automatiquement plus de difficultés à traiter les altérations qui modifient la phonétique d'un token. Les approches ayant choisi d'utiliser des techniques de traduction se voient très vite reprocher de dépasser trop largement les besoins d'une tâche de normalisation. Elles ne sont par ailleurs pas considérées comme capables de prendre en compte la créativité lexicale que l'on rencontre dans des messages très bruités (Kobus et al., 2008). Enfin certaines approches employant des techniques propres à la correction orthographique automatique ont pour *défaut* le fait d'analyser un texte mot à mot (voir toutefois Seddah et al., 2012), passant ainsi souvent à côté des problèmes d'agglutination de mots par exemple. Il est à noter de manière plus générale qu'un grand nombre de ressources s'appuient sur l'ajout de ressources lexicales afin de détecter et normaliser certains tokens propres au langage SMS. Ces tokens, parfois très transformés, peuvent être complexes à détecter autrement (ex : *slt*→*salut*, *pk*→*pourquoi*, *2m1*→*demain*). Toutefois, la langue étant par nature dynamique, ces créations lexicales évoluent rapidement et ne peuvent pas être listées de manière exhaustive. Ces ressources lexicales doivent ainsi très régulièrement être mises à jour sans quoi elles risquent de devenir assez rapidement obsolètes.

16. L'amorçage visuel est un concept propre au domaine de la psychologie cognitive qui désigne l'effet qu'a la présentation préalable d'un stimulus (l'amorce) pour influencer le traitement d'un autre stimulus (la cible). Il y a effet d'amorçage lorsque la cible est reconnue plus rapidement. Par exemple : si une personne lit une liste de mots dans laquelle figure le mot *table* et que plus tard on lui demande de donner un mot commençant par *ta*, il y a de fortes chances que ce dernier réponde *table*, ce mot étant déjà amorcé.

La majorité des systèmes de normalisation existants actuellement ont principalement pour but de traiter des messages correspondants à des SMS ou propres aux réseaux sociaux. Les textes que nous voulons analyser et normaliser ne sont pas tous de cette nature et sont souvent moins bruités. Les travaux réalisés en correction orthographique traitent au contraire des textes relativement peu bruités. Et, bien qu'ils ne partagent pas parfaitement les mêmes objectifs, ils ont inspiré de nombreux travaux de normalisation. C'est pourquoi nous nous sommes particulièrement intéressée par la suite aux études faites dans ce domaine.

3.2.2 Systèmes de correction

Comme nous l'avons évoqué ci-dessus, on attend d'un système de correction qu'il sache détecter automatiquement tous les tokens fautifs rédigés dans un texte et qu'il propose une ou plusieurs corrections pertinentes pour chacun d'entre eux. Contrairement à la normalisation, qui tente à l'aide de nombreuses stratégies de prendre en compte les altérations réalisées volontairement, la correction automatique ne fait que très rarement la distinction entre les altérations fautives et volontaires.

On divise généralement les erreurs en deux catégories distinctes : les fautes lexicales et les fautes grammaticales (Kukich, 1992). Sont placées dans la catégorie des fautes lexicales tous les tokens erronés qui ne figurent pas dans un dictionnaire (voir exemple *a*), contrairement aux fautes grammaticales qui ne peuvent être détectées que si le contexte est pris en compte (voir exemple *b*).

a) j'**aimrai resrever** 2 billets

b) **ces** nul ce forfait tu captes pas en montagne je le sais car je **lait**

Ces deux types d'erreurs sont souvent traités séparément dans la littérature. Le choix des techniques de correction utilisées est souvent guidé par l'objectif visé par le correcteur. Ainsi la détection d'une erreur lexicale sera souvent réalisée uniquement en vérifiant leur présence dans un dictionnaire contrairement aux fautes grammaticales pour lesquelles cette stratégie ne fonctionnerait pas. Certains correcteurs cherchent à corriger des fautes grammaticales en plus des fautes lexicales. Dans ce cas, la détection d'erreurs peut s'avérer plus complexe à réaliser et la génération de candidats de correction plus risquée. On peut par exemple citer les travaux de Carlson et Fette (2007) qui s'appuient principalement sur des approches n-gramme pour réaliser leur étude ou encore ceux de Yvon (2011) qui fondent leur système sur des modèles probabilistes (un modèle de langue et un modèle d'erreur) et qui l'implémentent à l'aide de transducteurs. Ces travaux demeurent toutefois très peu nombreux, c'est pourquoi nous nous concentrerons dans la suite de cette section sur les différentes stratégies mises en place afin de

traiter les fautes lexicales et grammaticales séparément [17], puis nous donnerons un rapide aperçu des systèmes existants pour l'aide à la correction.

3.2.2.1 Correction lexicale

Les travaux sur la correction lexicale ont débuté dans les années 1960 (Blair, 1960; Damerau, 1964). Au fil du temps et des avancées technologiques, ces travaux ont beaucoup évolué (Kukich, 1992; Mitton, 2009, 2010; Rozovskaya, 2013). L'état de l'art proposé ci-dessous n'a donc pas pour objectif d'être exhaustif, mais plutôt de donner un aperçu des techniques utilisées.

Les premiers systèmes proposés en correction automatique se limitaient aux tokens à corriger sans prendre en compte leur contexte. Ces systèmes reposent souvent sur la notion de *distance d'édition* introduite par Damerau (1964) puis Levenshtein (1966). Cette notion met en œuvre quatre types d'opérations (l'insertion et la suppression d'une lettre, la substitution d'une lettre par une autre, l'inversion de deux lettres consécutives). L'idée est de s'appuyer sur ces opérations, que l'on peut donc considérer comme des règles de correction, pour générer les candidats de correction d'un token erroné. Initialement, chaque candidat était pondéré en fonction du nombre d'opérations nécessaires à sa génération.

Cette méthode a été par la suite reprise et améliorée par de nombreux travaux. Kernighan et al. (1990) et Church et Gale (1991) tentent ainsi de retrouver les candidats de correction d'un token erroné en cherchant quelle opération a permis sa création en s'appuyant sur des modèles de langue et d'erreur. Ils montrent par ailleurs que le contexte dans lequel se produit une erreur peut être pris en compte. Oflazer (1996) applique ce même type de correction en utilisant un système de vérification dans le lexique plus tolérant [18]. Beaufort (2010) propose, quant à lui, de passer par des transducteurs. Les erreurs phonétiques étant difficilement détectables et corrigibles via des distances d'édition simples, plusieurs travaux ont choisi de prendre en compte la phonétique des mots (Véronis, 1988; Beaufort, 2010; Sagot et Boullier, 2008). Les fautes de proximité clavier, de casse ou encore de diacritiques étant assez spécifiques, plusieurs systèmes ont choisi d'affiner leurs paramétrages (Sagot et Boullier, 2008; Beaufort, 2010). On peut enfin citer aussi Wisniewski et al. (2010) qui montrent la possibilité d'extraire des règles de correction en s'appuyant sur un corpus d'erreur annoté.

17. Bien que les fautes OCR (voir par exemple Sagot et Gábor, 2014) et spécifiques à l'apprentissage des langues (voir par exemple Rozovskaya, 2013) ont fait l'objet de nombreuses études, nous ne nous attarderons pas dessus dans ce document.

18. Il part du principe que chaque mot inconnu résulte d'un token bien orthographié par l'application d'une ou plusieurs opérations sur les lettres de ce mot. Il existe quatre opérations : l'insertion, la suppression, le remplacement d'une lettre ou l'inversion de deux lettres. Lorsqu'on a un mot mal orthographié, on va le rechercher de manière tolérante dans un dictionnaire stocké sous la forme d'un automate fini. Cette méthode permet ainsi de faire de la reconnaissance approximative de mots.

Il est à noter que la pondération de ces règles ne se fait plus systématiquement en fonction de la distance d'édition, c'est à dire du nombre exact d'opérations effectuées sur le mot. Le coût d'une correction peut aussi bien correspondre à une distance d'édition qu'à une assignation de poids réalisée au cas par cas, manuellement ou non, en fonction de l'opération et des lettres concernées par la règle en question (Véronis, 1988; Mitton, 1996; Sagot et Boullier, 2008; Beaufort, 2010; Wisniewski et al., 2010; Bouvier et Bellot, 2013).

En parallèle à la distance d'édition, une seconde approche a été proposée. Cette dernière suggère de représenter les mots à l'aide de clefs afin de retrouver plus aisément la correction d'une altération. Plus concrètement, la proposition faite est d'assigner des clefs aux chaînes de caractères en faisant en sorte que si un mot est altéré sa clef demeurera proche de sa forme initiale. Il est à noter que de manière générale les fautes phonétiques sont assez mal gérées par cette stratégie. Plusieurs stratégies ont toutefois été mises en place afin de créer ces clefs (Pollock et Zamora, 1984; Ndiaye et Vandeventer Faltin, 2004; Reynaert, 2004).

Pollock et Zamora (1984) et Ndiaye et Vandeventer Faltin (2004) proposent d'assigner à chaque entrée de leur lexique de référence une clef constituée des lettres de cette dernière réorganisées dans un ordre particulier. Ces clefs conservent généralement la liste des consonnes puis des voyelles distinctes composant l'entrée en question. Ces dernières sont souvent ordonnées soit dans leur ordre d'apparition soit dans un ordre plus motivé [19]. Reynaert (2004), quant à lui, suggère de s'appuyer plutôt sur des clefs numériques qui pointeraient sur toutes les anagrammes d'unigrammes ou de bigrammes de token. Ces clefs sont ainsi calculées à l'aide de la formule suivante :

$$Key(w) = \sum_{i=1}^{|w|} f(c_i)^n$$

dans laquelle w est un mot composé des caractères c_1 à $c_{|w|}$ et f correspond une valeur numérique assignée à chaque caractère c de w.

La distance d'édition et la création de clefs sont deux approches qui permettent notamment la production de candidats de correction. Et bien que de nombreux travaux s'en soient contentés, beaucoup ont proposé d'utiliser des systèmes de pondération plus élaborés (comme nous l'avons par exemple vu ci-dessus) et ont mis en place d'autres approches de correction plus flexibles, prenant souvent en compte beaucoup plus de paramètres.

De nombreuses études ont choisi d'utiliser des modèles de langage et d'erreurs s'appuyant sur des n-grammes afin de déterminer quel est la correction la plus probable pour une faute donnée. Ces n-grammes sont généralement utilisés au

19. Pollock et Zamora (1984) par exemple proposent entre autres de réordonner les consonnes en fonction de la probabilité qu'elles ont d'être omises.

niveau du token (Brill et Moore, 2000; Carlson et Fette, 2007) (voir toutefois Farra et al., 2014) ce qui permet de prendre en compte le contexte du mot erroné afin de proposer le candidat de correction le plus approprié. Toutefois, se limiter aux tokens n'est que partiellement satisfaisant. En effet, si le contexte du token erroné est lui aussi mal orthographié, l'analyse proposée sera limitée. C'est pourquoi Toutanova et Moore (2002) proposent d'utiliser des n-grammes phonétiques afin de pallier les erreurs phonétiques qui pourraient être présentes dans le contexte. (Boyd, 2009) quant à elle, suggère de prendre en compte ces deux types de n-grammes, et s'appuie ainsi à fois sur la graphie des tokens et sur leur phonétique. Ces modèles peuvent être associés ou non à un modèle d'erreur et s'appuient généralement sur de nombreux paramètres supplémentaires tels que la position d'une erreur dans un mot, la catégorie du mot à corriger, sa longueur ou encore sa phonétique.

D'autres études, différentes, s'appuient sur des mesures distributionnelles. Ainsi Bouvier et Bellot (2013) proposent un système non supervisé qui mêlerait distance d'édition et propriétés distributionnelles. C'est aussi le cas de Suignard et Kerroua (2013), qui se sont appuyés sur les travaux de Li et al. (2006). Ces derniers utilisent deux approches qui prendraient en compte des paramètres de similarité distributionnelle en plus d'autres plus courants tels que la distance d'édition entre deux chaînes, leur proximité phonétique ou encore leurs fréquences.

On peut aussi constater que certains systèmes ont plutôt choisi de s'appuyer sur des automates à états finis (Ringlstetter et al., 2006).

Enfin, notons que l'émergence d'internet a eu des répercussions sur plusieurs techniques de correction qui l'ont considérée comme une source pertinente d'informations. C'est, par exemple, le cas de Cucerzan et Brill (2004) ou encore de Qing et al. (2007) qui ont choisi de l'exploiter.

3.2.2.2 Correction grammaticale

Comme nous l'avons expliqué précédemment, les fautes grammaticales produisent des tokens présents dans le dictionnaire. Il peut ainsi aussi bien s'agir de fautes d'accord (par exemple : *danse/danses/dansent*) que de fautes qui produiraient un autre token rattaché à un autre lemme que celui attendu tel que les erreurs causées par homophonie ou paronymie (par exemple : *conseil/conseille* ou *pain/bain*). Dans cet état de l'art, nous ne nous attarderons pas sur les fautes d'accord, notre travail n'ayant pas pour objectif de traiter ce genre de fautes. Pour détecter et corriger ce second type d'erreurs, s'appuyer sur le contexte est inévitable. En effet, si l'on a la phrase « *Enfin un objet qu'***ont** *peut emporter partout avec* **sois** », on ne pourra corriger *ont* en *on* et *sois* en *soi* qu'à condition de prendre en compte leur environnement.

Ces dernières années, la correction grammaticale a fait l'objet de nombreuses études, et ce, notamment avec les campagnes d'évaluation (*shared tasks*) qui ont été organisées depuis 2011 : HOO — *Helping Our Own* — en 2011 et 2012 (Dale et Kilgarriff, 2011; Dale et al., 2012) et à la conférence CoNLL en 2013 et 2014 (Ng et al., 2013, 2014). Il est à noter que la majorité de ces études se concentrent sur les erreurs faites par les apprenants. Nos travaux ne se concentrant pas sur ce type d'altération, nous ne nous y attarderons pas.

Une grande partie des techniques employées pour la correction grammaticale reposent sur la même logique. Elles tentent dans un premier temps de détecter une erreur puis créent, dans un second temps, une liste de candidats de correction possibles pour enfin choisir la correction la plus probable.

Contrairement à la majorité des travaux réalisés en correction lexicale, on ne peut plus ici se contenter d'une simple vérification dans le dictionnaire pour stipuler qu'un token est erroné. Tous les tokens contenus dans une phrase sont susceptibles d'être mal orthographiés. C'est pourquoi il est nécessaire de déterminer tout d'abord quels sont les tokens que l'on veut corriger. Pour ce faire, on peut par exemple passer par la création d'ensembles de confusion [20] prédéfinis (Golding et Schabes, 1996; Golding et Roth, 1996; Carlson et Fette, 2007; Stehouwer et van Zaanen, 2009; Pedler et Mitton, 2010; Xu et al., 2011). Certaines études se concentrent ainsi sur les erreurs les plus fréquentes (Golding et Schabes, 1996), d'autres encore se limitent à certaines catégories grammaticales. Rozovskaya et Roth (2010) et Cahill et al. (2013), qui sont focalisés sur les prépositions, et Lee et Seneff (2008) et Rozovskaya et al. (2014), qui ont privilégié les verbes, illustrent ce dernier cas. Théoriquement, il est possible de corriger toutes les erreurs contenues dans un texte. Des ensembles de confusion peuvent, pour cela, être construits à petite ou à grande échelle, manuellement ou non. Mangu et Brill (1997) proposent, par exemple, de se reposer sur un système de règles de transformation tandis que Pedler et Mitton (2010) suggèrent de produire ces ensembles de confusion en appliquant un correcteur, ce dernier s'appuyant notamment sur la distance d'édition. Lorsque la création de ces ensembles est réalisée automatiquement, leur nombre n'est alors plus restreint. Cela permet à certaines études de vérifier l'orthographe de chaque token qui compose une phrase et de déduire pour chacun de ces tokens son ensemble de confusion. C'est par exemple le cas de Islam et Inkpen (2009) ou de Stehouwer et van Zaanen (2009) qui s'appuient sur des modèles n-gramme.

Suite à la sélection des fautes que l'on veut traiter, plusieurs stratégies de correction sont envisageables. On peut mettre en place un système de classification tel que le font Rozovskaya et Roth (2010) ou encore de Seo et al. (2012), qui utilisent un ensemble de classifieurs entraînés sur différents corpus. D'autres approches

20. Nous définissons dans ce travail un ENSEMBLE DE CONFUSION comme un ensemble de tokens souvent utilisés les uns à la place des autres par erreur (ex : {*on, ont*} ou {*est, ai, et, es*}). Cet ensemble représente généralement l'ensemble des candidats de correction possibles pour une faute donnée.

choisissent de mettre en place un système de traduction (Junczys-Dowmunt et Grundkiewicz, 2014) [21], d'employer des approches par règles (Mangu et Brill, 1997; Rozovskaya et Roth, 2010), de se servir des modèles de langue n-gramme ainsi que le propose par exemple Golding et Schabes (1996), qui s'aident des catégories grammaticales associées à des paramètres contextuels, ou encore comme Park et Levy (2011) qui accompagnent ce modèle probabiliste d'un modèle de canal bruité. Cette dernière solution souvent combinée à de gros corpus, à des modèles d'erreurs, à différents types de paramètres (fréquence, catégories morphosyntaxiques, informations contextuelles...) ou encore à des mesures de similarité (phonétique, graphique...), est très utilisée (Carlson et Fette, 2007; Islam et Inkpen, 2009; Xu et al., 2011; Nazar et Renau, 2012).

Certaines études proposent aussi la combinaison de différentes approches. C'est le cas de Stehouwer et van Zaanen (2009) qui utilisent un classifieur s'appuyant sur des modèles n-gramme ou de Whitelaw et al. (2009) qui intègrent dans leur système un modèle de langue, un modèle d'erreur et un classifieur. On peut aussi citer les travaux de Gao et al. (2010) qui combinent un classifieur (s'appuyant sur des modèles n-gramme, des informations lexicales et grammaticales) et un système de cooccurrences, de Zesch (2012), qui mêle approche statistique (de type n-gramme) et à base de connaissances, ou encore, de Susanto et al. (2014) qui proposent d'employer un système de classification et de traduction automatique pour procéder à la correction grammaticale d'un texte.

3.2.2.3 Outils industriels d'aide à la rédaction

L'orthographe étant un problème très actuel, de nombreux outils ont aussi été mis en place dans le commerce pour aider les utilisateurs lors de la rédaction de leurs documents. On peut distinguer parmi ces derniers ceux qui sont directement intégrés aux outils de rédaction (logiciels de traitement de texte, messagerie, etc.) et ceux qui sont autonomes (Jacquet-Pfau, 2001).

Les améliorations orthographiques d'un outil d'aide à la rédaction vont en grande partie dépendre de son lexique et du système implémenté pour la détection de tokens fautifs et pour la production de leurs candidats de correction.

Il est à noter qu'un système d'aide à la rédaction se distingue grandement des correcteurs automatiques. En effet, bien qu'un tel système annote les tokens détectés comme fautifs et suggère des candidats de correction en conséquence, c'est généralement à l'utilisateur de déterminer si le mot détecté comme erroné l'est vraiment et, s'il s'avère que c'est le cas, c'est encore lui qui sélectionne la correction valide. Ainsi, l'humain permet d'effectuer une validation des résultats proposés par les systèmes d'aide à la rédaction. En correction automatique, ce choix est légèrement plus complexe puisque le système est seul juge. Il doit donc être capable,

21. Cette technique est notamment utilisée pour corriger des fautes d'apprenants.

par lui-même, de déterminer quelle est la correction la plus probable pour une faute donnée. Il ne sera ainsi suivi d'aucune validation humaine.

Décrire le fonctionnement de ces outils d'aide à la rédaction demeure toutefois malaisé, ces derniers ne dévoilant que très peu l'architecture de leurs systèmes et les procédés de correction qu'ils ont choisi d'employer. Figurent parmi les plus complets sur le marché francophone les outils d'aide à la rédaction Antidote et Cordial [22].

Antidote (Brunelle, 2004), commercialisé par *Druide Informatique* depuis 1996, propose notamment d'aider les utilisateurs à plusieurs niveaux en leur fournissant plusieurs guides linguistiques, des dictionnaires (lexical, de synonymes, d'antonymes, de rimes, analogique...) [23] et un système de correction. Sont couverts par ce correcteur : les fautes orthographiques (lexicales), syntaxiques, sémantiques (confusion d'homophones, impropriétés...), typographiques et stylistiques (registre de langue, répétition , etc.). Le fonctionnement même du système de correction utilisé par Antidote est difficilement descriptible, ce dernier n'ayant que peu communiqué sur ce sujet. On peut toutefois préciser qu'il s'appuie, entre autres, sur de nombreuses ressources lexicales, sur un analyseur syntaxique employant des méthodes symboliques et statistiques (ex. : cooccurrences), sur un système de détection anaphorique, ou encore sur des classifieurs afin de réduire des cas de surdétection (Charest et al., 2007; Gotti et al., 2012; Pironneau et al., 2014).

Existant depuis 1995 environ, Cordial (CORrecteur d'Imprécisions et Analyseur Lexico-sémantique) est un logiciel d'aide à la correction lexicale et grammaticale développé par *Synapse Développement* (Laurent et al., 2009a,b; Laurent, 2012). Il s'appuie sur un analyseur syntaxique robuste. Ce dernier est notamment composé de plusieurs modules tels que la consultation des lexiques, la correction lexicale, la détection des entités nommées, la désambiguïsation grammaticale, le regroupement en constituants ou encore la désambiguïsation sémantique. La correction lexicale s'appuie ici notamment sur des notions de distance d'édition et de proximité clavier et phonétique (à l'aide d'un phonétiseur intégré). En plus de la pondération des scores de chacune de ces opérations de transformation s'ajoute la fréquence de chacun des candidats de correction et le contexte dans lequel ils s'intégreraient. Le contexte est ici, entre autres, pris en compte à l'aide d'un dictionnaire de cooccurrences. La correction grammaticale, quant à elle, est notamment réalisée à l'aide d'une analyse syntaxique et sémantique. Sont notamment utilisés pour cette opération un système de règles, un dictionnaire de cooccurrences et des tables statistiques (bigramme, trigramme) apprises sur de très gros corpus regroupant différents styles d'écriture et de sujets (journalistique, encyclopédique, littéraire, technique ou encore provenant de blogs). Ces tables prennent plusieurs

22. L'ordre choisi pour évoquer ces outils est alphabétique et non qualitatif. N'ayant pas pu tous les tester, nous ne proposerons ici aucun comparatif.

23. Ces dictionnaires sont mis à jour en deux grandes étapes : une première étape de détection de créations lexicales qui se fait à l'aide d'un système de veille informatique et une seconde d'intégration dans les dictionnaires qui se fait manuellement.

critères en compte comme la fréquence des tokens, leur degré d'ambiguïté ou encore leur catégorie grammaticale.

Bien que ces descriptions ne soient que très peu détaillées et ne soient pas représentatives de tous les systèmes existants sur le marché, elles offrent un aperçu de la grande couverture de ces outils et de leurs applications. Elles nous permettent notamment de faire plusieurs constats. Tout d'abord, malgré le fait qu'ils permettent la détection et la correction d'erreurs de nature très différente, ces outils sont toutefois trop complexes pour nos besoins actuels. Et pour cause, ils n'ont pas été développés dans le but de procéder à une correction automatique et systématique de gros volumes de textes mais dans l'objectif d'aider un utilisateur à rédiger et à améliorer ses écrits. Par ailleurs, ces analyses sont souvent adaptées à une écriture plutôt soignée. Si un utilisateur prend le temps de faire une vérification orthographique et grammaticale du texte qu'il rédige, on peut supposer qu'il a essayé de faire attention à son orthographe. Par conséquent, ces outils, peu confrontés à des textes très bruités et non implémentés pour traiter des erreurs de type « texto », sont souvent mis à mal face à des corpus du web. Enfin, choisir un de ces systèmes plutôt qu'en implémenter un sous-entendrait que l'on puisse avoir la capacité de le modifier afin de l'adapter à notre tâche et de l'enrichir en fonction de nos besoins. Or cela est difficilement réalisable si ces outils sont industriels. En outre, nous avons besoin dans le cadre de ce travail d'un outil qui puisse être intégré à une chaîne de traitement industrielle entièrement automatisée, ce pour quoi ces outils ne sont pas conçus. Ainsi en plus du fait que sélectionner une telle solution peut s'avérer coûteux, utiliser ce type de logiciels dans notre cas était difficilement envisageable.

3.3 Conclusion

La tâche que nous voulons effectuer est une tâche de normalisation, au sens où nous l'avons définie précédemment. En effet, nos résultats n'ont pas besoin d'être aussi précis que ceux proposés par un correcteur automatique. Nous acceptons donc toute modification d'une forme si elle ne diffère de celle attendue qu'au niveau de sa morphologie flexionnelle. Par ailleurs, nous avons pour objectif de mettre en place un système qui serait apte à traiter toutes les altérations possibles et pas uniquement celles qui ont fait l'objet d'une faute d'orthographe.

Ainsi, mettre en place un système de correction *lâche* ne serait pas suffisant. En outre, comme nous l'avons souligné précédemment, les systèmes de normalisation reposant sur de la traduction automatique ou sur de la reconnaissance vocale demeurent trop spécifiques et trop restrictifs pour notre cas. C'est par conséquent en connaissance de cause que nous avons finalement choisi de mettre en place un système de normalisation qui s'appuierait en grande partie sur des techniques de correction lexicale et grammaticale.

Deuxième partie

Mise en place d'un système de normalisation

Introduction de la deuxième partie

Le système décrit dans cette partie a pour objectif de normaliser, à la morphologie flexionnelle près, des textes bruités. Pour ce faire, il doit être capable de détecter chaque altération présente dans un texte puis de la normaliser, qu'elle soit connue ou non de notre lexique. On parlera ainsi de TOKEN INCONNU ALTÉRÉ dans le cas d'une altération qui donnerait lieu à un token inconnu de notre lexique et dans le cas inverse de TOKEN CONNU ALTÉRÉ [24].

En outre, le système mis en place doit fonctionner de manière indépendante de la langue, bien que dans cette partie nous nous restreignons au français. Notre système de normalisation devra pouvoir fonctionner pour d'autres langues reposant sur une morphologie dérivationnelle concaténative. Notre objectif est que ce système puisse ainsi être aisément adaptable à d'autres langues telles que l'anglais, l'allemand ou encore l'espagnol.

Avant de réaliser notre système, nous avons effectué une étude de corpus sur les tokens inconnus altérés. Elle nous a notamment permis d'obtenir une vision générale des inconnus que nous souhaitons normaliser. Cette étude portait sur 1 976 altérations extraites des corpus viavoo. Ces dernières peuvent ainsi être classifiées en trois grandes catégories et quatorze sous-catégories comme illustré dans la table 3.2. En outre, connaître la fréquence d'apparition de chacune de ces altérations dans nos corpus peut notamment aider à la mise en place de notre système puisqu'il est nécessaire de traiter en priorité les altérations les plus gênantes et les plus courantes. Ainsi les altérations introduisant des caractères non alphabétiques, par exemple, seront à traiter en premier puisqu'elles peuvent gêner la segmentation du texte donné.

24. En fonction du chapitre, le terme *altération* pourra être employé pour désigner uniquement l'un ou l'autre de ces deux types d'altérations, le contexte déterminant de manière non ambiguë la bonne interprétation.

Type d'altération	Altération	exemple	nb occ.	Fréquence
altération	Ajout d'une lettre	*impossi**l**ble - impossible*	237	12,0%
sur	Suppression d'une lettre	*veullez - veuillez*	448	22,7%
caractères	Diacritique	***e**clairage - éclairage*	281	14,2%
alphabétiques	Substitution d'une lettre	*arti**g**le - article*	230	11,6%
	Inversion d'une lettre avec la suivante	*ex**ca**tement - exactement*	14	00,7%
	Total		**1210**	**61,2%**
altération sur	suppression d'un espace	*bonton - bon ton*	121	06,1%
caractères non	Apostrophe	*l 'achat - l' achat*	153	07,8%
alphabétiques	**Total**		**274**	**13,9%**
	Étirement	*pourqu**ooiiii** - pourquoi*	13	00,7%
	Décomposition	*in-cro-ya-ble - incroyable*	6	00,3%
altération	Autocensure	*m***e - mince*	1	00,0%
	squelette consonantique	*bjr - bonjour*	23	01,2%
volontaire	troncation	*math - mathématiques*	97	04,9%
	réduction phonétique	*jam**é** - jamais*	331	16,7%
	réduction symbolique	***2**m**1** - demain*	21	01,1%
	Total		**492**	**24,9%**
Total			1976	100%

Table 3.2 – Étude de corpus des tokens inconnus altérés.

Le système proposé dans cette thèse est constitué de plusieurs modules. Ces derniers ont pour objectif de débruiter le texte traité au fur et à mesure afin d'améliorer la qualité de notre analyse et d'optimiser ainsi nos résultats. Procéder de la sorte présente notamment deux avantages concrets :

- Pour une tâche de détection et de normalisation des tokens inconnus altérés, cela peut nous permettre d'écarter en amont les inconnus, altérés ou non, qui auraient pu induire en erreur notre système et ainsi les traiter en conséquence. Dans le cas où une altération inconnue peut être étiquetée ou normalisée de plusieurs manières à la fois, cette ambiguïté sera conservée pour être résolue dans une étape ultérieure. Nous reviendrons sur cette notion d'ambiguïté dans les chapitres à venir.
- Pour une tâche de détection et de normalisation des altérations connues à présent, avoir un système progressif est d'autant plus important qu'il est nécessaire de prendre en compte le contexte. Faire précéder ce système d'étapes préliminaires améliora ainsi la qualité du texte traité et ainsi le contexte de l'altération à normaliser.

L'architecture générale de notre système de normalisation, illustré figure 3.1, se compose ainsi de quatre grandes étapes.

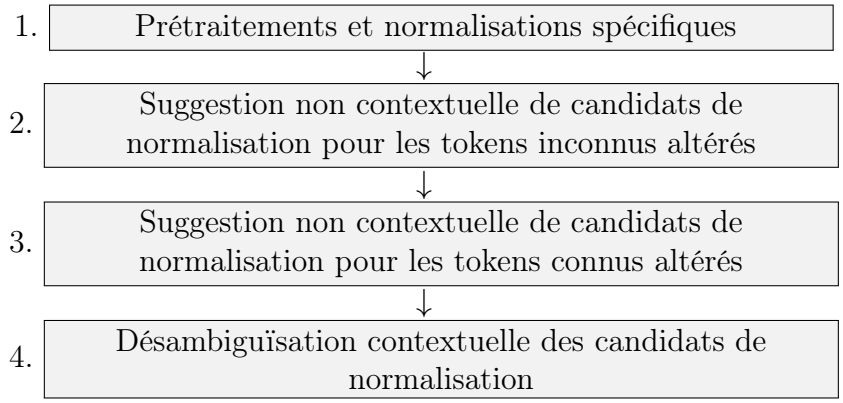

FIGURE 3.1 – Modules composant notre système de normalisation

La première de ces étapes rassemble plusieurs tâches de prétraitement et vise à simplifier l'étape ultérieure de normalisation globale. Elle réunit trois objectifs :

1. réaliser un découpage afin de distinguer chacun des tokens composant les différentes phrases du texte,
2. proposer des candidats de normalisation à certains tokens inconnus altérés spécifiques pour lesquels l'opération d'altération est connue, stable et régulière,
3. reconnaître et étiqueter les tokens inconnus ne correspondant pas à des altérations stables.

La mise en place de ces différents points est ainsi développée dans le chapitre 4.

La seconde étape de notre système a pour but de normaliser chaque inconnu correspondant à une altération. Elle cherche notamment à produire un ou des candidats de normalisation sans prise en compte du contexte en s'appuyant sur un système d'apprentissage par analogie de règles de normalisation. Cette étape sera détaillée dans le chapitre 5.

Enfin, les deux dernières étapes requièrent la prise en compte d'informations contextuelles. Elles seront toutes deux rassemblées dans le chapitre 6.

Ainsi, après avoir proposé des candidats de normalisation pour les altérations donnant lieu à des tokens inconnus, nous nous sommes appliquée à mettre en place une troisième étape dans notre système qui propose un ou des candidats de normalisation pour certains tokens connus de nos lexiques de référence.

Puis ce système s'achèvera avec la quatrième étape : l'étape de désambiguïsation. Cette dernière a pour objectif d'ordonnancer les différents candidats de normalisation obtenus précédemment et afin de ne conserver que le ou les plus pertinents.

CHAPITRE 4

Pré-traitements

Sommaire

Ce chapitre décrit la première étape de notre système de normalisation. Son objectif est d'exécuter, sur le texte que nous souhaitons analyser, un ensemble de prétraitements dont l'application aurait de nombreux intérêts pour un système comme le nôtre.

L'objectif de ces prétraitements est tout d'abord de distinguer, parmi les inconnus, quels sont ceux qui sont réellement à normaliser, ces derniers ne correspondant pas systématiquement à des altérations. En effet, tenter de tous les remplacer par d'autres tokens, ce qui semblerait au premier abord plus adéquat, pourrait au contraire dégrader grandement la qualité d'un texte. Par exemple, dans la phrase « *la pixelisation de ton screen est ultrafacile à coriger* », les tokens *pixelisation*, *screen*, *ultrafacile* et *coriger* sont inconnus de notre lexique de référence. Toutefois, seul le token *coriger* nécessite une opération. Nous ne voulons pas toucher aux néologismes *pixelisation* et *ultrafacile*, ni à l'emprunt *screen*. Être capable de distinguer les différentes catégories d'inconnus est donc nécessaire. Cette tâche de classification des inconnus n'est pas spécifique à la correction ou à la normalisation de texte. Ce besoin de distinguer une catégorie spécifique d'inconnus parmi les autres est partagé avec d'autres tâches en traitement automatique de la langue. C'est par exemple le cas du projet ANR EDyLex [1]. Ce projet avait pour objectif de faire de l'enrichissement dynamique de lexique. Pour ce faire, nous avons besoin de dessiner une frontière entre les différents inconnus existants afin de pouvoir déterminer quels sont ceux pouvant être ajoutés à un dictionnaire. Ce projet, auquel nous avons participé, partageait ainsi avec notre travail ce besoin de distinguer les néologismes des altérations même si nos objectifs finaux diffèrent.

1. `https://sites.google.com/site/projetedylex`

Ensuite, nous voulons être capable d'analyser de nombreuses altérations différentes les unes des autres. Nous avons donc mis en place des systèmes de normalisation (décrits dans les chapitres 5 et 6) relativement génériques mais qui sont conçus uniquement pour traiter des altérations plus classiques, accessibles à une normalisation reposant sur un système d'apprentissage de règles par analogie. Par conséquent, si une altération est créée volontairement à l'aide d'un procédé spécifique, sa normalisation pourra être moins évidente à réaliser pour ces systèmes. Toutefois, ces altérations peuvent résulter de procédés aisés à détecter et à normaliser par d'autres moyens. Par exemple, les tokens *nooooooooooon* ou *dégon-flé*, qui correspondent respectivement à *non* et *dégonflé*, sont détectables par la régularité et la singularité des procédés qui ont permis leur création. Ajouter un ou plusieurs systèmes prenant en compte ces différents types de mécanismes est une stratégie plus fiable que tenter de les analyser à l'aide de systèmes de normalisation génériques.

Le système présenté ici s'intègre dans une chaîne de traitement modulaire : SxPipe (Sagot et Boullier, 2008). Comme nous l'expliquerons ci-dessous, cette chaîne de traitement est générique. Nous l'avons par conséquent reprise et adaptée à nos besoins afin de la rendre apte à réaliser des tâches de normalisation, notamment sur nos corpus. Nous commencerons ainsi ce chapitre en expliquant ce choix et le fonctionnement global de cet outil (section 4.1), puis nous détaillerons à la section 4.2 les différents systèmes de normalisation mis en place pour les altérations spécifiques, stables et régulières qui sont fréquemment attestées, enfin nous décrirons dans les sections 4.3, 4.4 et 4.5 les diverses stratégies que nous avons mises en place afin d'écarter les inconnus ne correspondant pas à des altérations.

4.1 La chaîne de traitement SxPipe

Avant de présenter la chaîne de traitement SxPipe (Sagot et Boullier, 2008), il nous semble important d'expliquer les raisons qui nous ont poussée à la choisir. Tout d'abord, SxPipe est un outil sous licence Cecill-C [2] (compatible LGPL). Il est donc librement accessible, utilisable, modifiable et redistribuable y compris au sein d'une entreprise comme viavoo. Bien que cette chaîne de traitement soit générique, nous pouvons l'adapter à nos besoins. Cela est d'autant plus réalisable que SxPipe est modulaire. Il nous est ainsi possible de choisir les modules que nous voulons appliquer et leur paramétrage. Par ailleurs, si nous voulons personnaliser certains des modules appartenant à SxPipe, nous pouvons le faire, son code étant accessible et modifiable. Enfin, nous voulons mettre en place un système capable de traiter des textes en français, en anglais, en allemand et en espagnol, il est donc important de souligner le caractère multilingue de SxPipe, lequel est ainsi

2. Les licences CeCILL, abréviation de *CEA CNRS INRIA logiciel libre*, sont une famille de licences de logiciel libre.

en mesure de traiter toutes ces langues. Pour ce faire, SxPipe s'appuie sur des lexiques Alexina (Sagot, 2010) tel que le Le*fff* présenté section 1.2.2.

4.1.1 Format utilisé par SxPipe

Pour réaliser leur analyse, Sagot et Boullier (2008) s'appuient sur plusieurs notions telles que celles de forme simple et composée, de token ou encore d'amalgame, que nous avons déjà évoqué en section 1.1 (page 12). Ainsi, lors du découpage d'une phrase, tous les tokens qui y sont contenus obtiennent un identifiant unique indiquant leur position dans la chaîne de caractères initiale. Ils peuvent être ensuite rattachés à une forme simple ou composée ou à plusieurs formes amalgamées. Les auteurs ont par ailleurs introduit la notion de FORME SPÉCIALE afin d'abstraire un motif donné (composé d'une ou plusieurs formes) que l'on ne souhaite pas analyser, syntaxiquement parlant, autrement qu'en bloc. Nous illustrons ces différents cas dans la suite de cette section et dans la table 4.1 [3].

Lorsqu'une phrase est donnée à SxPipe, les tokens la composant sont alors stockés en COMMENTAIRE, entre accolades, avec leur identifiant. Cet identifiant, représenté par un élément XML, se présente de la sorte : {<F id="EiFj">token</F>}. Les indices i et j correspondent ici respectivement au numéro de la phrase et au numéro du token dans cette dernière. Pour des questions de lisibilité, un tel token pourra être représenté comme ceci {token$_j$}. Le token en commentaire est ensuite suivi de la forme qui lui est rattachée. Par exemple pour l'énoncé « *Le soleil se couche* », SxPipe proposera la sortie suivante :
{<F id="E1F1">Le</F>} le {<F id="E1F2">soleil</F>} soleil {<F id="E1F3">se </F>} se {<F id="E1F4">couche</F>} couche

Comme illustré dans le tableau 4.1, un token peut être dupliqué en sortie de SxPipe s'il correspond à deux formes amalgamées. Il peut aussi être regroupé à d'autres tokens au sein d'un même commentaire s'il appartient à une forme composée ou spéciale. Dans le cas d'une forme spéciale, c'est non plus la forme en question qui sera réécrite à la droite du token, mais son étiquette. L'étiquette d'une forme sera ainsi toujours inscrite à la suite d'un tiret bas en caractères majuscules.

Si un énoncé peut être analysé de plusieurs manières différentes, SxPipe conservera cette ambiguïté. Sagot et Boullier (2008) utilisent la notion de graphe orienté acyclique (DAG). Ils proposent ainsi de représenter une ambiguïté sous deux formes différentes : sous la forme d'une expression régulière ou sous la forme d'une liste de transitions. Nous pouvons illustrer ces deux représentations avec l'énoncé « *pomme de terre cuite* » proposé dans (Sagot et Boullier, 2008) :

3. Dans leur travail, Sagot et Boullier (2008) utilisent aussi la notion d'éléments détachables tels que les préfixes, les suffixes ou encore les clitiques qu'ils emploient lors de leurs segmentations en tokens et en formes. Notre travail ne s'appuyant pas sur cette analyse, nous ne nous attarderons pas dessus ici.

Type de Forme :	token(s)	Correction
Forme simple	*soleil*	{soleil$_1$} soleil
Forme amalgamée	*au*	{au$_1$} à {au$_1$} le
Forme composée	*a priori*	{a$_1$ priori$_2$} a_priori
Forme spéciale	*8 avenue Foch*	{8$_1$ avenue$_2$ Foch$_3$} _ADRESSE

Table 4.1 – représentation des différents types de formes dans SxPipe

1. ({pomme$_1$} pomme {de$_2$} de {terre$_3$ cuite$_4$} terre_cuite | ({pomme$_1$ de$_2$ terre$_3$} pomme_de_terre | {pomme$_1$} pomme {de$_2$} de {terre$_3$} terre) {cuite$_4$} cuite)

2. ##DAG BEGIN
   1 {pomme} pomme_de_terre 4
   1 {pomme} pomme 2
   2 {de} de 3
   3 {terre} terre 4
   3 {terre cuite} terre_cuite 5
   4 {cuite} cuite 5
   ##DAG END

4.1.2 Fonctionnement global de SxPipe

SxPipe est donc une chaîne de prétraitement modulaire qui applique à des corpus bruts une série de traitements de surface. Elle est composée de divers modules qui s'appliquent successivement.

Pour passer d'un texte brut à un texte analysé, Sagot et Boullier (2008) expliquent que ces modules se répartissent en 5 étapes :

1. les traitements effectués au niveau du texte brut,
2. le découpage du texte brut en phrases et en tokens,
3. les traitements effectués au niveau des tokens,
4. le regroupement des tokens en formes,
5. les traitements réalisés au niveau du DAG de formes.

Tous les modules composants les étapes 1, 3 et 5 peuvent être ajoutés ou non dans une chaîne de traitement SxPipe et paramétrés en fonction des besoins. Parmi ces derniers figurent ceux ayant pour but de détecter des motifs (noms propres (personnes, lieux, entreprises), adresses, nombres, numéros de téléphone, heures, dates, smileys, URL ou encore adresses e-mail).

Le correcteur lexical intégré à l'étape 4 fonctionne grâce à un système de règles de substitution permettant de remplacer un ou plusieurs caractères par un autre ensemble de caractères. Ces dernières sont créées et pondérées manuellement. Elles peuvent aussi bien concerner des fautes typographiques (fautes de proximité clavier, erreurs de type Damerau-Levenstein,...) que des erreurs phonétiques. En fonction de l'erreur produite, leur coût de correction pourra varier. Un utilisateur de SxPipe pourra ensuite choisir le type de correction qu'il veut mettre en place. Il faut néanmoins noter que ce correcteur peut être très bruyant si son paramétrage est trop lâche (cf. exemple ci-dessous). Les scores appartenant à chaque candidat de correction peuvent, à ce stade, être fournis dans le format suivant : [|score|]. Par exemple, pour la phrase « *Je bois ammoureusement un cafe nespresso* », on obtient le résultat suivant :
{Je} je [|0|] {bois} bois [|0|] {ammoureusement} amoureusement [|20|] {un} un [|0|] {cafe} café [|5|] {nespresso} empressa [|340|]
Comme l'illustre cet exemple, le correcteur proposé dans SxPipe fonctionne relativement bien face à des fautes simples, peu coûteuses. Ses corrections lourdes seront toutefois moins fiables. Il est à noter que, dans l'exemple ci-dessus, la correction a été paramétrée afin d'autoriser de telles corrections, avec des coûts élevés, et de manière à ce que le système ne propose qu'un seul candidat de correction. En réglant autrement la limite du poids de correction autorisé pour une forme donnée, ce type de surcorrection peut être évité.

4.1.3 Conclusion

Bien que SxPipe semble être relativement complet pour procéder au prétraitement d'un texte, il ne répond pas tel quel aux besoins de notre tâche. En effet, même s'il propose de procéder à la détection de nombreux motifs, il ne couvre pas pour autant l'étendue des tokens inconnus pouvant apparaître dans un texte relativement bruité. Comme nous l'expliquions section 1.3.3.1, la dynamique lexicale permet l'intégration d'inconnus pouvant être répartis en quatre catégories :

1. les inconnus provenant de l'ensemble des variantes graphiques potentielles (ou altérations) des tokens existants déjà dans le lexique réel,
2. les inconnus provenant du non-lexique,
3. les inconnus provenant du xénolexique,
4. les inconnus provenant du lexique potentiel.

À ce stade, SxPipe couvre en partie la première de ces catégories ainsi que la seconde. Et, bien qu'il prenne en compte par exemple les fautes lexicales, son système de correction reste très générique et limité. Il pourra notamment passer à côté d'altérations ou de candidats de correction/normalisation qui auraient pourtant pu être facilement appris automatiquement. Par ailleurs, SxPipe ne propose

aucun système de détection pour les inconnus provenant soit du xénolexique, soit du lexique potentiel.

Pour la suite de notre travail, détecter parmi des inconnus quels sont ceux qui sont réellement à normaliser est primordial. En effet, un système qui considérerait tout token inconnu de son lexique comme une altération risquerait d'être très bruyant, de normaliser beaucoup plus de tokens que nécessaire. C'est afin d'éviter cela que nous avons mis en place les modules introduits dans la suite de ce chapitre. Nous commencerons ainsi (en section 4.2) par introduire les différents modules libres compatibles à SxPipe que nous avons mis en place pour normaliser les altérations détectables de manière sûre, provenant de variantes graphiques stables. Puis, nous évoquerons rapidement les modules appliqués pour la détection des tokens inconnus provenant du non-lexique (section 4.3). Enfin, nous nous intéresserons plus particulièrement à la détection des inconnus correspondant à des tokens appartenant au xénolexique (section 4.4) ou à notre lexique potentiel (section 4.5).

4.2 Normalisation des inconnus provenant de variantes graphiques stables

Comme nous l'avons déjà évoqué, une altération peut être décrite en fonction de trois critères : la raison qui a conduit à sa production, le procédé qui a été utilisé pour la produire et le résultat obtenu après la modification du token concerné. Certaines altérations provoquent la création de tokens inconnus qui, par leur forme, sont assez spécifiques pour être identifiables. Ces dernières étant assez fréquentes dans les corpus de type web, nous avons cherché à mettre en place différents systèmes plus spécifiques afin de les détecter et de les normaliser en conséquence. Ces derniers nous permettent de restreindre le nombre de tokens inconnus correspondant à des altérations avant leur passage dans les modules de classification des inconnus. De plus, la normalisation de ces altérations réduit les chances qu'aura notre système de normalisation lexicale (décrit chapitre 5) de proposer des candidats de normalisation fautifs.

Ces différents systèmes mis en place ont pour objectif d'être intégrés comme modules au sein de la chaîne de traitement SxPipe. On conservera donc tout au long de ce travail le format de sortie de SxPipe et par conséquent la notation des tokens d'origine sous la forme de commentaires, ces derniers pouvant être utiles pour certaines opérations. Par ailleurs, si nous procédons à la normalisation d'une forme, nous conserverons cette information au sein des commentaires du ou des tokens correspondants à cette forme. Cette annotation se présentera sous le format suivant : {<F id="EiFj">token◁Type_Normalisation</F>} Forme_normalisée. La notation de cette forme pourra être simplifiée de la sorte : {token$_1$◁Type_Normalisation} Forme_normalisée. Par ailleurs, bien que la détec-

tion des altérations spécifiques puisse être réalisée de manière sûre et fiable, ce n'est pas systématiquement le cas pour la génération de leurs candidats de normalisation. On pourra parfois vouloir conserver une ambiguïté soit entre plusieurs candidats de normalisation soit entre la forme initiale et un ou plusieurs candidats de normalisation. Pour ce faire, nous reprendrons la notion de DAG utilisée par SxPipe et nous proposerons, en sortie de certains de nos modules, des analyses sous forme d'expressions régulières tel que l'exemple proposé ci-dessous :
({token$_1$} Forme_initiale | {token$_1$◁Type_Normalisation1} Forme_normalisée1 | {token$_1$◁Type_Normalisation2} Forme_normalisée2)

Précisons que les formes composées et les entités nommées, dès lors qu'elles sont reconnues comme telles par SxPipe, ont de fortes chances de ne pas être altérées. Elles ne seront par conséquent pas traitées ici. Seront ainsi encore à analyser les formes restantes, considérées comme simples par SxPipe. Ces formes, identiques à des tokens, pourront ainsi correspondre à des tokens simples, à des amalgames altérés ou encore à des composants de formes composées altérées, que ces derniers soient altérés ou non. Ainsi c'est bien sur des tokens que s'effectueront toutes nos analyses.

Les modules présentés dans cette section ont tous pour but de détecter et de normaliser des tokens correspondant à des inconnus. C'est pourquoi il est important de préciser ici ce qu'on entend par inconnu. On considère comme tel tout token qui n'apparaît pas dans un lexique de référence. Dans ce travail, nous avons choisi d'utiliser comme lexique de référence le Le*fff*. Ce lexique de formes a pour intérêt d'être assez complet et d'être déjà employé dans SxPipe. En outre, il s'agit d'un lexique Alexina. Les lexiques Alexina, existants dans de multiples langues telles que l'anglais, l'allemand (DeLex (Sagot, 2014)) ou encore l'espagnol (Le*ffe*(Molinero et al., 2009)), partagent un même format. Notre système se voulant non dépendant d'une langue donnée, accepter en entrée un format de lexique disponible dans plusieurs langues présente un réel avantage. Comme nous serons amenée à travailler sur les tokens, nous utilisons la version tokenisée de ce lexique [4] c'est-à-dire une version contenant tous les tokens figurant dans ce lexique de formes. Toutefois, comme nous l'avons précisé en section 1.3.3.2, les formes contenues dans un lexique informatisé ne représentent qu'une partie de celles appartenant au lexique réel. Pour cette raison, dans certains modules, notre lexique de référence le Le*fff* sera complété par d'autres ressources lexicales proposant une plus large couverture de la langue, tels que le Wiktionnaire ou encore la liste des tokens figurant dans les corpus WaCky [5] (Baroni et al., 2009). Dans

4. Notons que nous avons fait le choix de conserver et de prendre en compte les entités nommées (prénom, nom de villes, etc.) contenues dans ce lexique. Un nom propre pouvant tout autant être altéré qu'un nom ou un verbe.

5. Les corpus WaCKy (Baroni et al., 2009) sont des corpus construits par l'aspiration d'un grand nombre de pages Internet allant d'articles journalistiques à des messages extraits de forums. Ce type de corpus, constitué de nombreux textes de qualité plus dégradée, contient par conséquent de nombreux néologismes. Des corpus WaCKy ont été constitués pour plusieurs langues. Le corpus WaCKy de référence pour le français est frWaC, celui pour l'anglais est

ce dernier cas, nous détaillerons, dans la description de ces modules en question, l'ajout de ces ressources lexicales. Afin de ne pas avoir à réitérer, pour chaque module, la vérification qu'un token figure bien dans le lexique Alexina correspondant, nous annotons en début de traitement toutes les tokens inconnus dans notre texte en employant la notation suivante : {token$_1$} _INC_token.

Dans cette section nous commencerons par présenter les différents modules de normalisation spécifiques ajoutés, puis, nous expliquerons (section 4.2.8) leur intégration dans la chaîne de traitement SxPipe avant de les évaluer (section 4.2.9).

4.2.1 Erreurs d'apostrophe

On retrouve de nombreuses fautes d'apostrophes sur le web. Ces dernières, généralement provoquées non volontairement, sont souvent dues à une faute de frappe. Nous avons relevé trois cas d'erreurs fréquents : (i) le remplacement de l'apostrophe par un espace, (ii) l'insertion d'un espace entre le token élidé et l'apostrophe (iii) la suppression totale de l'apostrophe. Bien que peu gênantes pour la compréhension d'un texte par un humain, ces altérations peuvent avoir des répercussions sur une analyse automatique. Cela peut d'ailleurs se vérifier. Nous avons fait analyser l'énoncé « *J'en veux* » avec et sans erreur d'apostrophe par SxPipe et, comme nous pouvons le constater ci-dessous, les analyses proposées par la chaîne de traitement diffèrent.

- « *J'en veux* » : {J'$_1$} j' {en$_2$} en {veux$_3$} veux
- « *J en veux* » : {J$_1$} J {en$_2$} en {veux$_3$} veux
- « *J ' en veux* » :{J$_1$} J ({'$_2$} '' | {'$_2$} _EPSILON) {en$_3$} en {veux$_4$} veux
- « *Jen veux* » : ({Jen$_1$} Jen | {Jen$_1$} j' {Jen$_1$} en) {veux$_2$} veux

Ainsi, si l'apostrophe est déplacée ou remplacée par un espace, le pronom *je* n'est plus considéré que comme la lettre *J*[6]. Il faut par ailleurs noter que dans le cas où un espace est inséré avant l'apostrophe, cette dernière est alors considérée comme une forme à elle toute seule. Cela produit donc l'ajout d'une forme au sein d'une phrase ce qui complexifie automatiquement son analyse bien que l'étiquette _EPSILON autorisera le système à l'ignorer. La suite de notre travail s'appuyant sur les formes détectées par SxPipe, il est important que ces dernières soient altérées au minimum.

Pour pallier ce problème, nous avons mis en place, manuellement, une série d'expressions régulières ayant pour but de modifier le texte brut avant son découpage

ukWaC et celui pour l'allemand s'intitule deWaC. Il n'existe actuellement aucun corpus WaCKy pour l'espagnol.

6. Ici le J est en majuscule parce qu'il est en tête de phrase, en milieu de phrase la lettre aurait été en minuscule. Le comportement qu'aura SxPipe sur une forme peut en effet varier en fonction de sa place dans la phrase.

en tokens et en formes. Afin d'éviter de traiter des cas particuliers dans lesquels la graphie proposée était valide (ex : « *C'est le jour J allons y !* »), nous nous sommes limitée aux erreurs les plus fréquentes dans nos corpus.

Nous normalisons le cas (i), où l'apostrophe est remplacée par un espace, en nous appuyant sur quelques expressions régulières (cf. table 4.2). La majorité d'entre elles permettent d'ajouter une apostrophe dans le cas où les formes fléchies *ai, as, a, es* sont précédées de la particule de négation *n* ou d'un pronom élidé *c, j, l, m, s, t*. La seconde partie de nos expressions régulières se concentre plutôt sur les cas où les déterminants *un, une* auraient pu être précédés d'une apostrophe et sur les cas où le pronom élidé *qu* est suivi d'une voyelle.

Regex	Exemple
(j) +ai	J ai
(m|l|n|t) +(est ?|ai ?s ?|à)	il m est cher, que tu n ais, qu'as-tu, qu à...
(c|s) +est	c est, s est
(d|l) +une ?	l un, d une, ...
(qu) +[aàeéèêiou].*	qu écrire, qu en, qu il,...

Table 4.2 – Regex permettant la détection d'omission d'apostrophe

Le second cas d'erreur (ii), moins dangereux à corriger, est traité par une seule expression régulière. Cette dernière propose de supprimer un espace dans un texte si celui-ci est suivi d'une apostrophe et est précédé des lettres [*cdjlmnst*] ou de la chaîne *qu*[7].

Enfin, pour le cas (iii), où l'apostrophe a été omise (ex : *lenvie/l'envie*), nous vérifions pour chaque token inconnu commençant par les lettres *j, l, s* ou par *qu*[8] si la chaîne de caractères suivant la lettre correspond à un token existant dans notre lexique. Si c'est le cas, nous normalisons le token inconnu. Ainsi une altération comme *lenvie* sera normalisée en *l'envie*.

Les deux premiers cas d'erreurs (i) et (ii) sont annotés avec les étiquettes ◁AddApos, ◁SuppApos et ◁GlueApos. Le dernier cas d'erreur (iii), considéré comme une agglutination, est normalisé au sein d'un autre module (cf. section 4.2.5). Il est annoté avec l'étiquette ◁Agglu. Des exemples des différentes normalisations faites pour les erreurs d'apostrophes figurent dans la table 4.3.

7. Il est à noter que nous avons proposé des expressions régulières plus précises pour le premier cas d'erreur, cas dans lequel une apostrophe aurait été omise, que dans le second. Cela s'explique par le fait que le risque d'erreur est beaucoup plus grand et nous ne voulons aucunement insérer le moindre bruit dans notre analyse. En effet dans les corpus bruités, faire une expression régulière générale pourrait être plus dangereux. Ainsi, si nous n'avions pas été si restrictive, nous aurions très probablement normalisé l'énoncé « *Ms m un film comme ça[...]* » (« *Mais même un film comme ça[...]* ») en « *Ms m'un film comme ça[...]* ».

8. Ces lettres correspondent aux lettres élidées les plus fréquentes que nous avons constatées dans notre étude de corpus.

	Altération	Sortie txt	Sortie SxPipe
(i)	l idée	l' idée	{l$_1$◁AddApos} l' {idée$_2$} idée
(i)	c est	c' est	{c$_1$◁AddApos} c' {est$_2$} est
(ii)	qu ' un	qu' un	{qu_'$_1$◁GlueApos} qu' {un$_2$} un
(ii)	m 'a	m' a	{m$_1$◁AddApos} m' {'a$_2$◁SuppApos} a
(iii)	jaime	jaime	({jaime$_1$◁Agglu} j' {jaime$_1$◁Agglu} aime | {jaime$_1$} _INC_jaime)
(iii)	quil	quil	({quil$_1$◁Agglu} qu' {quil$_1$◁Agglu} il | {quil$_1$} _INC_quil)

Table 4.3 – Exemples de normalisation pour les erreurs d'apostrophes

4.2.2 Fautes d'accentuation

Les fautes d'accentuation constituant l'une des catégories d'erreurs les plus conséquentes, comme nous l'avons constaté lors de notre analyse de corpus, nous avons fait le choix de les normaliser de manière spécifique. Pour chaque mot inconnu, on vérifie ainsi si sa version non accentuée est connue de notre lexique de référence [9]. Si c'est le cas, on proposera toutes les normalisations alors possibles, c'est-à-dire tous les mots connus dont la version désaccentuée est identique au token considéré. Les tokens normalisés avec ce module sont annotés avec l'étiquette ◁MAJDIA comme nous pouvons le voir dans la table 4.4.

Altération	Sortie au format SxPipe
eleve	({eleve$_1$◁MAJDIA} élevé | {eleve$_1$◁MAJDIA} élève)
ecrire	{ecrire$_1$◁MAJDIA</F>} écrire

Table 4.4 – Exemples de normalisation d'erreurs d'accentuation

4.2.3 La réticence de plume

La réticence de plume ou autocensure est utilisée dans le but de rendre moins intelligible un token portant un sens grossier. Sa création consiste à remplacer une partie des lettres composant ce token par des symboles non alphabétiques (ex. : *m@#%$*). Bien qu'elles ne soient que peu fréquentes (cf. notre étude de corpus, table 3.2 p. 68), leur détection demeure primordiale. En effet, ce type d'altérations intègre souvent des symboles de ponctuation qui peuvent fausser le découpage d'un texte en phrases et en tokens. Dans notre cadre, même s'il s'agit d'altérations, nous ne voulons pas les normaliser, mais uniquement les détecter afin de ne pas les analyser.

9. Notre lexique de référence, le Le*fff*, stocke pour chacun des tokens le composant sa version avec et sans accents.

La détection d'une réticence de plume se fait à l'aide d'une expression régulière. Cette dernière a pour but de correspondre au format des autocensures, c'est-à-dire commencer par une lettre et contenir au moins deux caractères non alphabétiques autres qu'un tiret ou une apostrophe. Tout inconnu correspondant à cette expression régulière sera ainsi considéré comme une autocensure et annoté comme tel avec l'étiquette _AUTOCENSURE (cf. table 4.5).

Altération	Sortie au format SxPipe
m***de	{m$_1$ *$_2$ *$_3$ *$_4$ de$_5$} _AUTOCENSURE
m@#%$	{m@$_1$ #$_2$ %$_3$ $$_4$} _AUTOCENSURE

Table 4.5 – Exemples de normalisation de tokens inconnus étirés

4.2.4 Décompositions

Parmi les altérations se démarquent les tokens dits *décomposés*. Ces derniers sont aisément identifiables puisqu'ils correspondent à des tokens existants qui auraient été segmentés par des tirets (ex : « *ja-mais* » ou « *im-pos-si-ble* »). Pour procéder à leur normalisation, un module dédié parcourt les inconnus d'un texte donné et, s'ils possèdent un ou plusieurs tirets, nous effectuons des tests sur chacun d'entre eux en fonction du nombre de tirets contenus.

Dans le cas où le token ne comporte qu'un seul tiret et que les deux parties segmentées le composant sont formées de plus d'une lettre chacune, nous vérifions si le token correspondant à la concaténation de ces deux parties existe. Si c'est le cas, on considère que ce dernier token qui correspond au pendant normalisé de l'altération traitée. De la sorte, nous normalisons des altérations comme *su-per* tout en évitant de modifier les formes composées comme *franco-russe* qui ne sont pas systématiquement présentes dans notre lexique de référence ou les formes telles que *dual-s* [10].

Si le token que nous analysons est segmenté en plus de deux parties, nous calculons alors l'écart-type des longueurs des parties le composant. Si cet écart-type est inférieur à 2 et que la concaténation de toutes ces parties donne un token existant dans notre lexique de référence, alors on le normalise. Cela pourra ainsi donner lieu à des normalisations de type : *su-bli-me* → *sublime*. Dans le cas où l'écart-type serait supérieur à 2, on peut supposer qu'il s'agit soit d'un token étrangement décomposé, soit d'un ensemble de tokens rattachés par des tirets. Nous vérifions dans un premier temps que cette décomposition n'apparaît pas dans d'autres corpus bruités tels que le corpus WaCky (Baroni et al., 2009). Si ce n'est pas le cas, on considère que plusieurs analyses sont possibles pour ce token et on conserve cette

10. Le *Dual-S* est un type récepteur/décodeur de programmes télévisés destiné à la réception satellite.

ambiguïté. Ainsi, si par exemple le token *acteur-chanteur-compositeur* apparaît dans notre texte, on l'annotera comme ambigu. Lorsqu'un token est normalisé par ce module, cette modification est notée dans les commentaires avec l'étiquette ◁SEGEMPH comme illustrée dans la table 4.6.

Altération	Sortie au format SxPipe
su-per	{su-per$_1$◁SEGEMPH</F>} su-per
in-cro-ya-ble	{in-cro-ya-ble$_1$◁SEGEMPH</F>} incroyable
auteur-chanteur	({auteur-chanteur-compositeur$_1$} auteur-chanteur-compositeur
-compositeur	| {auteur-chanteur-compositeur$_1$◁SEGEMPH} auteur
	{auteur-chanteur-compositeur$_1$◁SEGEMPH} chanteur
	{auteur-chanteur-compositeur$_1$◁SEGEMPH} compositeur)

Table 4.6 – Exemples de normalisation de tokens inconnus décomposés

4.2.5 Agglutinations

Parmi les fautes fréquentes que l'on retrouve souvent dans les corpus provenant du web figurent celles dues à l'oubli d'un espace entre deux tokens. L'inconnu produit par une erreur de ce type correspond à la concaténation de deux tokens.

Nous détectons et normalisons ces cas d'erreurs en générant toutes les segmentations possibles sur les tokens inconnus. Si par exemple nous avons l'inconnu *tropcher*, nous proposerons toutes les segmentations suivantes : *t+ropcher*, *tr+opcher*, *tro+pcher*, *trop+cher*, *tropc+her*, *tropch+er* et *tropche+r*. Nous ne conservons ensuite que les segmentations pour lesquelles les chaînes de caractères constituant les parties droite et gauche de notre inconnu correspondent à des tokens connus de nos lexiques. Ici seule la segmentation *trop+cher* serait conservée et utilisée pour normaliser notre inconnu. Si plusieurs segmentations sont possibles, on les conserve toutes. Par exemple si on a l'énoncé : « *il a récupéréson coli* », le token inconnu *récupéréson* peut correspondre à l'agglutination des deux tokens *récupéré+son* ou des deux tokens *récupérés+on*. Dans notre format de sortie SxPipe, une normalisation réalisée avec ce module sera annotée en commentaire à l'aide de l'étiquette ◁Agglu. Ce module pouvant parfois générer des candidats de normalisation erronés, nous conservons l'ambiguïté entre le token initial et sa version normalisée. On obtiendra par conséquent des sorties telles que celles présentes dans la table 4.7.

4.2.6 Étirements

Les étirements sont des altérations utilisées volontairement afin d'accentuer le caractère expressif d'un message. Ils sont produits en dupliquant une ou plusieurs

Altération	Sortie au format SxPipe
tropcher	({tropcher$_1$◁Agglu} trop {tropcher$_1$◁Agglu} cher | {tropcher$_1$} _INC_tropcher)
tufais	({tufais$_1$} _INC_tufais | {tufais$_1$◁Agglu} tu {tufais$_1$◁Agglu} fais | {tufais$_1$◁Agglu} tuf {tufais$_1$◁Agglu} ais)
récupéréson	({récupéréson$_1$◁Agglu} récupéré {récupéréson$_1$◁Agglu} son | {récupéréson$_1$◁Agglu} récupérés {récupéréson$_1$◁Agglu} on | {{récupéréson$_1$} _INC_récupéréson)

Table 4.7 – Exemples de normalisation de tokens inconnus agglutinés

lettres (généralement des voyelles) trois fois ou plus (ex. : *jamaaaiiiis* ou *quoiii*). Bien que ces altérations soient relativement peu courantes (cf. table 3.2 p. 68), elles sont réalisées volontairement, souvent pour insister sur le sens et la portée d'un token. Elles sont par conséquent importantes au niveau sémantique et pragmatique. De plus, elles sont aisées à traiter.

Pour normaliser ces altérations, on vérifie pour chaque inconnu si certaines de ses lettres sont répétées consécutivement au moins 3 fois. Si c'est le cas, on modifie ce token afin de générer ses versions contenant la ou les lettres étirées au nombre de 1 et 2. Ainsi pour un mot de la forme *abbbbc*, la liste de ses candidats de normalisation sera donc : [abc,abbc]. On génère alors la liste finale de ses normalisations possibles en ne conservant que celles qui figurent dans notre lexique. Si une seule proposition de normalisation est proposée, on considère qu'elle est valide. Dans le cas où plusieurs sont possibles, on conserve l'ambiguïté. Enfin si nous ne parvenons pas à trouver un candidat de normalisation pour un token inconnu étiré, nous conservons son statut inconnu et ses candidats de normalisation. Lorsqu'un token est normalisé par ce module, il obtient l'étiquette ◁ETIR. En sortie, en fonction du token étiré, le texte au format SxPipe proposé en sortie de ce module pourra ressembler aux exemples illustrés dans la table 4.8.

Altération	Sortie au format SxPipe
noooon	{noooon$_1$◁ETIR} non
jamaaaiiiis	{jamaaaiiiis$_1$◁ETIR} jamais
coooool	({coooool$_1$◁ETIR} col | {coooool$_1$◁ETIR</F>} cool)
aléééé	({aléééé$_1$◁ETIR} alé | {aléééé$_1$◁ETIR} aléé | {aléééé$_1$◁ETIR} _INC_aléééé)

Table 4.8 – Exemples de normalisation de tokens inconnus étirés

4.2.7 Lexique de substitutions

Il existe enfin des listes de substitutions qui permettent de passer d'un token erroné non ambigu à sa correction de manière sûre. Les prendre en compte peut être intéressant lorsque l'on veut procéder à la normalisation d'un texte. En effet, elles peuvent permettre de traiter certaines altérations complexes de par leurs graphies très éloignées de leur(s) token(s) d'origine (ex : *TMTC/toi-même tu sais* ou *bjrs/bonjour*). Par ailleurs, elles peuvent simplement être utilisées pour normaliser de manière sûre certaines fautes courantes afin d'améliorer le contexte d'autres fautes plus singulières.

Pour ce faire, nous laissons à l'utilisateur la possibilité d'intégrer à notre système des listes de fautes accompagnées de leur correction. Les fautes peuvent être écrites telles quelles (ex : *example* → *exemple*) ou généralisées à l'aide d'expressions régulières (ex : *(n/N)[éeè](c/ss ?)[eéè](ss ?/c)airr ?(e(s ?)/ement(s ?))* → *$1écessair$4*). Si une de ces altérations figure dans notre texte, elle sera ainsi étiquetée ◁AUTOCORR puis normalisée tel qu'illustré ci-dessous (table 4.9).

Altération	Sortie au format SxPipe
nessecairre	{nessecairre$_1$◁AUTOCORR} nécessaire
TMTC	{TMTC$_1$◁AUTOCORR} Toi TMTC$_1$◁AUTOCORR} même
	{TMTC$_1$◁AUTOCORR} tu TMTC$_1$◁AUTOCORR} sais
oqp	{oqp$_1$◁AUTOCORR</F>} occupé

Table 4.9 – Exemples de normalisation par substitution

À titre d'exemple pour l'utilisateur, nous proposons un ensemble de règles de substitution tirées de la Wikipédia[11]. Cette dernière est composée de 1853 règles.

4.2.8 Intégration dans SxPipe

Les modules présentés ci-dessus ne sont pas ajoutés directement à la suite des modules standard de la chaîne SxPipe. En effet, la chaîne de traitement SxPipe exécute dans un premier temps des modules sur le texte brut avant d'effectuer un premier découpage de ce dernier en tokens puis un second découpage en formes après avoir appliqué une autre série de modules. L'ordre des modules de SxPipe est schématisé dans la figure 4.1.

Cette figure donne une idée de la transformation subie par le texte au fur et à mesure. Ainsi, SxPipe prend bien un texte brut en entrée. Il applique suite à

11. `http://fr.wikipedia.org/wiki/Wikip%C3%A9dia:AutoWikiBrowser/Typos` et `http://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Liste_de_fautes_d%27orthographe_courantes`

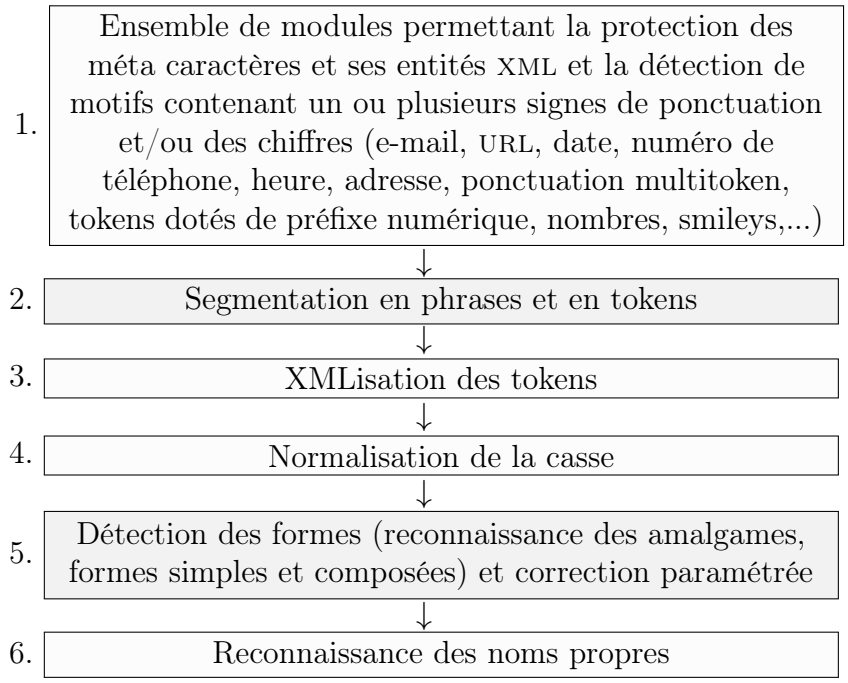

FIGURE 4.1 – Chaîne SXPipe simplifiée

cela une série de détection de motifs qui pourraient nuire à la segmentation de ce dernier, puis découpe ce texte en phrases, en tokens, et ensuite en formes. Il est important de préciser que tant que la segmentation du texte en formes n'a pas été effectuée, la chaîne de traitement demeure déterministe, ce n'est qu'après que le système devient non déterministe.

Parmi les modules que nous souhaitons ajouter à cette chaîne, tous, mis à part la détection d'autocensure, produisent une sortie non déterministe. Par ailleurs, il est aussi à noter que certains modules seront amenés à traiter avec des symboles de ponctuation, c'est le cas de la reconnaissance d'autocensure et de la normalisation des fautes d'apostrophe, ou avec d'autres symboles non alphabétiques, notamment pour la détection de décomposition. Dans ce cas, l'opération est effectuée en deux fois : une première fois avant la segmentation en tokens (pour la normalisation des apostrophes) et le découpage en formes (pour la normalisation des décompositions), puis une seconde fois lorsque la chaîne devient non déterministe. La chaîne obtenue ressemble ainsi à celle proposée figure 4.2.

La table 4.10 reconstitue, quant à elle, la chaîne SXPipe telle que nous l'avons adaptée en ajoutant nos différents prétraitements de normalisation. Elle contient des exemples illustrant le résultat obtenu en sortie de chaque module pour un texte donné. Une rapide comparaison de la chaîne SXPipe simple et de la chaîne adaptée nous donne par ailleurs un aperçu de l'apport de nos modules [12].

12. Il est à préciser que les résultats obtenus avec SXPipe peuvent différer en fonction de son paramétrage. Afin de rester neutre, nous utilisons son mode afin de représenter ses sorties.

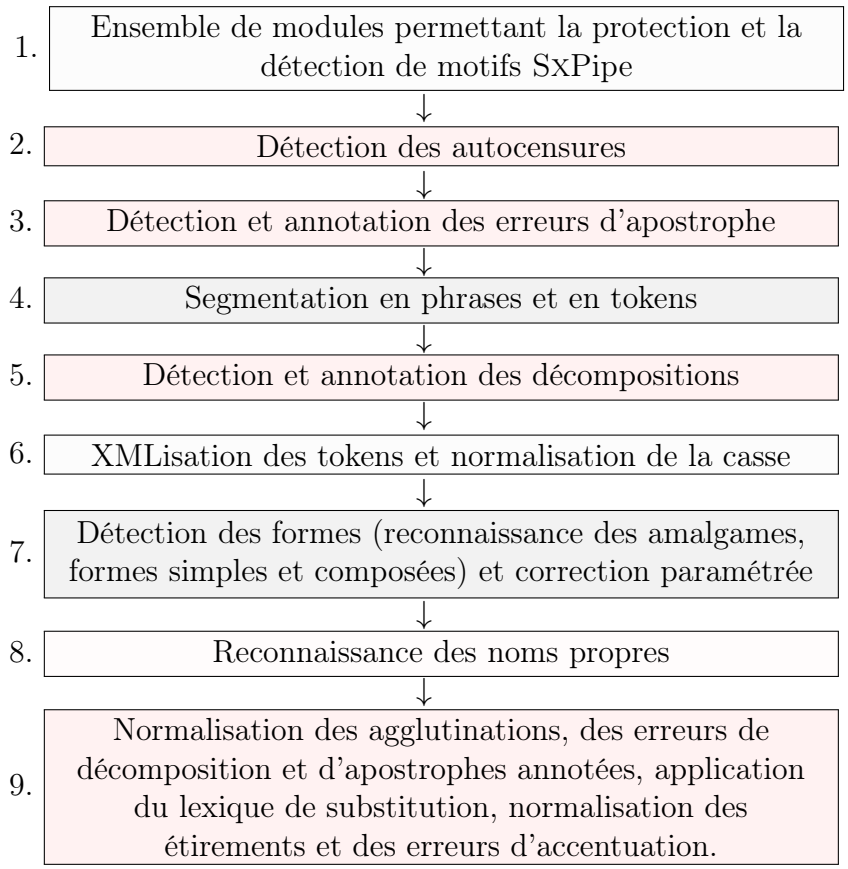

FIGURE 4.2 – Chaîne SxPipe adaptée

Module	Texte	Sortie SxPipe	Sortie SxPipe adaptée
Détection motifs SxPipe	*rdv à 15h ;)*	rdv à {15h} _HEURE {;)} _SMILEY	rdv à {15h} _HEURE {;)} _SMILEY
Détect. autocensures	*Et m***e*	{Et$_1$} Et {m$_2$} m {*$_3$} * {*$_4$} * {*$_5$} *{e$_6$} eh	{Et$_1$} Et {m***e$_2$} _AUTOCENSURE
Détect. err. apostrophes	*L a t 'il fait ?*	L a t 'il fait ?	{L__L'} _AddApos a {t__t'} _AddApos {'il__il} _SuppApos fait ?
Segment. phrases/tokens	*j'ai la 3G*	j' ai la 3G	j' ai la 3G
Détect. Décompositions	*ça sera spec-ta-cu-lai-re*	ça sera spec-ta-cu-lai-re	ça sera {spec-ta-cu-lai-re◁SEGEMPH} spectaculaire
Format XML tokens	*Je veux ce coli*	{Je$_1$} Je {veux$_2$} veux {ce$_3$} ce {coli$_4$} coli	{Je$_1$} Je {veux$_2$} veux {ce$_3$} ce {coli$_4$} coli
Segment. formes	*Je veux ce coli*	{Je$_1$} Je {veux$_2$} veux {ce$_3$} ce {coli$_4$} colis	{Je$_1$} Je {veux$_2$} veux {ce$_3$} ce ({coli$_4$} coli | {coli$_4$◁CorrSXP_40} colis | {coli$_4$◁CorrSXP_60} coolie | {coli$_4$◁CorrSXP_76} soli)
Détect. des NP	*je pars en Italie*	{je$_1$} je {pars$_2$} pars {en$_3$} en ({Italie$_4$} Italie| {Italie$_4$} _LOCATION |{Italie$_4$} _PERSONNE)	{je$_1$} je {pars$_2$} pars {en$_3$} en ({Italie$_4$} Italie | {Italie$_4$} _LOCATION [|2000000003175395 :Repubblica_Italiana|]
Norm. agglutinations	*jetravaille*	{jetravaille$_1$} retravaille	({jetravaille$_1$} _INC_jetravaille | {jetravaille$_1$} je {jetravaille$_1$} travaille | {jetravaille◁CorrSXP_78} retravaille | {jetravaille◁CorrSXP_83} retravaillé)
Norm. décompositions	*ça sera spec-ta-cu-lai-re*	{ça$_1$} ça {sera$_2$} sera {spec-ta-cu-lai-re$_3$} Spec {spec-ta-cu-lai-re$_3$} ta {spec-ta-cu-lai-re$_3$} chu {spec-ta-cu-lai-re$_3$} lai {spec-ta-cu-lai-re$_3$} ré	{ça$_1$} ça {sera$_2$} sera {spec-ta-cu-lai-re$_3$◁SEGEMPH} spectaculaire
Norm. err. apostrophes	*L a t 'il fait ?*	{L$_1$} L {a$_2$} a {t$_3$} t ({'$_4$} " | {'$_4$} _EPSILON) {il$_5$} il {fait$_6$} fait { ?$_7$} ?	{L$_1$◁AddApos} L' {a$_2$} a {t$_3$◁AddApos} t' {'il$_4$◁SuppApos} il {fait$_5$} fait { ?$_6$} ?
Norm. par Substitution	*t es oqp*	{t$_1$} t' {es$_2$} es {oqp$_3$} hop { ?$_4$} ?	{t$_1$◁AddApos} t' {es$_2$} es {oqp$_3$◁AUTOCORR} occupé { ?$_4$} ?
Norm. étirements	*trop cooooool*	{trop$_1$} trop {cooooool$_2$} col	{trop$_1$}trop({cooooool$_2$}cool|{cooooool$_2$}col)
Norm. err. accentuations	*c'est un dègènèrè*	{c'$_1$} c' {est$_2$} est {un$_3$} un ({dègènèrè$_4$} dégénère | {dègènèrè$_4$} dégénéré)	{c'$_1$} c' {est$_2$} est {un$_3$} un {dègènèrè$_4$} dégénère

Table 4.10 – Intégration dans SxPipe

Dans l'optique de mettre en place un système indépendant de la langue, précisons que ces modules peuvent être adaptés ou réutilisés. Ainsi les modules traitants les autocensures, des décompositions, les agglutinations, décompositions et des étirements peuvent fonctionner pour d'autres langues que pour le français. Il en est de même pour celui analysant les erreurs d'accentuation, qui fonctionnera pour toute langue dotée de diacritiques. Le module de normalisation par substitution et celui de normalisation des apostrophes devront quant à eux être adaptés à la langue traitée, ces derniers étant actuellement spécifique au français.

4.2.9 Évaluation

4.2.9.1 Données d'évaluation

Afin d'évaluer nos différents modules, nous avons mis en place deux corpus de test constitués chacun de messages provenant des divers canaux traités par l'entreprise viavoo. Ils regroupent ainsi des messages provenant d'e-mails, de tchats, de formulaires, de sites d'avis, de forums et d'enquêtes de satisfaction. Pour chacun de ces canaux, nous avons veillé à extraire des messages provenant d'organisations différentes afin de couvrir une grande partie des sujets que l'on sera amenée à rencontrer et à traiter. Le premier est un corpus relativement restreint contenant 10 123 tokens. Ce dernier ayant été annoté manuellement, il nous permettra d'évaluer de manière plus complète notre système. Sa composition est détaillée dans la table 4.11.

Canal	Nb tokens	Répartition
E-mails	2170	21,4%
Tchat	1914	18,9%
Sites d'avis	2016	19,9%
Forums	2013	19,9%
Enquêtes satisfaction	2010	19,9%
Total	10123	100%

Table 4.11 – Composition du corpus d'évaluation annoté

Le second est de plus grande taille (100 295 tokens), mais n'est pas annoté. La table 4.12 illustre ainsi la taille couverte par chaque canal distinct dans ce dernier corpus.

4.2.9.2 Résultats

Afin d'évaluer les modules présentés ci-dessus, nous avons passé nos deux corpus d'évaluation dans notre chaîne de traitement SxPipe adaptée. En cas d'ambiguïté

Canal	Nb tokens	Répartition
E-mails	20053	20%
Tchat	20097	20%
Sites d'avis	20011	20%
Forums	20063	20%
Enquêtes satisfaction	20071	20%
Total	100295	100%

Table 4.12 – Composition du corpus d'évaluation non annoté

dans nos sorties, nous considérons qu'une altération est correctement normalisée dès lors qu'au moins un des candidats proposés correspond à une des formes fléchies du lemme auxquelles notre token altéré aurait dû être rattaché. Certains de nos modules, que nous considérons comme potentiellement bruyants, peuvent conserver le token altéré d'origine dans leur sortie. Ce procédé a pour avantage de permettre aux modules le suivant de tenter de suggérer d'autres analyses plus pertinentes et de ne pas considérer celles proposées précédemment comme complètement fiables [13]. Dans ce cas, on considéra cette analyse comme non erronée dans la mesure où elle permet aux autres modules d'analyser autrement ce token inconnu.

Pour la suite de cette évaluation, nous utilisons les notions habituelles de précision, rappel et f-mesure. La précision P correspond au rapport entre le nombre de tokens bien normalisés et ceux pour lesquels nous avons proposé au moins un candidat de normalisation. Le rappel R sera calculé comme le rapport entre le nombre de tokens bien normalisés et le nombre de tokens donnés en entrée nécessitant une transformation. La f-mesure F, quant à elle, sera calculée comme habituellement : $F = 2PR/(P + R)$.

Le premier corpus annoté se verra ainsi évalué à l'aide de ces trois mesures. Le second étant non annoté ne pourra, quant à lui, être utilisé que pour confirmer sur une masse de données plus importante la précision calculée avec notre premier corpus. Il est par ailleurs à préciser que nous n'avons volontairement pas évalué le module de correction par substitution. Ce choix est dû au fait qu'il a été implémenté afin de donner la possibilité aux utilisateurs d'ajouter à leur guise les paires (*altération-normalisation*) de leur choix. La qualité de ce module dépendra par conséquent des ressources fournies par l'utilisateur.

Bien que notre premier corpus d'évaluation ait l'avantage d'être annoté, il ne contient que relativement peu d'altérations. Les résultats obtenus sont par conséquent à prendre avec précaution. En effet, ils sont empreints d'une marge d'erreur importante causée par le faible nombre d'altérations analysées. Ces résultats sont

13. Un système de désambiguïsation est ensuite appliqué en fin de chaîne (cf. chapitre 6) afin choisir quelle analyse nous souhaitons conserver.

présentés dans la table 4.13 accompagnés du nombre d'occurrences trouvé par chacun de nos modules afin de donner un ordre de grandeur du nombre d'éléments concernés par ces résultats.

Altération	Nb Occ	Précision	Rappel	F-mesure
Apostrophe	74	99%	87%	93%
Accentuation	19	89%	100%	94%
Autocensure	2	100%	100%	100%
Décomposition	16	100%	100%	100%
Agglutination	45	100%	100%	100%
Étirement	0	—	—	—

Table 4.13 – Évaluation du système sur le corpus annoté

Il est à noter que, parmi les modules évalués, seul le module de détection des tokens agglutinés conserve systématiquement la forme initiale du token normalisé. Les autres proposent soit de conserver cette forme s'ils jugent son analyse peu fiable, soit de la supprimer imposant ainsi leur analyse.

- On constate que le nombre d'étirements présents dans ce corpus d'évaluation est nul, ce qui est peu surprenant si l'on se réfère à l'étude de corpus que nous avions réalisé (cf. la table 3.2 p. 68 récapitulant le nombre d'étirements trouvés au cours de cette précédente étude). Ce constat est le même en ce qui concerne le faible nombre de tokens correspondant à des cas d'autocensure.
- Notre module de réaccentuation suggère une normalisation valide à tous les tokens altérés contenant une erreur d'accentuation ce qui explique son rappel. Il propose toutefois deux normalisations erronées sur 19 (sur un emprunt non adapté et sur un cas d'incomplétude lexicale) ce qui justifie notre précision de 89%.
- Le module de détection des erreurs d'apostrophe se révèle très peu bruyant. Il ne produit en effet qu'une seule normalisation erronée dans laquelle il ajoute une apostrophe entre deux tokens (*l* et *ors*) alors que ces deux derniers auraient dû être concaténés ensemble. Le rappel de ce module atteint 87%. Ce nombre peut paraître peu satisfaisant, mais il s'explique par le fait que ce module fonctionne à l'aide d'expressions régulières assez précises. Ces dernières assurent la fiabilité des normalisations suggérées, privilégiant ainsi la précision au rappel.
- Notre système de détection des tokens décomposés parvient à détecter tous les cas qui étaient annotés dans notre corpus et ne génère aucune mauvaise normalisation.
- Enfin, le module de détection des tokens agglutinés ne génère lui non plus aucune mauvaise normalisation. En effet, bien que ce dernier tente, à de nombreuses reprises, de normaliser des inconnus ne correspondant pas à

des agglutinations, il conserve systématiquement leur forme initiale, ce qui empêche à ce module de produire du bruit ou d'affecter la suite de la chaîne de traitement avec ses analyses. On peut toutefois noter qu'en conservant systématiquement ce token en plus de son analyse, ce module génère de l'ambiguïté. On constate qu'il ajoute ainsi en moyenne 1,3 candidats de normalisation en moyenne sur ce corpus en plus du token inconnu. Il a notamment tendance à se tromper sur des noms de produits (ex. : *livebox*) et sur certaines fautes d'orthographe (*damenager* ou *sechoir*).

Ainsi qu'expliqué ci-dessus, nous avons réalisé une seconde évaluation de notre chaîne sur un corpus qui a l'avantage d'être plus volumineux. Nous n'avons pour ce dernier évalué que la précision puisqu'il n'est pas annoté. La table 4.14 illustre ainsi les résultats obtenus par chaque module sur ce corpus.

Altération	Nb Occ	Précision
Apostrophe	848	99,6%
Accent	222	90,1%
Autocensure	0	—
Décomposition	137	98,8%
Agglutination	513	100%
Étirement	14	100%

Table 4.14 – Éval du système sur le corpus non annoté

On peut ainsi constater que nos modules conservent de très bonnes précisions, presque identiques à celles obtenues avec le corpus précédent. Seule celle obtenue par notre système de détection de décomposition décroit légèrement de 100% à 98,8%. Cela est dû à deux erreurs d'analyses. La première concerne la forme *sous-traite* (forme fléchie du verbe *sous-traiter*), absente du Le*fff*, que notre module propose de normaliser en *soustraite* (forme fléchie du verbe *soustraire*). La seconde est réalisée sur la chaîne « *c'est plus cher-chez XXX* ». Notre système est induit en erreur par l'absence d'espace avant et après le tiret et propose de normaliser *cher-chez* en *cherchez*.

4.3 Détection des inconnus provenant du non-lexique

La section précédente décrit comment nous traitons les altérations aisément détectables. Cela ne nous permet d'écarter toutefois qu'une faible partie des inconnus. Bon nombre d'entre eux ont une origine difficile à retracer et sont considérés comme provenant du non-lexique (cf. section 1.3.2). Appartiennent à cette catégorie toutes les entités nommées au sens large, c'est-à-dire : les noms propres de

lieux, personnes ou organisations, mais aussi les e-mails, les URL, les adresses, les heures, les dates ou encore les nombres.

Comme expliqué ci-dessus, la chaîne de traitement SxPipe propose déjà de réaliser ce type de détection. Nous avons donc fait le choix d'appliquer les modules correspondant à la détection de ces motifs plutôt que d'en réimplémenter de nouveaux. La table 4.15 donne ici un exemple des résultats et des annotations proposés par ces différents modules [14].

Altération	Exemple	Annotation SxPipe
Personne	*Les frères Cohen*	{(Les$_1$ frères$_2$ Cohen$_3$} _PERSON [|1000000002964224 :Joel_et_Ethan_Coen|] | {Les$_1$} Les {frères$_2$} frères {Cohen$_3$} Cohen)
Lieu	*Venise*	({Venise$_1$} Venise | {Venise$_1$ } _LOCATION [|2000000003164603 :Venezia|])
Organisation	*Apple*	({Apple$_1$} Apple | {Apple$_1$ } _COMPAGNY [|1000000000000063 :Apple|])
e-mail	*monmail@mail.com*	{monmail@mail.com$_1$} _EMAIL
URL	*www.monsite.com*	{www.monsite.com$_1$} _URL
nombre	*0612345678*	{0612345678$_1$} _NUMBER
date	*12-02-15*	{12$_1$ -$_2$ 02$_3$ -$_4$ 15$_5$} _DATE_arto
heure	*09h13*	{09h13$_1$} _HEURE
adresse	*77 bd Foch - Paris*	{77$_1$ bd$_2$ Foch$_3$ -$_4$ Paris$_5$} _ADRESSE

Table 4.15 – Exemple de motifs annotés par SxPipe

4.4 Détection des inconnus provenant du xénolexique

Parmi les différents types d'inconnus figurent également ceux qui proviennent du xénolexique. Nous rassemblons sous cette appellation tous les tokens empruntés à une langue étrangère ainsi que les smileys et les nouvelles onomatopées et interjections qui représentent de manière graphique le monde qui nous entoure (comme expliqué à la section 1.3.3.1).

En ce qui concerne la détection de smileys, elle est effectuée via un ensemble d'expressions régulières appliquées dans un module SxPipe. Nous nous sommes contentée ici d'enrichir la liste des smileys que ce module était capable de reconnaître [15]. La détection des nouvelles onomatopées et des nouvelles interjections

14. Les informations entre crochets et barres verticales associées à certaines entités sont des informations de liage : numéro de l'entité dans la base de référence Aleda (Sagot et Stern, 2012) et forme normalisée de cette entité.

15. Le module SxPipe de détection des smileys se contentait jusqu'alors de reconnaître les smileys les plus courants tels que : :) , :-) , :'(, :D , :-(, :P...

n'est, quant à elle, pas très pertinente. En effet, en plus du fait qu'il soit difficile de prédire quelles seront les prochaines onomatopées ou interjections créées, les tokens correspondant à cette catégorie d'inconnus sont généralement à des variantes d'interjections ou d'onomatopées déjà existantes. Nous évoquerons donc ces dernières à la section 4.5.

Cette section se concentrera ainsi sur les emprunts non adaptés. Nous ne voulons pas tenter ici de les normaliser d'une quelconque façon (en proposant la traduction la plus proche de la notion désignée par exemple). Cela reviendrait à modifier trop grandement le texte initial. Le système que nous proposons (Baranes, 2012) a donc pour vocation de détecter et d'annoter automatiquement les emprunts non adaptés afin d'empêcher les modules suivants de les analyser et de les modifier. Il apparaît que la majorité des emprunts réalisés correspondent à des tokens anglais. Sur les 36 000 tokens que nous avons parcourus dans nos corpus français afin de réaliser une annotation de ces emprunts, nous n'avons en effet rencontré que des tokens anglais. Pour cette raison, nous choisissons ici, non pas de détecter la langue de chaque token inconnu, mais de déterminer pour chaque inconnu s'il s'agit d'un token dans la langue traitée, ici le français, ou d'un token provenant de l'anglais [16]. Les inconnus dits *français* correspondent alors généralement à des altérations ou encore à des néologismes (cf. exemples 1a et 1b), ceux dits *anglais* étant des tokens non modifiés de l'anglais (cf. exemples 1c et 1d).

(1) a. *Il y a marqer ke la carte n'est pas disponible*

b. *J'ai tellement débranché pour rebooter, reseter que je pourrais le faire les yeux fermés*

c. *y a-t'il la possibilité de les utiliser online ?*

d. *OMFG si no fake go sinon joli montage*

Bien que cette tâche puisse sembler très similaire aux travaux réalisés dans le domaine de la détection de langue, elle s'en écarte beaucoup. En effet, comme nous l'avons expliqué en section 2.1.2, cette tâche cherche à analyser non pas l'ensemble d'une phrase, mais un token isolé. Nous ne pouvons ainsi pas nous appuyer sur le contexte du token dont on veut connaître la langue ni sur les tokens grammaticaux présents dans la phrase. Pour réaliser notre tâche, nous avons choisi d'implémenter un système de classification.

16. N'ayant pas fait d'étude de corpus approfondie pour d'autres langues que le français, il est possible que cette approximation soit à adapter en fonction de la langue traitée.

4.4.1 Mise en place du système de classification

4.4.1.1 Données d'apprentissage

Un classifieur permet de prédire la classe d'un objet parmi un ensemble de classes données. Les classes proposées par notre classifieur sont le français (la langue traitée) et l'anglais (qui correspond à la classe des emprunts adaptés). Un corpus d'entraînement a donc été mis en place pour chacune d'entre elles. Le type de corpus choisi pour réaliser cet entraînement peut avoir un impact sur nos résultats. Nous avons par conséquent réalisé nos tests avec trois types corpus distincts afin d'arrêter notre choix :

1. Des corpus constitués de ressources lexicales, qui prennent en compte les formes fléchies de chaque lemme du français et de l'anglais, tel que le lexique Le*fff* du français (Sagot, 2010) et sa contrepartie pour l'anglais Enlex.

2. Des corpus correspondant à des textes *non bruités*, comme des corpus journalistiques, des extraits de livres ou encore des articles Wikipédia.

3. Des corpus plus *bruités*, tels que les corpus WaCKy (Baroni et al., 2009) qui sont des corpus construits par l'aspiration d'un grand nombre de pages Internet (allant d'articles journalistiques à des messages extraits de forums). Ce type de corpus contient donc en quantité importante des textes de qualité dégradée et contenant de nombreux néologismes, plus proches de ceux que nous avons à traiter que les corpus « propres ». Des corpus WaCKy ont été constitués pour plusieurs langues. Le corpus WaCKy de référence pour le français est frWaC, celui pour l'anglais est ukWaC.

Nous avons entraîné nos systèmes de classification, décrits ci-dessous, sur des corpus relevant de ces trois cas. Les résultats pour le corpus provenant de ressources lexicales, pertinentes par leur richesse en tokens distincts, n'ont toutefois pas été concluants. Nos évaluations nous ont en effet permis de constater que la présence de trop nombreux mots rares avait un impact sur la qualité de nos modèles. Le second corpus composé de textes *non bruités*, constitué de Wikipedia (français et anglais), du corpus Brown (anglais) et du corpus de l'Est Républicain (français), s'est avéré légèrement meilleur, mais insuffisant à cause de la différence de qualité rédactionnelle entre les données d'entraînement et les données de référence. Enfin, de meilleurs résultats ont été obtenus en entraînant notre système sur les corpus WaCKy, ces derniers étant plus proches des nôtres.

Les systèmes de classification décrits par la suite auront, par conséquent, tous été entraînés sur ces derniers corpus. FrWaC ne contient pas que des tokens du français, mais aussi des inconnus (tokens mal orthographiés, néologismes ou emprunts). Toutefois, pour notre classifieur, un token à annoter *anglais* (donc emprunté) sera généralement plus caractéristique, statistiquement, de ukWaC que de frWaC, et ce, même s'il apparaît dans ce dernier.

4.4.1.2 Systèmes de classification

Pour réaliser notre classification, nous avons mis en place, dans un premier temps, une *baseline* puis un système de classification reposant sur trois jeux de traits.

Baseline
Notre baseline correspond à un système de classification naïf qui repose sur les fréquences des tokens dans frWaC et dans ukWaC. Nous faisons l'hypothèse que les tokens présents dans les données que nous avons à traiter ont de fortes chances d'apparaître dans ces corpus.

On compare ainsi le nombre d'occurrences de chaque inconnu dans les corpus ukWaC et frWaC. S'il s'avère que l'inconnu traité est plus fréquent dans le premier, on considère qu'il s'agit d'un emprunt non adapté, il est alors annoté *anglais*, sinon on l'annote comme *français*. Si l'inconnu n'apparaît dans aucun des deux corpus, ce système lui attribue aléatoirement une des deux catégories.

Système proposé
Pour aller au-delà de ce classifieur naïf, nous avons défini trois jeux de traits modélisant les tokens de nos corpus d'entraînement. Ces derniers sont illustrés dans la table 4.16.

1. Comme expliqué à la section 2.1.2, les travaux sur la reconnaissance de langue s'appuient fortement sur les n-grammes. Nous avons par conséquent extrait pour chaque token de frWaC et ukWaC les n-grammes qui les composent [17] et nous avons construit un trait booléen pour chacun d'entre eux. Nos diverses expériences s'appuient soit sur une seule classe de n-grammes (par exemple, seulement les trigrammes), soit sur une combinaison de n-grammes (par exemple, les bigrammes et les trigrammes).

2. Nous avons également ajouté des traits booléens issus de la discrétisation du rapport entre la fréquence d'un mot donné dans frWaC et celle du même token dans ukWaC. Cette discrétisation a été réalisée comme suit. Nous avons calculé le rapport fréquence de tous les tokens compris dans nos données francophones et anglophones. Lorsque la fréquence d'un token est supérieure ou égale à 1 il est considéré comme français, sinon comme anglais. Pour chacune de ces deux langues, nous avons réparti ces résultats en 4 classes de taille identique. Cela nous permet donc d'obtenir 8 classes au total. L'utilisation de ces traits est indiquée par l'abréviation *freq-ratio* dans les tableaux ci-dessous, où les classes obtenues par discrétisation sont nommées F-R et sont numérotées de 1 à 8 (F-R1 à F-R8) [18].

17. n varie de 1 à 4 dans les résultats rapportés ici.
18. Les traits F-R1, F-R2, F-R3 et F-R4 permettent de signaler qu'un token est plus présent en français, les autres indiquent qu'un token est plus fréquent en anglais.

3. L'inconvénient des traits de type freq-ratio est qu'ils ne prennent pas en compte la significativité statistique du rapport de fréquences : un mot attesté une fois dans l'un des corpus et deux fois dans l'autre sera dans la même classe qu'un mot attesté 1 000 fois dans le premier et 2 000 fois dans le second. C'est pourquoi nous avons également réalisé des expériences en utilisant comme traits des classes de t-test (ou test de Student), permettant de mesurer la significativité de l'écart entre la fréquence d'un mot dans frWaC et celle de ce même mot dans ukWaC. Pour ce faire, nous définissions F et U comme le nombre de tokens total contenu respectivement dans frWaC et ukWaC et $f(t)$ et $u(t)$ comme le nombre d'occurrences du token t dans frWaC et ukWaC puis nous réalisons l'opération suivante pour calculer la valeur t-test d'un token t :

$$\text{t-test(t)} = \frac{\frac{f(t)}{F} - \frac{u(t)}{U}}{\sqrt{\varphi(1-\varphi)(\frac{1}{F}+\frac{1}{U})}}$$

φ étant ici égale à $(f(t)+u(t))/(F+U)$. La discrétisation de nos données a ensuite été réalisée en 8 classes ici aussi (4 classes correspondant au français et 4 autres pour l'anglais). Les traits résultants de cette opération, indiqués par la mention *t-test* dans les tableaux ci-dessous, sont nommés T-T et peuvent être ainsi échantillonnés entre 1 et 8.

Token Inconnu	nb occ. ukWaC/ frWaC	Trait 1 : 2-grammes	Trait 2 : freq-ratio [19]	Trait 3 : t-test
access	797 734/ 8 898	_a, ac, cc, ce, es, ss, s_	F-R6 $(33 \leq 89 < 100)$	TT5 $(-6476 \leq -728 < -11)$
vanquish	641/51	_v, va, an, nq, qu, ui,...	F-R7 $(3 \leq 12 < 33)$	TT5 $(-6476 \leq -18 < -11)$
activié	0/27	_a, ac, ct, ti, iv, vi,...	F-R1 $(0 \leq 0 < 0,01)$	TT3 $(3,6 \leq 6,2 < 16)$
regler	12/970	_r, re, eg, gl, le, er, r_	F-R2 $(0,01 \leq 0,01 < 0,05)$	TT1 $(16 \leq 37 < 7425)$

TABLE 4.16 – Illustration des traits de notre module : n-grammes (ici bigrammes), freq-ratio et t-test

Afin de construire nos données d'apprentissage, nous avons, dans un premier temps, assigné la classe *français* à tous les tokens de frWaC et la classe *anglais* à tous les tokens de ukWaC. En plus de cela, chacun de ces tokens s'est vu assigné les diverses combinaisons des traits présentées ci-dessus. Puis, dans un second

19. Traits représentés ainsi : « Classe (x<val<y) » : Classe contenant un mot dont la valeur est comprise entre x et y.

temps, nous avons entraîné ces différentes données sur le système de régression binomiale implémenté dans MegaM [20] (Daumé III, 2004). Chaque combinaison de traits (par exemple, bigrammes + t-test) conduit à un modèle différent. Face à un inconnu à classifier, nous extrayons ainsi ses traits en fonction du modèle sur lequel nous voulons nous appuyer, puis nous calculons la prédiction proposée par ce modèle au vu de ces traits.

4.4.2 Évaluation

4.4.2.1 Données d'évaluation

Notre système est évalué à l'aide d'un corpus contenant 1 128 inconnus nous avons manuellement annoté *français* ou *anglais*. Nous avons par conséquent conservé, pour chaque langue, les 564 premiers inconnus rencontrés dans nos données. Bien qu'ils figurent en quantité égale dans notre corpus, ces nombres ne sont pas représentatifs de leurs fréquences d'apparition [21]. Parmi les tokens annotés *français* figurent des tokens altérés, des néologismes et des emprunts adaptés. Ceux annotés *anglais* correspondent à des tokens propres au monde du web, à des noms de jeux/films non traduits et à des altérations. La table 4.17 illustre ce corpus en donnant un échantillon des 15 premiers inconnus annotés pour nos deux classes.

Inconnus *français*	*abitacle, abonment, achet, actionet, activié, additionels, adébloquée, adhére, adoore, afi, agreabl, aimmerais, aixenProvence, ala*
Inconnus *étranger*	*access, add, advanced, advantage, adventure, adventures, after, again, agency, agreement, airline, airport, all, allOffTheLights, american*

TABLE 4.17 – Échantillon de mots inconnus présents dans les données de référence

Bien que ce corpus de test soit constitué uniquement de tokens absents de notre lexique de référence, précisons que certains d'entre eux figurent toutefois dans les corpus WaCKy ainsi que le montre la table 4.18.

Tokens	Annotés *français*	Annotés *étranger*	Total
dans les corpus WaCKy	402	556	958
absents des corpus WaCKy	162	8	170
Total	564	564	1 128

TABLE 4.18 – Répartition des inconnus du corpus test dans les corpus WaCKy

20. http ://www.cs.utah.edu/∼hal/megam/

21. Environ 26 000 mots ont été parcourus pour trouver les inconnus *français* et près de 34 000 pour les *anglais*

En constituant ce corpus, nous évaluons les performances de notre système de manière isolée. Ce module sera toutefois appliqué à la suite de ceux décrits précédemment. Ses résultats pourront par conséquent être affectés par les résultats de ces étapes préalables dans notre chaîne de traitements.

4.4.2.2 Résultats

Pour cette évaluation, nous nous appuyons sur les taux d'erreur obtenus par chaque modèle sur notre corpus de test. L'existence d'un token dans frWaC et ukWaC ayant des conséquences sur nos résultats, nous commencerons par décrire les résultats obtenus uniquement avec les tokens présents dans ces corpus (table 4.19) puis les résultats obtenus avec ceux qui en sont absents (table 4.20) pour enfin évaluer ce corpus dans son intégralité (table 4.21).

Considérons la table 4.19 qui contient les taux d'erreur de nos modèles en fonction des traits choisis et combinés dans le cas où un token de notre corpus d'évaluation apparaît aussi dans notre corpus d'apprentissage. La pertinence de nos traits s'appuyant sur la fréquence d'un token apparaît nettement ici. Avec les unigrammes, ce taux d'erreur diminue ainsi fortement en passant de 0,305 à 0,066. Notre baseline, s'appuyant sur la fréquence des tokens connus du corpus WaCKy, obtient un moins bon taux d'erreur (0,073) que ceux obtenus par notre système lorsque l'on combine cette fréquence à nos traits n-gramme. Ce constat est satisfaisant puisqu'il montre l'effet positif de ces traits sur nos résultats.

Nous avons ensuite évalué les mots absents des corpus WaCKy (cf. la table 4.20). Dans ce contexte, les résultats obtenus avec nos traits de fréquence et notre baseline (50%) ne sont pas significatifs puisqu'ils s'appuient sur la présence de ces tokens dans nos corpus. Les résultats des n-grammes seuls ne sont, quant à eux, pas surprenants puisqu'ils sont similaires à ceux des mots connus (table 4.19).

La dernière table 4.21 présente les taux d'erreur obtenus lorsqu'on évalue la totalité de notre corpus de référence. Il montre que si nous utilisons uniquement des modèles n-grammes, nos modèles sont peu satisfaisants. Notre baseline, dont le taux d'erreur général est de 0,136, se révèle même meilleure. Cela s'explique par le fait que les corpus frWaC et ukWaC contiennent beaucoup de tokens que nous souhaitons annoter (958/1128). Par ailleurs, on constate que le taux d'erreur de nos modèles n-gramme décroit au fur et à mesure que la valeur de n augmente. En effet, plus la taille d'un n-gramme croît plus on a de fortes chances d'y stocker des tokens entiers. Les modèles combinant les n-grammes et les traits de fréquence freq-ratio et t-test sont pour ces mêmes raisons plus performants. Enfin, les résultats obtenus par la combinaison des n-grammes et du t-test valide bien l'idée qu'avoir des valeurs statistiques plus significatives du rapport de fréquence permet d'optimiser nos résultats. Notre modèle atteint 90% de bonnes classifications. Ces résultats sont satisfaisants. En effet, si on se réfère par exemple à ceux de Grefenstette (1995), on constate que sur un texte français restreint à un ou deux

mots il obtient 69,2% de bonnes détections avec un modèle trigramme et 30,8% avec un modèle linguistique qui ne prend en compte que les mots courts de la langue.

	n-grammes	n-grammes + freq-ratio	n-grammes + t-test
1-grammes	69,5%	92,7%	93,4%
2-grammes	78,4%	92,7%	93,0%
3-grammes	83,3%	92,7%	92,0%
4-grammes	88,1%	91,8%	91,8%
1 à 2-grammes	78,8%	92,7%	93,0%
1 à 3-grammes	84,3%	92,6%	91,8%
2 à 3-grammes	83,1%	92,6%	92,0%

TABLE 4.19 – Précision de nos modèles sur les mots présents dans le WaCKy

	n-grammes	n-grammes + freq-ratio	n-grammes + t-test
1-grammes	67,8%	04,7%	66,7%
2-grammes	77,8%	11,7%	74,3%
3-grammes	84,8%	21,6%	80,1%
4-grammes	86,6%	32,2%	78,9%
1 à 2-grammes	76,0%	09,9%	73,7%
1 à 3-grammes	80,7%	12,9%	79,5%
2 à 3-grammes	83,0%	25,7%	81,9%

TABLE 4.20 – Précision de nos modèles sur les mots absents du WaCKy

	n-grammes	n-grammes + freq-ratio	n-grammes + t-test
1-grammes	69,3%	79,4%	89,3%
2-grammes	78,3%	80,5%	**90,2%**
3-grammes	83,5%	82 1%	**90,2%**
4-grammes	87,1%	82,8%	89,8%
1 à 2-grammes	78,4%	80,2%	**90,1%**
1 à 3-grammes	23,8%	80,6%	89,9%
2 à 3-grammes	23,1%	82,5%	**90,5%**

TABLE 4.21 – Précision de nos modèles sur la totalité du corpus WaCKy

Ce module, bien qu'évalué pour le français, pourrait être adapté à d'autres langues. En fonction de la langue étudiée, la principale langue d'origine d'un emprunt pourra changer. Il est donc à noter que plus la langue source et la langue cible d'un emprunt seront différentes, plus notre système sera performant. Deux langues trop proches risqueraient d'induire en erreur notre système.

Lors de sa sortie au format SXPipe, un inconnu traité par ce module sera annoté avec l'étiquette ◁ETR. Ainsi si notre module est confronté au token *gamer*, il

proposera l'analyse suivante : ({gamer$_1$◁ETR} gamer | {gamer$_1$◁ETR} gamer). Tandis que pour *what* on aura :{what$_1$◁ETR} what.

4.5 Détection des inconnus provenant du lexique potentiel

La dernière catégorie d'inconnus identifiables correspond aux inconnus provenant du lexique potentiel. Comme défini en section 1.3.2, le lexique potentiel correspond à l'ensemble des tokens pouvant être créés en appliquant les règles lexicogéniques de la langue sur notre lexique réel. Cet ensemble comprend par conséquent la totalité des néologismes [22] pouvant être créés. Nous avons, par ailleurs, fait le choix de considérer les onomatopées et interjections altérées comme appartenant à cette catégorie aussi.

La détection de ce dernier type d'altérations est réalisée à l'aide d'une liste des onomatopées et interjections les plus usuelles. Nous représentons ces dernières sous forme d'expressions régulières afin que cette représentation puisse correspondre à toutes les altérations du token en question [23]. Tout token reconnu par ce module sera soit normalisé ainsi {mouaaahahahhahahh$_1$◁ONOMATOPEE} mouahaha, soit annoté comme tel : {mouaaahahahhahahh$_1$ } _ONOMATOPEE en fonction des besoins de l'utilisateur.

La présence de néologismes au sein d'un texte, quant à elle, correspond à une problématique plus générale fréquemment retrouvée dans les travaux réalisés en traitement automatique des langues. Comme évoqué précédemment, nous avons partagé avec le projet ANR EDyLex le besoin de détecter automatiquement ces derniers. En effet, nous souhaitons ici pouvoir les étiqueter afin de ne pas les normaliser. Dans le cas d'EDylex, dont l'objectif principal était l'acquisition dynamique de nouvelles entrées dans les lexiques existants, il était nécessaire de déterminer parmi tous les inconnus ceux qui correspondent à des néologismes afin de les ajouter à une base lexicale. Le système de détection des néologismes décrits ci-dessous a ainsi été réalisé avec les acteurs du projet EDyLex chargés de cette tâche [24]. En plus de la mise en place de ce système, cette collaboration nous a permis de réaliser une étude plus approfondie du mécanisme de construction des néologismes ainsi qu'une observation plus précise des travaux réalisés dans ce

22. Puisqu'il ne s'agit pas d'inconnus, nous ne traitons pas des cas où une forme graphique connue est employée avec une catégorie inconnue du lexique (conversion) ou avec un sens nouveau (néologie sémantique).

23. Prenons l'exemple de *haha*, cette interjection peut être reprise de multiple façon : *hahaha, ahahah, hahahaaha, etc.* Nous avons pour cela entré cette interjection comme ceci a ?h+a+(h+a+)+h* dans notre lexique.

24. Benoît Sagot, Damien Nouvel et Virginie Mouilleron

domaine (les résultats de cette collaboration ont été décrits dans notre article : Sagot et al., 2013). Nous nous concentrerons principalement dans cette section sur la mise en place et l'évaluation de notre système de détection des néologismes.

4.5.1 Système de détection des néologismes

Avant l'implémentation de ce système, une étude de corpus a été réalisée sur des textes AFP devant être traités par le projet EDylex (Sagot et al., 2013). Elle a notamment permis d'étudier de manière plus détaillée la nature et la construction des tokens considérés comme néologismes dans ce type de corpus. Parmi les inconnus susceptibles d'entrer dans la catégorie traitée ici, nous avons relevé trois cas de figure :

1. le cas où le token concerné est un néologisme dérivationnel [25],
2. le cas où le token est un néologisme compositionnel [26],
3. le cas où le token en question ne correspond pas réellement à une création récente, mais plutôt à une incomplétude lexicale.

Contrairement à ceux trouvés dans le corpus AFP, nous avons constaté après une observation de nos données viavoo que les néologismes produits sur le web étaient majoritairement dérivationnels et que les affixes permettant leur construction ne correspondaient pas toujours aux plus fréquents relevés par le projet EDyLex. Le système présenté ici est une chaîne de traitement qui vise à couvrir tous les cas relevés.

Les dérivés correspondant aux cas les plus fréquents dans nos corpus, nous nous sommes particulièrement concentrée sur la mise en place d'un système par analogie les détectant. Les acteurs EDyLex se sont quant à eux principalement appliqués à mettre en place des procédés afin de réduire le nombre de tokens considérés comme inconnus à cause de l'incomplétude lexicale et de détecter automatiquement les néologismes compositionnels. Nous commencerons ici par décrire la mise en place de leurs modules puis nous décrirons plus en détail le travail que nous avons réalisé sur la détection des néologismes dérivationnels. Il est à préciser que ce système, ayant été implémenté pour répondre à deux objectifs différents, devra non seulement détecter les inconnus correspondant à des néologismes mais aussi en extraire les informations utiles pour la création d'entrées lexicales. Nous avons

25. Nous limitons ici les opérations de dérivation aux opérations de préfixation et/ou de suffixation étant donné que nous travaillons sur des langues (français, espagnol ou encore anglais) où c'est très majoritairement voire presque exclusivement le cas.

26. Par composition nous faisons ici référence aux néologismes construits à partir d'unités lexicales combinées et adaptées afin de n'en former plus qu'un.

pour cela besoin de connaître, pour chaque création lexicale détectée, sa forme de citation et sa classe flexionnelle [27].

4.5.1.1 Atténuation de l'incomplétude lexicale

Les notions d'inconnu et de néologisme étant définies par rapport à un lexique de référence, le Le*fff*, une façon simple de les traiter consiste à les rechercher dans d'autres ressources lexicales librement disponibles. Nous avons ainsi fait appel au Wiktionnaire (`fr.wiktionary.com/`), dictionnaire collaboratif, à Morphalou (Romary et al., 2004), lexique morphologique extrait du TLFi, et à ProLexBase (Maurel, 2008), base de noms propres incluant de nombreux gentilés.

Toutefois, l'enrichissement du Le*fff* avec des entrées lexicales manquantes extraites de ces ressources nécessite de rendre ces dernières compatibles avec le Le*fff*, en les transformant en un inventaire d'entrées lexicales associant une forme de citation à une des classes flexionnelles du Le*fff*. Pour chacune de ces trois ressources, nous avons donc utilisé des outils de conversion automatique vers le formalisme Alexina, puis avons projeté les classes flexionnelles obtenues vers celles utilisées par le Le*fff* (Sagot, 2016, chap. 4, sect. 4.4) . Ce processus, bien que nécessairement imparfait, a permis de détecter des erreurs dans les trois ressources d'origine [28]. Le Wiktionnaire, Morphalou et ProLexBase ont été ainsi transformés en des lexiques Alexina partageant la même grammaire morphologique que le Le*fff* et comprenant respectivement environ 1 million, 400 000 et 125 000 entrées produisant au total 1 100 000 formes fléchies distinctes, parmi lesquelles 700 000 ne sont pas couvertes par le Le*fff*. Ainsi que nous l'expliquions section 1.2, nous ne souhaitons pas ici élargir la couverture du Le*fff*, les entrées lui faisant défaut correspondant souvent à des formes peu usitées, trop spécifiques, voire erronées. Le module introduit ici recherche par conséquent les inconnus dans ces différentes ressources et propose autant d'analyses qu'il trouve d'entrées.

Ainsi, sur un inconnu tel que *blablatez*, ce module retrouvera cette entrée dans le Wiktionnaire et l'annotera comme tel avec l'étiquette ◁NEO suivie des informations morphosyntaxiques (lemme, catégorie grammaticale et classe flexionnelle) lui étant rattachées entre des balises [| et |]. La sortie rendue par ce module pour cet inconnu sera donc de la forme suivante :
{blablatez$_1$◁NEO[|frW :blablater :v-er !std :V_P2p ;frW :blablater :v-er !std :V_Y2p|]} blablatez.

27. Ces informations seront indiquées sous le même format que celui utilisé dans le Le*fff*. Ainsi une classe flexionnelle pourra être de la forme : *v-er :std, nc-2m* ou encore *adj-4*

28. Par exemple une entrée lexicale peut se retrouver à associer une forme de citation avec une classe flexionnelle. Ce qui la grammaire morphologique du Le*fff* considère comme incompatible

4.5.1.2 Détection des néologismes compositionnels

Afin de traiter les expressions construites par composition, les acteurs du projet EDyLex ont commencé par se focaliser sur les composés dotés d'un ou plusieurs tiret(s) et ont ainsi tenté de décrire les différents éléments la constituant. Cette identification des composants leur permet ainsi d'interroger le Le*fff* pour y faire des recherches distinctes en fonction du cas analysé. Trois cas sont ainsi recherchés, par ordre de préférence :

(i) les expressions multi-mots : on cherche donc ici une expression dans laquelle les tirets sont remplacés par un blanc (ex. : *chauffe-main*),

(ii) les compositions : on procède à la recherche de chaque composant séparément (ex. : *satiristes-polémistes*),

(iii) les compositions avec adaptation : dans le cas où ce néologisme est constitué de deux composants uniquement, on recherche le dernier composant tel quel puis on recherche les formes de citation partageant un préfixe commun avec le premier des composants [29] (ex. : *politico-judiciaire*). Il est à noter que ce dernier cas suit une règle de création lexicale propre au français qui permet la création de composés d'au moins deux noms ou adjectifs et dans laquelle tous les composants perdent leur autonomie au profit du dernier par un mécanisme de base supplétive.

Comme l'étude linguistique réalisée dans le cadre du projet EDyLex le suggère (Sagot et al., 2013), dans une grande majorité de cas, le dernier composant impose sa catégorie et sa classe flexionnelle au néologisme construit. S'il y a ambiguïté, la catégorie proposée par l'étiqueteur morphosyntaxique MElt (Denis et Sagot, 2012) appliqué en parallèle est utilisée, pour ne garder que les analyses compatibles avec cette catégorie. Ces informations sont donc conservées pour proposer la catégorie et la classe flexionnelle de la nouvelle entrée lexicale. Enfin, la forme de citation est construite en conservant tous les composants d'origine, sauf le dernier qui est remplacé par la forme de citation de l'entrée lexicale correspondante du Le*fff*.

L'analyse (iii) peut fournir de nombreuses hypothèses distinctes (forme de citation, catégorie, classe flexionnelle), notamment lors de la recherche par préfixe commun. Pour pallier cela, des contraintes ont été ajoutées dans ce cas. On s'est ainsi restreint aux expressions formées avec le mécanisme d'altération *–o* qui décrit correctement la composition adjectivale comme nous l'avons constaté dans l'étude de corpus réalisée par le projet EDyLex (Sagot et al., 2013). La table 4.22 donne quelques exemples de décompositions réalisées selon ces principes.

Il est à préciser que traiter de telles expressions lorsqu'elles sont amalgamées sans marqueurs de séparation (ni espace) demeure complexe, puisque l'on tombe

29. Ce préfixe doit contenir au moins la moitié du composant.
30. Ces formes relevant de l'incomplétude lexicale figure à présent dans le Le*fff*

Composé Inconnu [30]	Analyse(s) en composants	Cat., flexion	Forme de citaton
centre-ville	(a) centre ville (NC_mp, nc-2m)	NC_mp, nc-2m	centre ville
député-maire	(b) député + maire (NC_ms, nc-4)+ (NC_ms,nc-4sse)	NC_ms,nc-4sse	député-maire
lumino-technique	(c) lumino(lumineux)+ technique (ADJ_s,adj-ique2)+ (NC_fs,nc-2f)	NC_fs,nc-2f	lumino-technique

Table 4.22 – Analyse de composés

dans le cas difficile des mots composés standards (Sag et al., 2002). L'étude sur corpus menée par le projet EDyLex a toutefois montré que ce phénomène était marginal.

Ce module prend en entrée et rend en sortie un texte au format SxPipe. Ainsi si l'on rencontre, par exemple, l'inconnu *ésotérico-religieux*, ce module l'annotera comme tel avec l'étiquette ◁NEO suivie de toutes les informations morphosyntaxiques (lemme, catégorie grammaticale et flexion) pouvant lui être rattachées entre des balises [| et |]. Chaque analyse morphosyntaxique possible est ici espacée des autres avec un point virgule. La sortie pour cet inconnu sera donc de la forme suivante :
{ésotérico-religieux$_1$◁NEO[|Assoc(pref):ésotérico-religieux:adj-x3:ADJ_m;Assoc(pref): ésotérico-religieux:adj-x3:ADJ_m;Assoc(pref):ésotérico-religieux:nc-x3:NC_m; Assoc(pref):ésotérico-religieux:nc-x3:NC_m|]} ésotérico-religieux

4.5.1.3 Détection des néologismes dérivationnels

4.5.1.3.1 Analyse des dérivés par analogie Les néologismes dérivationnels sont construits à l'aide de l'application des règles morphologiques d'une langue sur des tokens déjà existants dans son lexique réel, représenté ici par notre lexique de référence. Nous faisons ici l'hypothèse qu'un dérivé partage avec sa forme initiale la même famille morphologique. On définit ici une famille morphologique comme un ensemble d'entrées lexicales, sémantiquement reliées les unes aux autres, qui ne diffèrent que par leurs affixes (ex. : *divulgable–divulgation*). Au vu des diverses langues que nous souhaitons traiter, nous nous limiterons ici à la morphologie concaténative, soit, aux préfixes et aux suffixes.

Notre module décrit ici s'inspire de travaux sur l'analogie appliquée à la morphologie. Cette notion, expliquée à la section 2.2, permet d'établir un rapport

entre deux paires d'éléments : x est à y ce que z est à t, noté $x : y :: z : t$. Pour la néologie, l'idée consiste à apprendre les règles d'affixation que partagent les formes appartenant à une même famille morphologique. De la sorte, ces règles pourraient ainsi être réutilisées en tant que règles de transformation afin de passer d'un néologisme à une entrée connue de notre lexique de référence [31]. Soit t un néologisme et x, y et z des formes du Le*fff*. On dira que t peut être rattaché à z et partager ainsi la même famille morphologique que lui si t est relié à z par la même règle transformation que x et y. Dans le cas de *divulgable* par exemple, nous pouvons déduire qu'il s'agit d'un dérivé si nous trouvons conjointement (i) *divulgation* dans le Le*fff* et (ii) une règle de substitution du suffixe *–able* en *–ation* (extraite d'entrées du Le*fff* comme *acceptable-acceptation*).

Ce type d'analyse nécessite par conséquent un apprentissage de ces règles de transformation. L'algorithme effectuant cet apprentissage a notamment permis la réalisation d'un travail annexe consistant à rattacher les entrées lexicales dérivationnellement reliées ensemble. Ce travail, détaillé dans (Baranes et Sagot, 2014), a été réalisé en parallèle de notre travail de thèse sur la normalisation de textes bruités et ne sera pas décrit dans ce document [32]. Notons toutefois qu'il a aussi inspiré notre système d'apprentissage par analogie de règles de normalisation décrit en section 5. Dans le cadre de cet apprentissage des règles de transformation permettant de rattacher un néologisme à une entrée lexicale, notre algorithme, faiblement supervisé, se déroule en trois étapes :

1. Nous apprenons tout d'abord des règles de transformation préliminaires à partir des formes fléchies du Le*fff* [33] en comparant tous les couples (forme de citation, forme fléchie — reliée ou non à la forme de citation) qui ne diffèrent que par un suffixe ou par un préfixe. Parmi ces couples, ne sont conservés que ceux qui ont une partie commune d'au moins 5 caractères [34]

31. Le Le*fff* ne comportant pas de noms propres, notre chaîne ne permet pas l'analyse de dérivés dont la base est un nom propre, tel que *zlataner* ou *sarkozysme*.

32. Baranes et Sagot (2014) proposent ainsi la description et l'évaluation d'une approche non supervisée permettant de rattacher entre elles les entrées d'un lexique appartenant à une même famille morphologique. Pour ce faire, cette dernière s'appuie sur des règles de transformation apprises automatiquement à partir des formes fléchies et des catégories grammaticales contenues dans un lexique flexionnel. Cette approche a l'avantage d'être assez générique pour pouvoir être appliquée à plusieurs langues pour peu que ces dernières aient une morphologie majoritairement concaténative. Elle a ainsi été mise en place et testée sur quatre langues : l'anglais, le français, l'espagnol et l'allemand. Le rappel est difficilement évaluable pour un tel travail mais notons que la précision obtenue est satisfaisante et que les résultats pour le français ont pu être comparés favorablement à une autre ressource (bien que cette dernière s'appuie sur des informations lexicographiques développées manuellement alors que notre approche ne requiert qu'un lexique flexionnel).

33. Nous tirons parti du fait que le Le*fff* contient toutes les flexions possibles de chacune de ses entrées.

34. Notre objectif ici étant de rattacher automatiquement des formes appartenant à la même famille morphologique, celles sélectionnées doivent partager une même base. Conserver des paires ayant une partie commune plus courte génère trop de candidats peu représentatifs des règles d'affixation de la langue traitée.

et qui ne diffèrent que par un suffixe ou par un préfixe (en sélectionnant les règles de fréquence ≥ 40 [35]). Ces règles préliminaires, préfixales ou suffixales, sont ensuite extraites de ces paires de formes. Elles ne couvrent pas les cas où préfixe et suffixe changent tous deux simultanément. À la fin de cette première étape, nous obtenons ainsi deux jeux de règles, un jeu de préfixations et un de suffixations, relativement bruités. Chaque règle est ici conservée avec son nombre d'occurrences et les contextes (entre crochets) dans lesquels elle s'applique, ainsi que le montre la table 4.23.

Type de règle	Règle	Nb d'occ
Préfixale	(re→_)[ndmvsjlcphbgqft]	14329
Préfixale	(en→dé)[nrdvjlchgft]	2123
Suffixale	[wradyukhgftienémlcpbo](s→_)	2123
Suffixale	[rxdyuhftinémvslpb](ent→aient)	7229
Suffixale	[nrvmlpgt](iste→e)	236

Table 4.23 – Exemples de règles préfixales et suffixales apprises

2. Les règles extraites dans cette première étape sont ensuite utilisées pour grouper les entrées du L*efff* (supposées partager une même base lexicale) en s'autorisant à appliquer une opération préfixale et suffixale en parallèle. Cela nous permet de constituer des paires de formes accompagnées de règles de transformation pour passer de l'une à l'autre. Nous ne conservons toutefois que les paires qui relient une forme fléchie à une forme de citation. Une forme fléchie peut ainsi être rattachée à sa propre forme de citation (ex : *danse, danser*) ou encore à une forme de citation de la même famille morphologique qu'elle (ex. : *danse, danseur*). À ce stade, si la règle de transformation extraite est non pas une règle dérivationnelle mais une règle flexionnelle, il est facile de la détecter : les deux formes concernées partageant alors le même lemme (ex. : *chanta/chanter*). La table 4.24 illustre ainsi la liste de paire de formes extraites accompagnée des règles qui ont permis leur construction [36].

3. Nous avons à ce stade la liste des paires de formes x, y (forme de fléchie, forme citation reliées soit par flexion soit par dérivation), qui vont nous servir à construire des relations analogiques de la forme $x : y :: t : z$, impliquant de la sorte un inconnu t. Pour ce faire, nous remplaçons chaque paire de formes par de une nouvelle règle de transformation reliant un couple préfixe/suffixe d'input à un couple préfixe/suffixe d'output. Cela nous permet de traiter les préfixations, les suffixations, et les dérivés parasynthétiques (cf. table 4.25) et d'extraire ainsi des règles dotées des informations suivantes :

35. Nous avons limité ce seuil à 40 afin de restreindre cet apprentissage aux règles régulières de la langue.

36. Ces règles ne sont accompagnées d'informations contextuelles que lorsque ces dernières semblent pertinentes.

Paire de tokens	Règles préf.	Règles suff.
édition-réédition	_→ré	—
troussage-détroussage	_→dé	—
anticipions-anticipation	—	ions→ation
travailleront-travailler	—	eront→er
avouée-inavouable	_→in	ée→able
équilibres-rééquilibrer	_→ré	[e](s→r)

Table 4.24 – Exemples des paires extraites et de leurs règles de transformation préliminaires

- Chaque côté de ces règles est assorti d'une catégorie grammaticale et de traits flexionnels (genre, nombre,...) [37].
- Ces règles indiquent le contexte dans lequel elles s'appliquent : si une substitution de suffixe, par exemple, n'a lieu que devant un *e* ou un *a*, nous conservons l'information afin de ne réappliquer cette règle que dans ce type de contextes.
- L'étude de l'ensemble de ces couples donnant parfois plusieurs fois naissance à une même règle, nous conservons le nombre d'occurrences de chaque règle extraite.
- Enfin, nous pouvons typer chaque règle obtenue comme étant plutôt flexionnelle (F) si la plupart des couples sont formés de deux formes fléchies du même lemme, ou plutôt dérivationnelles (D) dans le cas contraire.

Nous obtenons ainsi 1 433 858 règles distinctes (ayant au moins deux occurrences) parmi lesquelles nous ne conservons que les règles de plus de 80 occurrences afin de minimiser le bruit. Cela nous donne une base de 32 508 règles de transformation dont quelques exemples sont montrés en table 4.25.

Cat.	Préfixe	Suffixe	Occ	Type	Exemple
adj_Kfp → v_W	—	ées → er	6483	D	*données → donner*
v_I12s → nc_fs	_→ dé	is → tion	116	D	*valorisais → dévalorisation*
v_P2p → v_W	—	z → r	7074	F	*dansez → danser*

Table 4.25 – Exemples de règles de transformation apprises

Nous sommes ainsi en mesure de déterminer si un inconnu, analysable par ces règles, est une création lexicale. Pour cela, nous tentons tout d'abord de le relier à une entrée du Le*fff* grâce à une règle dérivationnelle [38]. Si nous y parvenons, son

37. Outre le fait que l'on extrait ici des généralisations applicables à de nouveaux tokens, ces dernières peuvent permettre de préparer une ébauche de la proposition d'entrées lexicales candidates (forme de citation + catégorie) pour une tâche d'extension dynamique du lexique.

38. Nous reviendrons sur l'intérêt et l'utilité des règles flexionnelles en section 4.5.3

Dérivé	Règles appliquée	Famille morpho	Cat., flexion	Forme de cit.
apolitisons	-sons→-que, -ons→-me,	*apolitique*, *apolitisme,...*,	V_P1p,v-er ; V_Y1p,v-er	apolitiser
divulgables	-(a)bles→-(a)tion	*divulgation*	ADJ_p, adj-2	divulgable
pathologisant	-ant→-te, -sant→-que,	*pathologiste*, *pathologique,...*,	V_G, v-er	pathologiser
décrocheurs	-eurs→-er, -eurs→-age,	*décrocher*, *décrochage,...*,	NC_mp, nc-2m	décrocheur

Table 4.26 – Analyse des dérivés

lemme (forme de citation + classe flexionnelle) est obtenu en appliquant un outil intégré au Le*fff*. Ce dernier permet de calculer pour un triplet (forme, catégorie, étiquette morphologique) l'ensemble des lemmes morphologiquement compatibles avec la grammaire morphologique du Le*fff* d'une part et avec le triplet d'autre part. Dans le cas où plusieurs lemmes sont proposés par cet outil, nous conservons celui qui a le suffixe le plus long parmi ceux attestés parmi les formes de citation du Le*fff* de même catégorie. Il est à noter que pour la chaîne de traitement mise en place pour le projet EDyLex, on applique en parallèle l'étiqueteur morpho-syntaxique MElt (Denis et Sagot, 2012). Cela leur permet alors de ne conserver que les lemmes dont la catégorie est la même que celle proposée par MElt pour l'inconnu (s'il y en a plusieurs, on choisit le mieux pondéré, s'il n'y en a aucun, on déclare l'inconnu inanalysable par ce module). Cette précaution supplémentaire a été ajoutée pour ce projet afin d'optimiser la précision de ce module. Précisons que cette seconde vérification ne peut être réalisée que sur des textes très peu bruités. Les textes journalistiques de l'AFP sur lesquels travaille le projet EDylex sont donc parfaitement appropriés. Dans le cadre industriel de notre travail sur la normalisation, cette vérification serait moins pertinente puisque MElt serait moins efficace face à des textes bruités.

Dans les cas de dérivés, ce module nous informe ainsi des tokens appartenant à leurs familles, des règles qui ont contribué à leurs constructions et des éléments permettant la création de leurs entrées lexicales : leurs formes de citation, leurs catégories grammaticales et leurs classes flexionnelles (cf. table 4.26). Lors de sa sortie au format SxPipe, un inconnu traité par ce module sera annoté avec l'étiquette ◁NEO suivie d'un label Ana indiquant que c'est bien ce module qui a permis sa détection. Les informations morphosyntaxiques (lemme, catégorie grammaticale et flexion) et celles détaillant les règles de transformation appliquées pour la détection du néologisme sont conservées entre des balises [| et |]. La sortie rendue par ce module pour l'inconnu *pathologisé*, par exemple, sera donc de la forme suivante :

{pathologisé$_1$◁NEO[|Ana(D/192/#pathologique#/ADJ_s///sé#/que#),Ana(D/153/

#pathologiste#/NC_s///é#/te#):::V_Kms|]+NEO[|Ana(D/172/#pathologique#/ADJ_s///sé#/que#):::ADJ_ms|]} pathologisé

4.5.1.3.2 Détection à l'aide de motifs dédiés Suite à l'étude de corpus menée dans le projet EDyLex Sagot et al. (2013) et la mise en place du module de détection par analogie de néologismes dérivationnels, nous avons isolé certains phénomènes standards d'affixation qui demeurent pas ou peu gérés par nos systèmes [39] (un néologisme comme *cybermonde* n'est par exemple pas géré). Ainsi, nous avons mis en place un module qui, pour un inconnu donné, cherche un préfixe standard [40] et vérifie si la chaîne de caractères suivant ce préfixe (concaténé, éventuellement par trait d'union) est un mot connu du Le*fff*. Si tel est le cas, une analyse est proposée qui construit le lemme complet à partir de l'entrée trouvée. Un mécanisme similaire est implémenté pour les suffixes *iste*, *isme* et *isation*.

Ainsi un inconnu tel que *surendettement* sera annoté par ce module avec l'étiquette ◁NEO suivie d'un label Affixe indiquant que c'est ce module qui a permis sa détection et d'informations morphosyntaxiques (lemme, catégorie grammaticale et flexion) lui étant rattachées entre des balises [| et |]. La sortie rendue par ce module pour cet inconnu sera donc de la forme suivante : {surendettement$_1$◁NEO[|Affixe(Prefix_sur):surendettement:nc-2m:NC_ms|]} surendettement

Le néologisme dérivationnel *poufisation* sera quant à lui analysé de la sorte : {poufisation$_1$◁NEO[|Affixe(Suffix_isation):poufisation:nc-2m:NC_ms|]} poufisation

4.5.2 Évaluation

Nous avons évalué ces systèmes en deux temps. Les données que nous voulons traiter étant principalement dotées de néologismes dérivationnels, le système de détection des dérivés par analogie est particulièrement important dans notre cadre. Pour cette raison, nous avons tout d'abord évalué les performances de ce système de détection des dérivés sur nos données. L'intégralité de notre chaîne de traitement, réalisée en collaboration avec l'équipe chargée du projet EDyLex, a ensuite été testée dans son intégralité sur un corpus AFP. Nous commencerons ainsi par décrire brièvement les différentes données qui ont permis ces deux évaluations en section 4.5.2.1 puis nous détaillerons les résultats que nous avons obtenus en section 4.5.2.2.

39. Ces préfixes et/ou suffixes n'ont généralement pas été retenus par le système de détection de néologismes par analogie décrit à la section précédente en raison de leur fréquence trop faible dans nos données d'apprentissage.

40. *agri*, *anti*, *après*, *archi*, *contre*, *cyber*, *dé*, *demi*, *dés*, *e*, *ex*, *extra*, *grand*, *hyper*, *im*, *in*, *inter*, *intra*, *mal*, *maxi*, *méga*, *méta*, *mi*, *mini*, *multi*, *non*, *outre*, *para*, *péri*, *pluri*, *poly*, *post*, *pré*, *quart*, *quasi*, *re*, *ré*, *sans*, *semi*, *sous*, *sub*, *super*, *supra*, *sur*, *télé*, *tiers*, *trans*, *ultra*, *uni*, *vice*, *co*.

4.5.2.1 Données d'évaluation

4.5.2.1.1 Données viavoo Nous avons mis en place un premier jeu de données assez représentatif des inconnus que nous serons amenée à traiter dans les corpus viavoo. Pour ce faire, nous avons manuellement extrait de ces corpus les principaux types d'inconnus auxquels notre système sera confronté. Nous avons ainsi mis en place trois listes d'inconnus : une première composée des 85 premiers néologismes rencontrés, une seconde composée de 85 premières altérations rencontrées et une dernière composée de 86 premiers emprunts (adaptés ou non) rencontrés. Un extrait de ces trois corpus est illustré dans la table 4.27.

Néologismes	*agenceurs, aubryste, blablater, bloggueuse, boguer, bootable, bravitude, buguer, choucrouterie, cliquables, décodifiez...*
Altérations	*abitacle, abonment, adhére, agréabl, aimmerais, alor, alore, appuier, avaiet, baculer, bazards, bnéficier, collectione,...*
Emprunts	*advantage, adventure, after, apple, artist, author, battles, believe, boost, cheek, crazy, default, delete, desperate, error,...*

TABLE 4.27 – Échantillon des inconnus présents dans les données d'évaluation

4.5.2.1.2 Données AFP Ce travail ayant été réalisé en collaboration avec l'équipe du projet EDyLex, nous avons fait une première évaluation des modules proposés pour l'analyse des néologismes sur un corpus de dépêches AFP (francophones). Ce corpus est constitué de 200 dépêches de l'Agence France-Presse (73 353 tokens) tirées au hasard entre le 1er et le 14 janvier 2013. Il contient, en tout, 729 inconnus, dont 489 inconnus distincts [41]. Ce corpus a notamment été mis en place afin d'évaluer la chaîne complète utilisée dans la chaîne de traitement réalisée pour le projet EDylex.

4.5.2.2 Résultats

4.5.2.2.1 Évaluation de la détection des dérivés par analogie Nous avons tout d'abord calculé la capacité de ce module à détecter les néologismes dans nos données viavoo tout en proposant pour ces derniers une bonne description morphosyntaxique. Pour ce faire, nous avons observé, pour chaque entrée suggérée, l'analyse proposée par notre système qui a été la mieux pondérée. Puis, nous avons aussi testé notre système sur des inconnus ne correspondant pas à des néologismes afin de vérifier que ce dernier ne tente pas de les analyser, en produisant ainsi des surdétections. Les résultats que nous obtenons sont ainsi fournis dans la table 4.28.

41. Certaines classes d'inconnus ont été supprimées de cette étude (mots commençant par des chiffres ou des majuscules) afin de se focaliser sur les créations lexicales.

Tâche	précision	rappel	F-mesure
Détection des néologismes seuls	100%	58%	71%
Détection des néologismes (corpus viavoo)	76%	58%	65%
Non analyse des emprunts	86%	86%	86%
Non analyse des altérations	92%	92%	92%
Non analyse des inc. non néo (corpus viavoo)	83%	89%	86%

TABLE 4.28 – Évaluation du module de détection des dérivés par analogie

Ainsi pour chaque néologisme analysé, nous pouvons constater que la proposition ayant la meilleure pondération par notre système est toujours valide. De manière plus générale, si nous observons toutes les autres propositions faites par notre système, on constate qu'il ne propose une analyse morphosyntaxique erronée que pour 4 néologismes.

Le rappel obtenu pour les néologismes seuls montre par ailleurs que plusieurs cas demeurent non détectés. Cela peut s'expliquer par plusieurs raisons :

- la complexité morphologique de certains néologismes (*desintrouvabiliser*, *islamophile*)
- l'absence dans notre lexique de référence du token initial qui a été dérivé morphologiquement. Cette incomplétude concerne principalement les entités nommées (ex. : *sarkozisme, sarkoziens*), les emprunts (ex. : *bootable, reseter, scroller, upgrader*) et les tokens que l'on pourrait qualifier d'*argotiques* (ex. : *keumette ou pogotais*). En effet, nous n'utilisons pas de lexiques spécifiques pour traiter ces cas pour l'instant. Ce dernier point peut ainsi facilement être amélioré en passant plus de ressources linguistiques à notre système.

On constate par ailleurs que, parmi les altérations, seules 7 d'entre elles se voient analysées dérivationnellement par notre système ce qui correspond à un taux d'erreur de 8%. En ce qui concerne les emprunts, 12 sont analysés comme des analogies. Dans ce dernier cas, notre taux d'erreur est donc de 14%

Il est enfin à noter que, parmi les néologismes non détectés, un tiers d'entre eux sont reconnus par les autres modules appartenant à la chaîne de traitement de détection des néologismes. En effet, 6 d'entre eux, correspondant en grande partie à des emprunts adaptés (ex. : *rebooter* ou *surbooker*), sont détectés par le module d'atténuation de l'incomplétude lexicale et 4 d'entre eux le sont à l'aide du module reposant sur des motifs dédiés (ex. : *surréservation* ou *ultrafacile*).

4.5.2.2.2 Évaluation de la chaîne complète Les modules présentés ici ont été mis en place afin d'être intégrés à la suite de la chaîne de traitement SxPipe que nous avons adaptée. Ainsi que nous l'avons déjà expliqué, ce travail sur les néologismes veut répondre à un double objectif : savoir détecter, annoter, un

néologisme pour répondre à nos besoins et obtenir les informations nécessaires à l'ajout de ce dernier dans un lexique (sa forme de citation, sa catégorie et sa classe flexionnelle) afin de répondre aux besoins du projet EDyLex. C'est dans le cadre de cette collaboration que nous avons évalué de manière plus complète cette chaîne. Cette évaluation tend ainsi à décrire les résultats que nous obtenons dans le cas où, en plus de pouvoir détecter des néologismes, nous aimerions les intégrer automatiquement à des lexiques. Pour chaque occurrence d'inconnu, nous avons ainsi tenté de répondre aux questions suivantes : a-t-elle été correctement identifiée comme étant ou n'étant pas un néologisme ? Son entrée lexicale (y compris) proposée est-elle correcte ? Si c'est le cas, les informations constructionnelles qui lui sont associées, sont-elles correctes ?

C'est toujours dans le cadre de cette tâche d'enrichissement lexical portée par le projet EDyLex, que nous avons fait le choix d'évaluer notre système sur les ressources AFP introduites ci-dessus. Dans cette chaîne de traitement l'ordre de ces modules importe : le premier qui parvient à analyser un inconnu interrompra le processus d'analyse pour cet inconnu. Nous appliquons ainsi ces ressources dans l'ordre suivant : nous commençons par recherche un inconnu dans les lexiques externes, puis nous vérifions qu'il ne s'agit pas d'un néologisme dérivationnel avant de l'analyser dans le module de détection des néologismes compositionnels.

Le corpus d'évaluation AFP décrit ci-dessus contient 489 inconnus distincts. Parmi ces derniers, 449 (soit 92%) ont été correctement classés, dont 357 qui ne sont pas des néologismes et 92 néologismes. Les 40 inconnus restants (8% du total) ont été mal classé, dont 34 néologismes : seulement 6 inconnus ont été analysés à tort comme des néologismes (et ont donc donné lieu à des entrées lexicales candidates erronées). Nous obtenons ainsi les résultats figurant dans la table 4.29 pour la tâche de détection des néologismes et de non-analyse des autres types d'inconnus.

Tâche	précision	rappel	F-mesure
Détection des néologismes	94%	73%	82%
Non traitement des inconnus non néologiques	91%	98%	94%

TABLE 4.29 – Évaluation du module de détection des dérivés par analogie

Dans le cadre du projet EDyLex, les 98 inconnus détectés comme néologismes, y compris les 6 classés par erreur, ont conduit à la création de 93 entrées lexicales candidates, c'est-à-dire d'entrées (forme de citation, catégorie, classe flexionnelle) qui permettent de construire automatiquement toutes les formes fléchies correspondantes. Cette liste a été évaluée manuellement avec les résultats suivants : 73 propositions sur 93, soit environ 80%, sont totalement correctes, 5 ont la bonne catégorie, mais pas la bonne classe flexionnelle (ainsi *point-presse*, considéré comme féminin et prenant un *s* au pluriel), 13 n'ont pas la bonne catégorie (mais souvent les bonnes formes fléchies, car il s'agit fréquemment de confusions nom/adjectif,

par exemple *multi-facette*), 1 est douteuse et 1 est totalement erronée (le verbe *multi-voir* pour l'adjectif *multi-vues*).

Parmi les 73 entrées lexicales correctes, 52 ont été construites par l'un de nos modules d'analyse, et non au moyen de lexiques externes. Pour ces 52 entrées, nous disposons d'informations constructionnelles, de nature à permettre le calcul d'informations supplémentaires telles que la valence ou la sémantique lexicale. Ainsi, ayant correctement analysé *co-attribuer* comme issu d'une dérivation préfixale à partir du verbe *attribuer*, nous pouvons d'une part associer à *co-attribuer* les mêmes informations de valence que celles dont on peut disposer dans un lexique comme le Le*fff*, et d'autre part savoir que le sens de *co-attribuer* peut se construire compositionnellement à partir de celui du préfixe *co-* et de celui d'*attribuer*[42]. Nous avons étudié manuellement les informations constructionnelles obtenues au cours du processus d'analyse. Pour cette évaluation, celles-ci se sont toujours avérées correctes. Par exemple, le module d'analyse des dérivés par analogie a correctement relié le verbe néologique *galvaniser* au nom *galvanisation*. Le module d'analyse des composés a su analyser *politico-judiciaire* comme formé par la composition des adjectifs *politique* (avec altération) et *judiciaire*.

Enfin, soulignons que ces modules ne peuvent pas tous être adaptés à d'autres langues. En effet la détection des néologismes compositionnels et celle des néologismes dérivationnels à l'aide de motifs dédiés ont été réalisées pour le français suite à une étude de corpus. Ces deux modules ne peuvent ainsi pas être réappliqués à d'autres langues en l'état. La détection des néologismes dérivationnels appris par analogie peut, quant à elle, être utilisée sur des langues dont la morphologie est concaténative[43]. Il suffirait pour cela de réapprendre nos règles dérivationnelles sur la langue en question. Enfin, le module proposant de prendre en compte des lexiques externes peut lui aussi être adapté à toutes les langues aisément.

4.5.3 Normalisation des néologismes flexionnels

Comme évoqué dans la présentation du module de détection par analogie des néologismes dérivés (cf. section 4.5.1.3.1), on peut distinguer parmi les règles de transformation celles qui sont dérivationnelles de celles qui sont flexionnelles. Or, si un néologisme peut être créé par une opération de composition ou dérivation, il est rarement construit par une opération de flexion[44]. Les formes fléchies corres-

42. Même si ce dernier point est plus délicat, une des caractéristiques de la morphologie constructionnelle étant précisément le caractère non complètement prédictible de la sémantique résultante.

43. Notons qu'en fonction des langues, certaines adaptations au niveau de l'apprentissage des règles seront peut-être à réaliser. En effet, par exemple nous n'aurons pas intérêt à prendre en compte la casse des noms en allemand.

44. On peut toutefois évoquer l'apparition tardive de certaines formes fléchies. C'est par exemple le cas de nombreux noms féminisés qui sont arrivés dans la langue.

pondant à une forme existante sont en grande majorité déjà connues et présentes dans nos lexiques. Ainsi, si un inconnu semble être flexionnellement rattaché à un token existant et que cette analyse est bien scorée [45], il y a de fortes chances pour que ce dernier soit mal orthographié (par exemple, *soupirails* est rattaché flexionnellement à *soupirail*). Cette information délivrée par nos règles de transformation est donc pertinente. Elle nous permet en effet à la fois de déduire la nature de cet inconnu, en effet cette dernière a plus de chance de correspondre à une altération qu'à un néologisme, mais en plus, elle nous procure des informations afin de pouvoir la normaliser correctement.

Lorsque l'on est confronté à un inconnu analysé par une règle de transformation flexionnelle, on procède comme suit. Nous commençons donc par récupérer les informations contenues dans cette règle, à savoir la prédiction proposée par notre module de la catégorie morphosyntaxique de l'inconnu en question et la forme de citation de la forme à laquelle il est rattaché flexionnellement. Nous vérifions ensuite l'existence de la forme fléchie de cette forme de citation. Si cette dernière existe, nous considérons qu'il s'agit bien d'une forme altérée et nous la normalisons.

Par exemple, considérons l'inconnu *travails*. Notre système par analogie nous proposera de rattacher cette forme de manière flexionnelle à la forme de citation *travail*. Par ailleurs, en plus de nous dire que cet inconnu est une forme fléchie de cette forme de citation, il nous précise que *travails* correspond à un nom masculin pluriel. Afin de vérifier cette analyse nous cherchons donc dans notre lexique de référence le masculin pluriel du nom *travail* (à savoir *travaux*). Puis, cette forme fléchie existant déjà, nous la conservons comme candidat de normalisation pour notre inconnu. La table 4.30 illustre le type de normalisation pouvant être obtenue avec ce module.

Altération	Sortie au format SxPipe
travails	{travails$_1$◁NEOFLEX_travail_N_mp} travaux
lettones	{lettones$_1$◁NEOFLEX_letton_ADJ_fp} lettonnes

Table 4.30 – Exemples de normalisation de néologismes flexionnels

Dans notre corpus d'évaluation, ce type d'altération n'apparaît qu'une seule fois. Une évaluation de ce module ne peut donc pas être proprement réalisée. Nous pouvons toutefois noter que ce module est peu bruyant et qu'il ne concerne qu'une infime proportion des altérations contenues dans un texte bruité.

45. Le score d'une règle correspond à son nombre d'occurrences. Une règle de transformation flexionnelle doit être très fréquente pour être prise en compte (sans quoi elle risque d'être bruyante). Nous avons donc fixé, arbitrairement, son score minimum à 1000. En dessous de ce seuil, elle est considérée comme non fiable.

4.6 Conclusion

La chaîne de traitement présentée dans cette section a pour objectif de réduire la liste des inconnus susceptibles d'être altérés et devant être normalisés par la suite. Elle se compose d'un ensemble de prétraitements permettant la détection et la normalisation des altérations simples à détecter, puis propose une classification pas-à-pas des différents inconnus pouvant être considérés comme une altération potentielle que l'on tentera de normaliser par la suite. Ainsi, nous commençons par annoter les inconnus provenant du non-lexique, puis ceux provenant du xénolexique pour enfin détecter les différents inconnus provenant de l'ensemble du lexique potentiel. À ce stade, nous obtenons ainsi une chaîne de traitement composée des blocs des modules figurant dans la figure 4.3.

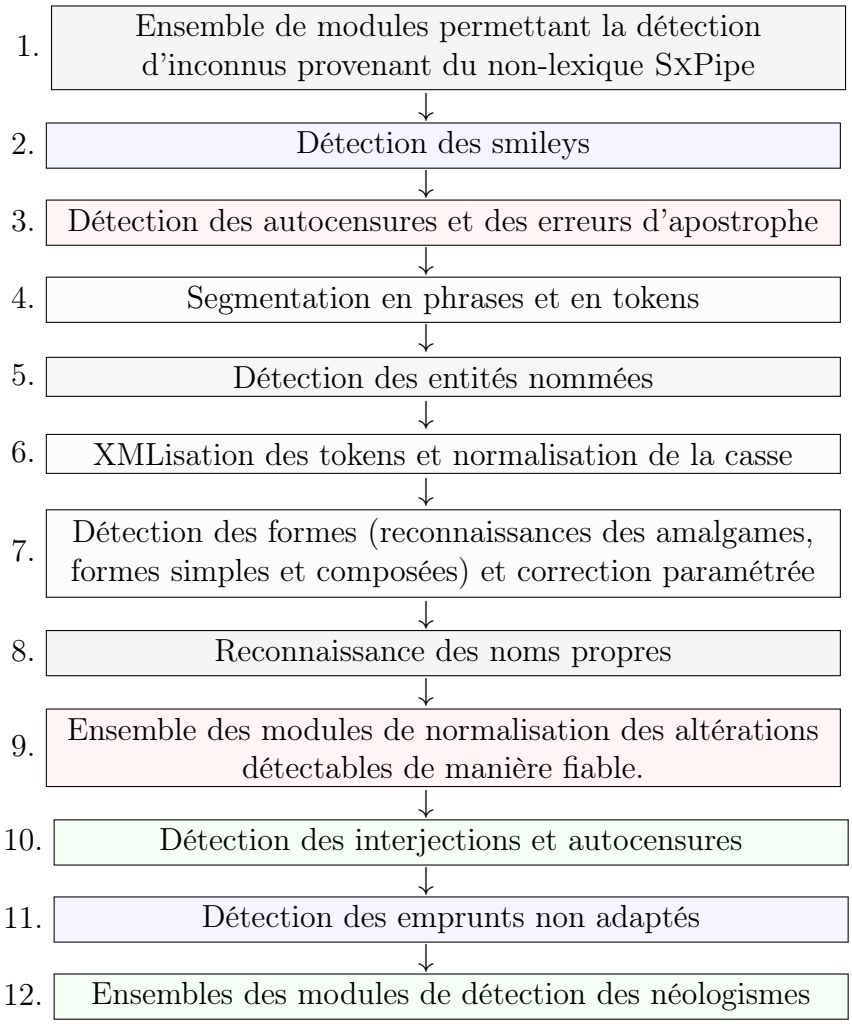

FIGURE 4.3 – Chaîne SxPipe adaptée

Si nous reprenons l'étude de corpus que nous avions réalisée avant la mise en place de cette chaîne de prétraitement (cf. figure 3.2), nous pouvons constater qu'en

plus de protéger la majorité des inconnus non altérés, les modules introduits ici permettent le traitement de près de 15% des tokens altérés. En effet, toutes les altérations ayant été produites à l'aide de caractères non alphabétiques et toutes les altérations volontaires de type étirement et autocensure ont été traitées dans ce chapitre. Par ailleurs, en ce qui concerne les tokens altérés volontairement de type SMS (squelette consonantique, troncation et réduction irrégulière), ces dernières étant très spécifiques nous prenons le parti de les inclure dans la liste de tokens devant être traités par notre lexique de substitution. Ainsi parmi les inconnus, seules les altérations non voulues réalisées sur des caractères alphabétiques et les systèmes de réduction régulière et volontaire d'un token sont encore à traiter. Elles feront l'objet du chapitre 5. Il est toutefois à noter que l'altération d'un token ne donne pas systématiquement lieu à des inconnus. Cela peut aussi produire d'autres tokens existants dans notre lexique de référence mais qui ne sont pas rattachés au même lemme que notre forme initiale. Ce dernier cas fera en partie l'objet du chapitre 6.

CHAPITRE 5

Normalisation des tokens inconnus altérés

Sommaire

5.1 Introduction

Comme nous l'avons vu précédemment, nous définissons la normalisation comme la tâche consistant à chercher, pour chaque token altéré présent dans un texte, soit sa version normativement correcte, soit un autre token qui lui serait flexionnellement rattaché. Jusqu'à présent, nous parlions des altérations comme d'une catégorie d'inconnus. Il est à préciser qu'altérer un token peut aussi résulter en d'autres tokens existant dans nos lexiques. Cela peut être problématique s'il s'agit de tokens qui ne partagent pas le même lemme que notre token initial. Nous traiterons ces différents cas de figure (altérations produisant des tokens inconnus ou connus) en deux temps. Nous commencerons tout d'abord par nous intéresser, dans ce chapitre, aux altérations produisant des tokens absents de notre lexique de référence (fautes lexicales ou altérations volontaires). Puis nous traiterons dans le chapitre suivant (chapitre 6) les tokens dont l'altération produit un autre token de notre lexique de référence (fautes grammaticales non flexionnelles [1], par exemple *sont/font* ou *et/est*). Les altérations évoquées dans la suite de ce chapitre feront ainsi toutes références à des tokens inconnus.

Bien que la majorité des tokens que nous considérons dans ce chapitre comme altérations sont des fautes lexicales, il peut aussi s'agir de réductions volontaires (ex. : *changemt* pour *changement*). De manière générale, si nous voulons normaliser une altération, deux possibilités s'offrent à nous. Nous pouvons soit tenter de la normaliser comme le ferait un outil de correction automatique, en proposant sa forme normativement correcte (ex. : *téléphonnez* → *téléphonez*), soit la remplacer par un token qui partage le même lemme que celui attendu (ex. : *téléphonnez* → *téléphoner* ou *téléphoniez*). Ce dernier rattachement ne provoquera aucun bruit puisque nous cherchons ici à améliorer un texte pour qu'il puisse être analysé par un système ultérieur qui s'appuie non pas sur les tokens mais sur les lemmes.

Notre objectif est donc de transformer un texte bruité en remplaçant ses inconnus altérés par leur correction idéale ou par une des formes fléchies du lemme qui était attendu. Pour ce faire, nous avons besoin dans un premier temps de construire des candidats de normalisation en lien avec l'altération réalisée puis, dans un second temps, de déterminer quel est le candidat le plus pertinent.

L'étude de corpus décrite en introduction de la seconde partie (cf. 3.2) montre que près de 60% des altérations étaient *non voulues* et réalisées sur des lettres, environ 15% étaient *non voulues* et faites sur des caractères non alphabétiques, enfin 25% étaient faites *volontairement*. Les prétraitements présentés au chapitre 4 nous permettent de normaliser la majorité de ces deux dernières catégories [2], écartant ainsi les altérations spécifiques aux corpus de type SMS ou dont l'orthographe est

1. Nous voulons ici normaliser un texte et non le corriger, traiter les fautes grammaticales flexionnelles (fautes d'accord ou de flexion) serait donc inutile.

2. Les squelettes consonantiques, troncations et réductions pouvant être en grande partie gérés par le lexique de substitution sans rencontrer de problème d'ambiguïté.

trop éloignée de leurs tokens normativement correctes (ex. : *tjs* ou *pk*). Plus de 60% des altérations restantes sont des altérations formées par une seule opération. Notre étude rejoint en cela le constat qu'avait fait Damerau (1964) : la majorité des fautes sont des fautes simples. Nous considérerons donc ici qu'un token altéré est produit par une seule et unique modification en son sein. Cette modification peut concerner un ou plusieurs caractères accolés et peut être influencée par les lettres qui l'entourent. En effet, on retrouvera certaines altérations plus facilement dans certains contextes que dans d'autres. Par exemple, en français l'altération *eu→ue* est principalement réalisée lorsqu'elle précède les lettres *c, g* et *q* et qu'elle est suivie des lettres *i, l, r* et *t*.

Le travail présenté ci-dessous s'inspire en partie du système de détection des néologismes présenté en section 4.5.1.3.1. Si on peut rattacher dérivationnellement une création lexicale à un lemme connu dans notre lexique de référence, relier orthographiquement une altération à un token existant en utilisant un procédé similaire doit être possible. Nous faisons ainsi le postulat que des règles de normalisation peuvent être apprises de peu qu'on ait un bon corpus d'apprentissage. En effet, bien que notre apprentissage par analogie est non supervisé puisqu'il ne s'appuie que sur des informations morpho-lexicales, nous ne pouvons pas procéder exactement de la même manière. Cet apprentissage nécessite d'avoir accès à des informations ne figurant pas dans des ressources lexicales. L'approche proposée ici sera donc supervisée par le corpus d'entraînement qu'il utilise, à savoir un corpus de fautes annotées.

Nous proposons ici de mettre en place un système de normalisation, inspiré des techniques de détection de rapports analogiques (décrites en section 2.2), qui utilise des méthodes de correction par règles pondérées automatiquement (cf. section 3.2.2.1). Nous commencerons par détailler notre système et les différents modules qui le composent à la section 5.2 avant de l'évaluer à la section 5.3.

5.2 Système proposé

La majorité des altérations que nous voulons normaliser ici proviennent d'opérations simples. Elles sont dues à de la proximité clavier (ex. : *znalise*) ou phonétique (ex. : *analize*), à de la duplication de lettres (ex. : *anallyse*) ou encore à un mécanisme de réduction volontaire trop irrégulier pour avoir été stocké dans un lexique de substitution (ex. : *pvoir*). Comme expliqué précédemment, nous faisons ici l'hypothèse que la majorité des procédés d'altération correspondent à l'application d'au plus une opération lourde [3] qui peut éventuellement avoir été associée

3. Nous appelons opération lourde les opérations introduites par Damerau (1964) et Levenshtein (1966) c'est-à-dire l'ajout et la suppression d'une lettre, la substitution d'une lettre avec une autre et l'inversion de deux lettres accolées. Par opposition aux fautes légères qui font plutôt référence à des erreurs de casse ou de diacritique.

à des opérations légères d'accentuation. Ces différents mécanismes d'altération partagent ainsi une certaine régularité.

L'analogie peut nous permettre d'apprendre cette régularité. Par exemple, si nous avons déjà vu au préalable et extrait à l'aide d'une généralisation appropriée la règle de correction permettant de passer de *engagemt* à *engagement*, nous devrions être capable de prédire l'orthographe correcte de *changemt* (cf Figure 5.1).

	ALTÉRATION : NORMALISATION			ALTÉRATION : NORMALISATION
::	engagemt : engagement	→ ::		engagemt : engagement
	changemt : ?			changemt : changement

FIGURE 5.1 – Correction de l'altération *changemt* par analogie

Le système proposé ici ainsi a pour objectif de normaliser chaque token inconnu du Le*fff* que nous considérons comme potentiellement altéré, dans la mesure où ce dernier n'a pas été traité ou identifié comme un emprunt non adapté ou un néologisme par les systèmes décrits précédemment (cf. chapitre 4). Pour ce faire, il suggère des CANDIDATS DE NORMALISATION pondérés à l'aide de RÈGLES DE CORRECTION apprises sur un corpus d'erreurs lexicales annoté.

Dans un premier temps, nous montrerons comment nous avons extrait automatiquement ces règles de correction (section 5.2.1), puis nous expliquerons comment nous combinons ces règles entre elles (section 5.2.2). Enfin, nous verrons comment pondérer les candidats de normalisation proposés par notre système afin de ne conserver que le ou les candidats les plus probables (section 5.2.3). Ce travail permettra ainsi la création d'un nouveau module proposant plusieurs candidats de normalisation pour une altération donnée. Un système de filtrage permettant de conserver que les candidats les plus pertinents sera ensuite proposé dans le chapitre 6.

5.2.1 Apprentissage des règles de correction

L'apprentissage de nos règles de correction s'appuie sur un corpus de fautes lexicales annotées, le corpus WiCoPaCo (Max et Wisniewski, 2010) sur lequel nous reviendrons à la section 5.3.1. Nous verrons notamment comment nous en avons extrait un ensemble de couples de tokens (token altéré, token bien orthographié) correspondant à des fautes lexicales. Un tel ensemble constitue l'entrée de notre système, et nous cherchons à en extraire automatiquement des règles pondérées de correction. Comme indiqué précédemment, nous faisons l'hypothèse que chaque altération est le résultat d'au plus une erreur « lourde ».

Ainsi, pour concevoir nos règles de correction, il nous suffit d'extraire automatiquement de chaque couple de tokens les informations suivantes : le contexte gauche complet précédant une altération, l'altération et sa correction, le contexte droit complet suivant l'altération. Précisons qu'afin de prendre en compte les

débuts et fins de tokens au sein du contexte, ces derniers seront marqués par les symboles ##. Ainsi les tokens *exempel*::*exemple*, par exemple, seront ici notés *##exempel##*::*##exemple##*. L'altération ayant lieu en fin de token, son contexte droit pourra tout de même être pris en compte. Les contextes gauche et droit étant identiques dans le token altéré et dans sa forme bien orthographiée, il est simple d'extraire les séquences de lettres constituant une chaîne de caractères altérée et représentant la correction de cette dernière. Pour le couple *##souevnt##*::*##souvent##* par exemple, l'alignement se fera comme suit :

$$\begin{array}{ccccc} \#\#\text{sou} & | & \text{ev} & | & \text{nt}\#\# \\ \updownarrow & & \updownarrow & & \updownarrow \\ \#\#\text{sou} & | & \text{ve} & | & \text{nt}\#\# \end{array}$$

Cet alignement nous permet d'extraire de cette paire de tokens les informations suivantes :

- la ZONE ALTÉRÉE et sa contrepartie corrigée, c'est-à-dire la chaîne de caractères concernée par la correction avant et après sa modification. Ce sont ces deux chaînes qui sont utilisées pour passer d'un token à un autre. Dans notre exemple, il s'agira de « *ev* → *ve* » ;
- le contexte dans lequel s'applique cette substitution. Dans notre exemple, on obtiendra pour contexte gauche « *sou* » et pour contexte droit « *nt* ».

Une règle peut ensuite, à partir de ces données, être inférée comme suit : *{##sou}{ev→ve}{nt##}*. Il apparaît toutefois que cette règle est trop spécifique par le contexte qu'elle sélectionne. Elle ne s'appliquera à aucune autre altération. A contrario, une règle trop large, par exemple sans prise en compte du contexte, surcorrigerait. Afin d'éviter chacun de ces cas, nous avons construit plusieurs jeux de règles généralisant de façons différentes des règles individuelles directement extraites des altérations, ou RÈGLES DE BASE.

Lorsque l'on souhaite généraliser une règle de correction, nous pouvons procéder de trois manières. En effet, nous pouvons choisir de :

1. privilégier la précision de notre système en choisissant de construire des règles spécifiques en généralisant faiblement les règles de base,
2. avantager le rappel de notre système en proposant des règles plus larges, en généralisant fortement les règles de base,
3. favoriser clairement le rappel de notre système en proposant des règles génériques induites sans corpus d'apprentissage. Les candidats suggérés pourraient alors être pondérés en fonction de leur fréquence en corpus.

On peut donc s'interroger ici sur les éléments à prendre en compte afin de généraliser une règle. Dans ce travail, nous ne voulons pas généraliser la transformation

elle-même [4]. En effet, cela risquerait de rendre notre système de normalisation bruyant, car il serait alors trop permissif. C'est pourquoi il paraît plus fiable de se concentrer sur la généralisation du contexte dans lequel une transformation est possible. Une première question à se poser concerne alors la taille du contexte à prendre en compte. Un contexte très petit aurait tendance à généraliser la règle de correction mise en place. Un contexte trop long serait quant à lui trop restrictif. En outre, on peut noter que plus les lettres appartenant au contexte d'une altération sont éloignées des caractères altérés, moins elles seront discriminantes et donc pertinentes. Dans le travail présenté ici, nous proposons de prendre en compte un contexte de longueur constante. Ce choix a notamment l'intérêt de nous permettre de traiter chaque token de manière équivalente.

Dans ce cadre, plusieurs procédés ont été appliqués afin de généraliser un contexte. Nous proposons ainsi de permettre la combinaison des contextes gauche et droit de manière plus tolérante. Par exemple, pour la création d'une même règle de base, nous pourrions parfaitement rassembler ses contextes gauches et ses contextes droits possibles. Une autre idée serait de se concentrer sur la généralisation du contexte en fonction de sa position, de sa distance de la chaîne altérée. Nous pourrions par exemple généraliser la lettre précédant et suivant la zone altérée, afin d'autoriser plusieurs lettres ou catégories de lettres. En effet, il serait possible de regrouper les lettres ou catégories de lettres pouvant être placées à une même position.

De manière plus générale, il est très complexe de proposer une généralisation propre à l'altération produite (proximité clavier, phonétique, autre). Tout d'abord parce que nous ne pouvons pas savoir quelle altération a été produite. Réaliser une ou plusieurs généralisations du contexte adaptées à un ou plusieurs types d'altérations serait alors complexe à mettre en place et à appliquer. De plus, si nous mettons l'accent sur un ou plusieurs types d'altérations, cela se fera forcément au détriment d'autres. Nous pourrions par exemple nous concentrer sur les fautes typographiques (proximité clavier). Dans ce cas, on peut alors supposer que l'altération en question se fera la majorité du temps sur un seul caractère et son contexte, restreint à une lettre avant et après la zone altérée, et qu'elle sera généralisée en fonction du placement des touches sur un clavier. Ce procédé sera probablement efficace pour ce type d'erreur mais clairement pas pour tous les autres types d'altérations. Il en est de même pour les erreurs phonétiques qui, au contraire, n'auraient pas une taille fixe de contexte. Mettre en place un système trop précis ne serait ainsi pas concluant. Par ailleurs, généraliser le contexte en se concentrant à la fois sur les proximités clavier et phonétique serait complexe, car les informations à conserver ne seraient pas de taille régulière et donc difficilement positionnable. Par opposition, une solution possible est de généraliser les contextes en nous appuyant sur les notions de consonnes et de voyelles. Ces notions

4. Nous reviendrons toutefois sur le cas spécifique des accents et des cas de duplication de lettres.

se distinguent clairement, linguistiquement, l'une de l'autre de par leurs fonctionnements différents. C'est pourquoi nous avons opté pour cette méthode.

Avant de détailler ces deux jeux de règles, nous avons toutefois besoin de définir quelques termes. Étant donnée une zone altérée, nous appellerons CONTEXTE DE NIVEAU 1 le contexte formé des deux lettres encadrant immédiatement cette zone (la lettre à sa gauche et celle à sa droite). Nous nommerons CONTEXTE DE NIVEAU 2 les deux lettres situées à une distance de 2 de la chaîne de caractères altérée, c'est-à-dire les deux lettres encadrant immédiatement le contexte de niveau 1. Ainsi, si nous reprenons la paire de tokens *##souevnt##*::*##souvent##*, qui a pour zone altérée : « *ev* → *ve* », le contexte de niveau 1 sera *u* et *n* et celui de niveau 2 sera *o* et *n*. Plus nous nous éloignons de l'altération réalisée, moins le contexte peut avoir un impact sur elle. Prendre en compte des contextes de niveau 3 ou plus ne serait donc pas réellement pertinent. C'est pour cette raison que nous ne prendrons pas en compte de contexte supérieur dans ce travail.

Nous avons ainsi fait le choix d'apprendre à partir d'un corpus d'entraînement deux jeux de règles dotés de différents niveaux de détail. Le premier, le jeu de règles spécifiques, contiendra des règles assez précises permettant ainsi de favoriser la précision de notre système, tandis que le second, le jeu de règles larges, sera constitué de règles plus générales optimisant ainsi le rappel de notre système. La confection de ces deux premiers jeux de règles est décrite ci-dessous.

- **Règles spécifiques :** Le premier jeu de règles est produit par une généralisation limitée des règles de base. Ces règles ne conservent que le type (consonne (C), voyelle (V) ou #) des lettres de contexte de niveau 2 mais laisse inchangé le contexte de niveau 1. Ainsi, si l'on rencontre la paire *souevnt*::*souvent* lors de l'apprentissage, la règle spécifique qui sera extraite est : *{Vu}{ev→ve}{nC}*.

  Lorsqu'une nouvelle règle spécifique ne diffère d'une autre déjà extraite que par son contexte de niveau 1, nous les fusionnons : leurs contextes droits et gauches sont unifiés. Ceci constitue une généralisation par rapport aux deux règles prises isolément. Par exemple, si l'altération suivante à traiter est *##machiaevls##*::*##machiavels##*, on obtient la règle de base *{Va}{ev→ve}{lC}* : la règle de substitution extraite et le contexte de niveau 2 sont identiques à ceux de la règle précédente. Seul le contexte de niveau 1 diffère. Nous fusionnons alors les contextes de niveau 1 par disjonction, la règle obtenue étant alors notée : *{V[au]}{ev→ve}{[nl]C}*. Cette fusion constitue une généralisation puisque cette règle couvre alors également les règles de base *{Vu}{ev→ve}{lC}* et *{Va}{ev→ve}{nC}*, pourtant non extraits. En revanche, dès lors que le contexte de niveau 2 ne correspond pas, nous créons une nouvelle règle. Ainsi, l'altération *##réserev##*::*##réserve##* induira l'ajout d'une nouvelle règle à notre jeu de règles : *{Vr}{ev→ve}{##}*.

- **Règles larges** Le second jeu de règles est produit par une généralisation plus forte des règles de base : le contexte de niveau 2 est effacé, et seul le type des lettres du contexte de niveau 1 est conservé. Pour l'altération *#souevnt#*::*#souvent#*, la règle large induite sera donc : *{V}{ev→ve}{C}*.

  Dans le cas où la même règle de base réapparaît via une autre paire de tokens, le type des contextes concernés sera combiné. Par exemple, si nous avons la paire *#evuillez#*::*#veuillez#* alors notre règle précédemment extraite sera généralisée de la sorte *{[#V]}{ev→ve}{[CV]}*

Nous avons également appliqué certaines généralisations à ces deux jeux de règles, au niveau de la règle de substitution elle-même. Tout d'abord, nous avons généralisé les erreurs de duplication d'une lettre (ex. : « *fautte* ») et de suppression d'une lettre doublée (ex. : « *ereur* »), en les représentant comme suit :

(2) *fautte*::*faute* : *{Vt}{+_→_}{e#}*

(3) *ereur*::*erreur* : *{Vr}{_→+_}{eV}*

S'il s'agit d'une erreur de duplication de lettre ainsi que l'illustre l'exemple 2, la chaîne de caractères altérée sera réécrite dans notre règle de transformation comme la concaténation du symbole + et d'un tiret bas. Le symbole + étant à gauche de ce tiret bas, il signifiera que le contexte gauche de niveau 1 a été dupliqué. Si à présent, la chaîne de caractères *+_* correspond à la chaîne de caractères normalisée, dans ce cas cela signifiera que le contexte gauche de niveau 1 doit être dupliqué comme le montre l'exemple 3. Ainsi, en fonction de si la chaîne « *+_* » est à gauche ou à droite de la flèche, on couvre ainsi respectivement les cas de duplication manquante et de duplication erronée. Ce procédé nous permet de nous concentrer non pas sur la lettre impactée mais sur l'opération elle-même.

Enfin, nous avons généralisé les fautes d'accents et de cédille. Pour ce faire, nous représentons une lettre accentuée par la combinaison de deux caractères : l'accent en question suivi de la lettre concernée. Ainsi, nous noterons le token *arrêt* comme ceci : « *arr^et* » et, pour l'altération *arret*, nous obtiendrons la règle large suivante : « *{C}{_→^}{V}* », qui couvre le rajout d'un accent circonflexe sur toute voyelle, pour peu que les contextes correspondent.

Ainsi, pour construire ces deux jeux de règles, nous parcourons chaque couple (token mal orthographié, token corrigé) du corpus annoté et nous extrayons les règles spécifiques et larges comme décrit ci-dessus. Toutes ces règles sont pondérées comme suit. Dès qu'une règle est extraite ou modifiée par un couple, son nombre d'occurrences est incrémenté de 1. Nous prenons alors le logarithme de ce nombre d'occurrences, puis nous le normalisons par transformation affine entre 0 (règle au nombre d'occurrences minimal) et 1 (règle au nombre d'occurrences maximal). Le résultat constitue le poids de la règle. La table 5.1 illustre quelques exemples de règles de correction apprises par notre système.

Type	Règle	Poids	Exemple
Spécif.	{V[pfnlmctbsredg]}{_→+_}{[aieuonyr]C}	0,970	***aten****dre*→***atten****dre*
	{V[rmltdvcsnpyxg]}{a → e}{[nmilu]C}	0,660	*r**alant**ir*→*r**alent**ir*
Large	{C}{io → oi}{C}	0,298	***tiol****e* → ***toil****e*
	{´ → ` }{V}	0,738	*él**é**ve* → *él**è**ve*

Table 5.1 – Exemples de règles de correction extraites

5.2.2 Génération de règles de correction génériques

Ainsi qu'évoqué plus haut, un jeu de règle très générique, mis en place automatiquement, peut favoriser le rappel d'un système. Bien que les deux jeux de règles présentés ci-dessus soient assez complets, il peut arriver qu'une simple faute typographique, trop peu fréquente ou réalisée dans un contexte inattendu ne puisse pas être normalisée. Par fautes typographiques nous entendons ici les altérations dues à l'ajout et à la suppression d'une lettre (ex. : *boneur*, *bonheure*), à la substitution d'une lettre par une autre (ex. : *bomheur*) et à l'inversion de deux lettres ensemble (ex. : *bohneur*). Ces quatre opérations sont à la base de la notion de distance d'édition (Damerau, 1964). Ces erreurs produisant souvent que peu d'ambiguïté, nous avons mis en place un jeu de correction plus générique apte à corriger ces erreurs peu coûteuses. Pour ce faire, il cherche tous les tokens du L*ffff* qui sont à une distance d'édition de 1 de l'altération concernée. Nous dénoterons ce module par le terme de *correction générique*.

Cette méthode, plus risquée que les précédentes, est susceptible de provoquer du bruit dans nos résultats. C'est pourquoi la correction générique, n'utilisant pas des règles apprises sur un corpus d'erreurs, ne corrigera que les altérations que notre système n'aura pas traitées. Chaque candidat proposé par le module de correction générique est pondéré par le logarithme de sa fréquence dans la Wikipédia française. Il est ensuite normalisé de façon affine de telle sorte qu'un inconnu de la Wikipédia, qui sera considéré comme y étant attesté 0,1 fois, obtienne un poids de 0 et que le token le plus fréquent de la Wikipédia ait un poids de 1. Ainsi les tokens les plus fréquents de la langue auront plus de chance d'être choisis comme candidat de normalisation lors de l'évaluation, dès lors que l'on prendra la fréquence en compte.

5.2.3 Application des différents jeux de règles

5.2.3.1 Génération des candidats

Notre système vise à proposer les candidats de correction les plus probables pour une altération donnée, dans le but d'obtenir une ou plusieurs normalisations va-

lides. Pour ce faire, nous appliquons successivement plusieurs procédés jusqu'à obtenir des candidats de correction.

1. Nous commençons ainsi par utiliser nos deux premiers ensembles de règles (spécifiques et larges). Chaque règle qui peut s'appliquer sur une altération tente de produire un candidat de normalisation qui n'est retenu que s'il est présent dans notre lexique de référence, le Le*fff*.
2. En l'absence de candidat, nous autorisons chaque règle à être complétée par des changements d'accentuation. Pour cela, nous vérifions pour chaque candidat obtenu par l'application d'une règle si sa contrepartie non accentuée est présente dans une version préalablement désaccentuée du Le*fff*. Si c'est le cas, nous considérons toutes les versions accentuées correspondantes comme des corrections possibles. Pour *elevve* par exemple, nous proposerons *élevé* ou *élève*[5]
3. Dans le cas où nous n'avons toujours pas de candidat, nous tentons d'en produire à l'aide de nos règles de correction génériques.

5.2.3.2 Pondération des candidats

Si nous parvenons, lors d'une de ces étapes, à produire des candidats de normalisation, nous les pondérons en nous appuyant sur nos différents jeux de règles.

Pondération des candidats obtenus par les règles spécifiques et larges Nos deux premiers jeux de règles proposent chacun des candidats accompagnés de leurs scores comme illustré dans la table 5.2. Il est à noter que si une altération est reconnue par notre jeu de règles spécifiques, elle le sera automatiquement aussi par notre jeu de règles large[6]. Toutefois, les règles spécifiques, de par leur nature, proposeront moins de candidats que les règles larges. Par ailleurs, les scores proposés par ces deux jeux pour une même normalisation ne seront jamais identiques. Enfin, il est à noter que si un candidat résulte d'une correction spécifique ou large combinée avec une correction légère d'accentuation, son score de correction sera celui de la correction par règle. La faute d'accentuation étant considérée comme peu coûteuse, elle ne s'inscrit pas dans la pondération d'un candidat. Afin d'attribuer un score global à un candidat, nous prenons en compte les analyses obtenues par nos deux jeux de règles.

Nous avons par ailleurs constaté que certaines propositions de normalisation correspondent à des tokens assez rares dans la langue et sont, malgré tout, trop bien

5. Dans le cas où un accent était déjà présent dans le token altéré, on ne garde que les candidats contenant le nombre d'accents minimum à ajouter tout en conservant celui déjà présent. Par exemple, pour le token *elevvé*, seul *élevé* sera proposé, car *élève* supposerait 3 modifications d'accentuation et supprimerait le seul accent qui était initialement présent.

6. L'inverse n'est pas vrai pour autant.

Token altérée	Candidats proposés	Poids des règles spécifiques	Poids des règles larges
onférence	*conférence*	0.459	0.754
	inférence	0.236	0.543
fitrage	*titrage*	_	0.188
	filtrage	_	0.781
	vitrage	0.001	0.094
sufisemment	*suffisamment*	0.000	0.000
arquéologues	*archéologues*	0.101	0.149
innacompli	*inaccompli*	0.351	0.094

Table 5.2 – Exemples de candidats (cas de base : sans changement d'accent, règles génériques non déclenchées)

pondérées. Pour pallier cela, nous avons choisi de prendre en compte la fréquence de chaque candidat dans la langue. Pour cela, nous procédons de la même manière que pour la correction générique. Nous extrayons ainsi de la Wikipédia des scores de fréquence pour chaque token. Ces scores, normalisés de façon affine, sont ainsi compris entre 0 et 1.

Pour un même candidat de normalisation c du token d'origine w nous pouvons donc avoir jusqu'à trois scores :

- le score $S_s(c, w)$ égal au poids de la règle spécifique qui permet de passer de w à c, s'il y en a une,
- le score $S_l(c, w)$ égal au poids de la règle large qui permet de passer de w à c,
- la score de fréquence $F(c)$ de la correction proposée c.

Pour évaluer, pondérer et trier chaque candidat, nous voulons prendre en compte ces trois scores. Pour ce faire une solution est de les combiner linéairement afin d'obtenir un score global. Soit λ_s le coefficient assigné au score de la correction spécifique et λ_l celui assigné au score de la correction large. Nous définissons le score global d'une correction par $S_{\lambda_s,\lambda_l}(c, w) = \lambda_s S_s(c, w) + \lambda_l S_l(c, w) + (1 - \lambda_s - \lambda_l)F(c)$, où λ_s et λ_l sont compris entre 0 et 1 et $\lambda_s + \lambda_l \leq 1$.

Précisons qu'il peut arriver qu'un candidat soit dépourvu de score provenant de la correction spécifique. On peut alors supposer que ce cas n'est pas comparable au cas où le poids de la correction large serait combiné à celui de la correction spécifique qui aboutit au même candidat. Nous faisons toutefois l'hypothèse que traiter ces deux cas de la même manière (soit en utilisant les mêmes valeurs de λ_l) ne sera pas pour autant discriminant. La vérification de cette hypothèse est par ailleurs réalisée en section 5.3.2.2.2.

C'est ainsi à partir de ce score global que nous définissons le meilleur ou les n meilleurs candidats pour une altération donnée. Naturellement, ce score dépend des valeurs choisies pour λ_s et λ_l. Nous avons donc fait varier ces deux paramètres lors de l'évaluation de notre système de pondération et du choix des candidats parmi l'ensemble des candidats proposés pour chaque altération. Au préalable, nous avons évalué la pertinence des candidats eux-mêmes.

Pondération des candidats obtenus par les règles génériques Dans le cas où c'est le jeu de règle générique qui a produit des candidats, le seul paramètre dont nous disposons est la fréquence du candidat proposé comme illustré dans la table 5.3. Nous conserverons donc uniquement ce score.

Altération	Candidat de Normalisation	Score Freq	Type de règle
coéhrence	cohérence	0,780	inversion
ocrivain	écrivain	0,642	substitution
fromageç	fromage	0,381	suppression

TABLE 5.3 – Évaluation des candidats après sélection par l'oracle pour une tâche de correction

5.3 Évaluation

5.3.1 Données utilisées

5.3.1.1 Corpus WiCoPaCo

Afin d'entraîner et d'évaluer notre système, nous avons utilisé le corpus d'erreurs WiCoPaCo Max et Wisniewski (2010). Ce corpus, créé à partir des révisions des pages de la Wikipédia francophone, est composé de 72 483 erreurs lexicales et de 74 100 erreurs grammaticales qui ont été annotées, automatiquement, comme telles dans le corpus par leurs auteurs. Pour ce faire, ils ont extrait parmi les tokens qui avaient subi une modification au fur et à mesure des révisions, ceux qui correspondaient à une correction [7]. Si le token corrigé était auparavant inconnu, il est considéré comme une erreur lexicale (*non-word error*) sinon comme une erreur grammaticale (*real-word error*). Chacune des fautes est ainsi associée à sa correction, effectuée par un contributeur de la Wikipédia. Puisque dans ce chapitre nous nous intéressons uniquement aux fautes lexicales (et non grammaticales), nous n'avons utilisé que les fautes annotées *non-word error*. Par ailleurs, nous ne

7. Les corrections ont dans leur travail été distinguées des reformulations et des vandalismes à l'aide de la taille de la modification réalisée. Une « petite » modification est considérée comme la correction d'une erreur.

voulons pas que la fréquence d'un token puisse biaiser la pondération des règles qui en seront extraites. Nous n'avons donc conservé qu'une seule occurrence de chaque faute annotée (soit un total de 36 344 fautes).

5.3.1.2 Corpus d'entraînement et d'évaluation

Corpus d'entraînement Afin d'évaluer la taille nécessaire à notre corpus d'entraînement, nous avons effectué plusieurs tests en prenant 60% et 90% du corpus WiCoPaCo décrit ci-dessous. Nous avons ainsi constaté que prendre un corpus correspondant à 90% de l'ensemble des fautes lexicales n'améliorait pas nos résultats de manière significative. Pour cette raison, nous avons préféré utiliser 60% de ces dernières, soit 21 581 fautes, comme données d'entraînement pour l'apprentissage de nos deux jeux de règles [8]. Nous avons obtenu de la sorte un jeu de 4 795 règles spécifiques et un jeu de 3 073 règles larges.

Corpus d'évaluation Environ 7,5% des fautes lexicales de WiCoPaCo (soit 2 731 fautes), sans recouvrement avec les données d'entraînement, ont été conservées pour constituer notre corpus de test. Comme indiqué plus haut, notre système a pour but de s'insérer dans notre chaîne de traitement SxPipe adaptée. Dans la mesure où nous cherchons ici à évaluer notre système de normalisation indépendamment, nous avons d'une part désactivé les modules de détection des emprunts et des néologismes et d'autre part vérifié qu'il n'y avait pas de créations lexicales ou de tokens étrangers dans nos données d'évaluation, supposées ne contenir que des fautes lexicales et leur correction. En réalité, nous avons trouvé quelques cas ne relevant pas de fautes lexicales, mais plutôt de substitutions d'un token par un autre. Nous avons ainsi identifié et retiré manuellement de nos données d'évaluation 38 occurrences de créations lexicales (ex. : *herbologiste*, *windobe*, *zitanoisme*) ou à des tokens étrangers (ex. : *poesía*, *musgos*), qui ont été remplacées dans la Wikipedia par d'autres [9]. Ces cas constituaient environ 1,4% des données d'évaluation. Par ailleurs, 137 « fautes » ont été écartées parce qu'elles appartenaient déjà à notre lexique de référence, le Le*fff*, et ne constituaient pas à ce titre des candidats à la correction pour notre système (ex. : *téléscope*, *soeur*). Suite à ces retraits, nos données de test contiennent 2 556 fautes lexicales pour lesquelles

8. Nous avons par ailleurs l'intention d'étendre nos tests en travaillant sur un autre corpus d'erreurs non décrit ici, en cours de développement, d'environ 20 000 fautes. Afin que nos expériences puissent être comparables, il est préférable que ces deux corpus fassent la même taille, c'est aussi pourquoi nous n'avons conservé que 60% des fautes lexicales de WiCoPaCo pour l'apprentissage.

9. Les emprunts sont généralement remplacés par leur traduction dans la langue cible (*poesía*→*poésie*), les créations lexicales quant à elles sont souvent soit remplacées par leur token existant (*herbologiste*→*herboriste*), soit par une version plus politiquement correcte (*windobe*→*windows*) soit par un autre token sans lien sémantique avec le premier (*zitanoisme*→*taoïsme*).

nous disposons d'une correction spécifiée par les contributeurs de la Wikipédia (soit environ 7% des fautes lexicales de WiCoPaCo.

5.3.2 Résultats obtenus

Bien que nous ayons développé des techniques de correction automatique dans notre système, rappelons que notre objectif reste la normalisation. Nous avons fait ici le choix d'évaluer ce système pour des tâches de correction automatique et de normalisation. Ces deux tâches se distinguent par leur manière de valider un candidat pour une altération donnée. En normalisation, notre système peut nous proposer n'importe quelle forme fléchie du lemme *correct*. En correction automatique, on n'acceptera que sa forme normativement correcte. Il se peut, parmi les normalisations proposées pour une altération donnée, que plusieurs soient valides alors qu'aucune ne correspond pour autant à la bonne correction. Pour *dormion* par exemple, si notre système nous propose les candidats *dormir* ou *dormons*, nous considérerons que l'altération a été normalisée correctement bien que la correction attendue ne figure pas parmi les candidats.

Pour la suite de cette évaluation, nous avons utilisé les notions de précision, rappel et f-mesure introduite précédemment. Si l'on ne prend en compte qu'un seul candidat par altération, la précision P correspond au rapport entre le nombre de tokens bien corrigés (resp. normalisés) et le nombre de tokens pour lesquels nous avons proposé au moins un candidat de correction. Le rappel R sera calculé comme le rapport entre le nombre de tokens bien corrigés (resp. normalisés) et le nombre de tokens donnés en entrée. Si l'on cherche à prendre en compte plusieurs candidats, on considérera qu'une altération est bien corrigée/normalisée dès lors qu'une bonne correction/normalisation se trouve parmi lesdits candidats. La f-mesure F, quant à elle, sera calculée comme ceci : $F = 2PR/(P + R)$.

Les résultats obtenus, sur le corpus test présenté ci-dessus, ont été évalués en deux temps. Nous nous sommes tout d'abord intéressée à la qualité de la totalité des candidats proposés sans prendre en compte leur pondération (section 5.3.2.1). Puis nous nous sommes concentrée sur la pertinence de notre système de pondération (section 5.3.2.2).

5.3.2.1 Évaluation de l'ensemble des candidats

Cette première évaluation, effectuée à l'aide des données d'évaluation décrites ci-dessus, ne s'appuie sur aucun scores. Son but est principalement d'évaluer la qualité des candidats pour une tâche de correction et pour une tâche de normalisation. Elle permet ainsi de vérifier dans quelle mesure ce système est bruyant lorsqu'il propose des candidats. Pour ce faire, nous nous sommes intéressée à la quantité de candidats conçus pour chaque altération, puis à la couverture de ce

système. Nous nous sommes, par la suite, penchée sur la qualité des propositions réalisées pour des tâches de correction et de normalisation en calculant les scores optimaux que nous pouvons obtenir et en proposant une baseline pour notre système.

5.3.2.1.1 Nombre de candidats proposés Il est important de vérifier dans un premier temps que notre système ne propose pas trop de candidats, notamment au niveau du module de correction générique. Nous avons donc compté les candidats obtenus avec et sans ce dernier. Le nombre de candidats proposés reste raisonnable, même avec le module de correction générique. En effet, comme le montre la figure 5.2, seule une altération sur quatre se voit attribuer plus de 4 candidats. Le nombre de ces derniers décroît par ailleurs assez rapidement puisque même avec le module de correction générique, près de 45% des tokens altérés de notre corpus de test ne se voient attribuer qu'une seule proposition de correction.

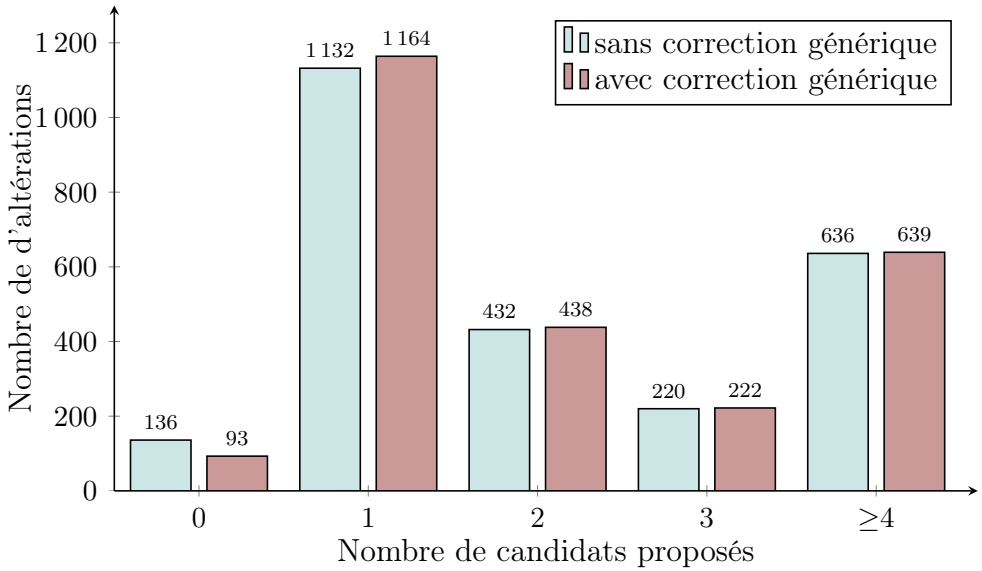

FIGURE 5.2 – Nombre de candidats proposés avec et sans correction générique

5.3.2.1.2 Couverture Cette figure nous montre aussi l'utilité de la correction générique. En se limitant aux seuls jeux de règles, 5,3% des tokens ne reçoivent aucun candidat. La correction générique permet de réduire ce taux à 3,6%. Les 93 tokens qui demeurent non traités ont été étudiés manuellement : 85% d'entre eux correspondent à des altérations trop complexes ou trop nombreuses (ex. : *arondisemernts*, *aérdorme*) et 15% à des séquences de caractères difficilement interprétables, et ce même pour un humain (ex. : *klàkoes*, *piwut*).

Nous avons cherché, par ailleurs, à estimer la qualité des candidats proposés. Pour ce faire, étant donné un inventaire de candidats pour chaque altération, nous pouvons calculer une borne inférieure et une borne supérieure pour le système dans son ensemble. Ces bornes encadrent les performances que nous pourrons obtenir lorsque nous attribuerons des valeurs aux coefficients λ_s et λ_l, c'est-à-dire lorsque nous évaluerons la qualité de notre système de pondération à la section 5.3.2.2. Ces bornes ne prennent pas en compte les scores de correction de chaque candidat, et sont définies comme suit.

5.3.2.1.3 Borne supérieure pour le système complet Afin de déterminer le score maximum que notre système pourrait atteindre étant donné les candidats obtenus, nous utilisons un oracle. Pour chaque altération, dès lors qu'une bonne correction est trouvée, l'oracle le choisit. À défaut, si au moins une bonne normalisation est proposée, l'une d'entre elles est choisie au hasard par l'oracle. Si aucune normalisation valable ne se trouve parmi les candidats obtenus, le choix de l'oracle importera peu. Un tel oracle a été évalué pour les deux tâches de correction et de normalisation. Ses résultats obtenus figurent respectivement dans les tableaux 5.4 et 5.5. Ils sont nécessairement meilleurs que ceux que nous obtiendrons avec notre système de storage, et ce quel que soit le nombre de candidats conservés en définitive.

Altérations	Précision	rappel	F-mesure
sans correction générique (2 556 tokens)	94,1	89,1	91,6
avec correction générique (2 556 tokens)	93,7	90,3	92,0
uniquement concernées par corr. générique (sur 93 tokens)	69,8	22,1	33,5

TABLE 5.4 – Évaluation des candidats après sélection par l'oracle pour la tâche de correction

Altérations	Précision	rappel	F-mesure
sans correction générique (2 556 tokens)	95,6	90,4	92,9
avec correction générique (2 556 tokens)	95,1	91,6	93,3
uniquement concernées par corr. générique (93 tokens)	69,8	22,1	33,5

TABLE 5.5 – Évaluation des candidats après sélection par l'oracle pour la tâche de normalisation

Nous avons calculé la précision, le rappel et la f-mesure que notre système obtient sur notre corpus d'évaluation complet de 2 556 tokens avec et sans la correction générique. Nous nous sommes aussi intéressée aux résultats acquis par la correction générique seule, en évaluant ce système sur les 93 altérations n'obtenant pas de

candidats avec nos jeux de règles larges et spécifiques. Cela a pour principal objectif de mesurer l'impact de ce dernier type de correction. Cette évaluation a été faite de manière systématique pour la tâche de correction automatique, en vérifiant que le token attendu fasse partie des candidats proposés pour la tâche de correction (cf. table 5.4), et pour la tâche de normalisation, en vérifiant qu'au moins un des candidats appartienne au même lemme que le token attendu (cf. table 5.5).

Bien que l'écart entre nos scores avec et sans la correction générique ne soit pas très élevé, cette dernière nous permet de traiter plus de tokens (cf. les tables 5.4 et 5.5), sans faire diminuer de beaucoup notre précision. Par ailleurs, on constate que le rappel obtenu par la correction générique seule est très faible (pour les raisons citées en section 5.3.2.1) mais sa précision reste acceptable (70% environ). Nous l'avons donc conservée dans la suite de nos expériences.

Les scores obtenus pour les tâches de correction et de normalisation diffèrent peu. Cela montre que notre système de correction se trompe rarement dans la flexion du token qu'il tente de corriger, dès lors qu'il a correctement identifié le bon lemme. On note que les scores du module de correction générique sont identiques en correction et en normalisation. Cela s'explique par le fait que la correction générique, n'effectuant que des opérations non pondérées sur un caractère, a peu de chances de proposer une correction fautive qui soit une forme fléchie du bon lemme. En effet, supposons par exemple que l'on demande au module de correction générique de proposer un candidat pour l'altération *prèt*. Il pourra par exemple proposer, grâce à une unique substitution, les candidats suivants : *prêt*, *prit*, *près*. Cet exemple illustre le fait que, de façon générale, les tokens les plus proches (au sens de la distance de Levenshtein) d'une même altération ne sont pas tous, loin de là, des formes fléchies d'un même lemme. C'est d'autant plus vrai que le module de correction générique n'est appliqué qu'aux altérations suffisamment *complexes* ou *inattendues* pour que nos règles de correction, qui traitent plus de 96% des altérations, soient inefficaces.

5.3.2.1.4 Baseline pour le système complet Notre système de pondération doit être plus performant qu'un système de sélection aléatoire parmi les candidats proposés. Afin de pouvoir l'évaluer, nous avons choisi de mettre en place une baseline qui sélectionnerait aléatoirement, pour chaque altération, un ou plusieurs candidats de normalisation. À ce stade nous ne savons pas encore si nous conserverons un ou plusieurs candidats. Les trois quarts de nos propositions de candidats n'étant pas supérieures à trois candidats (cf section 5.3.2.1.1), nous nous limiterons aux résultats pouvant être obtenus par cette baseline dans le cas où elle conserve 1, 2 et 3 candidats. Nos candidats étant produits dans un ordre immotivé, nous conservons donc à chaque fois les premiers proposés sans prendre en compte leur poids.

Nous pouvons tout d'abord constater que la tâche de normalisation (table 5.7) obtient de meilleurs résultats que celle de correction (table 5.6). Ce constat était

Nb candidats conservés	Précision	rappel	F-mesure
1 candidat	50,7	48,9	49,8
2 candidats	64,6	62,2	63,4
3 candidats	70,6	68,0	69,3

TABLE 5.6 – Évaluation des candidats après sélection de manière aléatoire pour la tâche de correction

Nb candidats conservés	Précision	rappel	F-mesure
1 candidat	69,3	66,7	68,0
2 candidats	78,9	76,0	77,5
3 candidats	85,3	82,2	83,7

TABLE 5.7 – Évaluation des candidats après sélection de manière aléatoire pour la tâche de normalisation

attendu dans la mesure où cette tâche a des attentes moins strictes. Toutefois, on peut noter de manière plus générale que, pour un système effectuant un choix aléatoire, les scores obtenus sont assez élevés. De plus, les taux de précision, rappel et f-mesure croissent rapidement à mesure qu'on augmente le nombre de candidats conservés. Cela met ainsi en évidence la pertinence des candidats proposés par notre système.

5.3.2.2 Sélection des meilleurs candidats de normalisation

5.3.2.2.1 Procédure mise en place pour évaluer notre pondération

Après avoir évalué la qualité des candidats de normalisation que propose notre système, nous nous sommes intéressée à la pertinence de leurs scores. Pour ce faire, nous justifierons dans un premier temps le choix que nous avons fait, quant à la combinaison de nos différents scores de correction, pour obtenir un score global (section 5.3.2.2.2). Puis dans un second temps, nous décrirons les résultats obtenus en prenant en compte les scores proposés par ce système (section 5.3.2.2.3). Notons que la correction générique, améliorant nos résultats, a été conservée dans notre système. Tous les scores figurants dans la suite de cette section proviendront donc de notre système de normalisation combinant nos jeux de règles de corrections et la correction générique.

Ainsi qu'évoqué précédemment, ce système sera suivi d'un module de désambiguïsation contextuelle (cf. chapitre 6). Nous ne savons pas encore s'il est préférable de laisser ce système sélectionner le ou les candidats de normalisation les plus probables ou de laisser ce système de désambiguïsation effectuer ce choix. Notons toutefois que nous jugeons pertinent, à ce stade, de conserver les deux ou trois candidats les mieux pondérés afin d'effectuer une sélection définitive parmi eux en

nous appuyant sur le contexte. Nous évaluerons par conséquent notre système en conservant le candidat ayant obtenu le meilleur score, mais aussi, en conservant les deux et trois candidats les mieux pondérés [10]. Lorsque plusieurs candidats ont le même score et qu'il est maximal, l'un d'entre eux est choisi aléatoirement.

Comme nous l'avons expliqué en section 5.2.3.2, nous avons fait le choix de pondérer les candidats produits par les règles spécifiques et/ou larges en combinant le score qu'ils ont obtenu dans ces deux jeux de règles avec un score de fréquence : $S_{\lambda_s,\lambda_l}(c,w) = \lambda_s S_s(c,w) + \lambda_l S_l(c,w) + (1-\lambda_s-\lambda_l)F(c)$. Ainsi le score assigné à un candidat peut différer en fonction de la valeur attribuée à chaque λ. Cette variation peut par ailleurs faire fluctuer la précision, le rappel et la f-mesure obtenus par notre système sur le corpus test. Pour l'ensemble de ces configurations, nous avons donc procédé à de multiples évaluations en faisant varier les poids assignés à chaque λ, c'est à dire en faisant varier la valeur de λ_s et de λ_l (la fréquence normalisée étant pondérée par $1-\lambda_s-\lambda_l$).

Plutôt que chercher la valeur optimale de nos trois variables de cette façon, nous aurions pu nous appuyer sur un système reposant sur la maximisation de l'entropie ou sur un perceptron et ainsi les apprendre automatiquement. Néanmoins, une telle approche ne répondrait pas parfaitement à l'objectif que nous nous sommes donnée ici. En effet, nous ne voulons pas classifier les corrections proposées en deux catégories, celles qui sont fausses et celles qui sont correctes, mais les ordonner selon la confiance que l'on a en leur validité. Nous avons toutefois tenté d'évaluer notre système en nous appuyant sur les scores obtenus par un système par maximum d'entropie. Les résultats obtenus sont beaucoup moins bons que ceux acquis avec notre système que nous allons décrire ci-dessous (par exemple, sur la meilleure normalisation, nous perdons plus de 5.5% de précision et de f-mesure).

La somme des valeurs de λ_s et de λ_l doit être égale ou inférieure à 1. Nous avons donc commencé par tester toutes les combinaisons possibles de ces deux coefficients en les faisant varier entre 0 et 1 avec un pas de 0,1. Toutefois, il est apparu que ces coefficients étaient trop élevés et empêchaient la prise en compte de la fréquence. En effet, même si les scores de fréquence sont normalisés entre 0 et 1 (tout comme ceux de nos jeux de règles), ces derniers restent très faibles sur la grande majorité des candidats. Ce constat est dû au fait que seuls très peu de tokens ont un très grand nombre d'occurrences. Ainsi que le démontre la loi de Zipf, la majorité des tokens de la langue ne sont que très peu utilisés. Cela explique donc les scores de fréquences que nous avons obtenus. Nous avons donc réévalué notre système en faisant cette fois-ci varier λ_s et λ_l entre 0 et 0,001 avec un pas de 0,0001.

Les différentes expériences que nous avons menées afin de valider nos hypothèses et d'évaluer les performances de notre système nous ont permis d'obtenir de nom-

10. Nous avons aussi réalisé cette évaluation en conservant plus de candidats mais les résultats étaient très proches de ceux obtenus avec uniquement trois candidats. Nous n'avons par conséquent pas jugé pertinent de les faire figurer dans ce travail.

breux résultats. Ces résultats seront dans la suite de cette section rassemblés sous la forme de figures dont l'abscisse correspondra à la valeur de λ_s et l'ordonnée à la valeur de λ_l. Plus la zone d'une figure est claire, plus la précision, le rappel ou la f-mesure représentés dans cette dernière sera élevé. Par ailleurs, nous avons inséré des courbes de niveau (iso-précision ou iso-f-mesure) au sein de ces figures [11]. L'intégralité des figures détaillant la précision, le rappel et la f-mesure obtenus pour chacune de nos expériences, dans le cas où seuls 1, 2 ou 3 candidats sont concernés, sont par ailleurs mises à disposition du lecteur dans l'annexe A.

5.3.2.2.2 Pondération des règles Avant de nous intéresser davantage aux différents résultats de notre système, il est important de justifier notre choix concernant le calcul de la pondération d'un candidat. En effet, nous avons fait deux hypothèses lors de la mise en place de notre système qu'il convient de vérifier ici.

1. La première des conjectures que nous avons faites est que la valeur de λ_l peut rester la même avec ou sans présence de correction spécifique. On présume ainsi que conserver cette valeur telle quelle n'impactera pas pour autant nos résultats.

2. La seconde conjecture que nous faisons est qu'il est plus pertinent de combiner les deux jeux de règles spécifiques et larges plutôt que de les appliquer les uns à la suite des autres.

Afin de vérifier si la présence des règles spécifiques a une incidence sur la valeur optimale de λ_l, nous avons calculé, dans un premier temps, son score optimal lorsque nous retirons le jeu de règles spécifiques. Pour ce faire, nous avons calculé la f-mesure qu'il obtient sur notre corpus test en fonction des différentes valeurs de λ assignées aux règles larges ainsi que l'illustre la figure 5.3. Dans ce cas précis, la valeur de λ_s n'a aucune incidence sur nos scores puis que les règles spécifiques ne sont pas appliquées.

Dans cette figure, plus le score obtenu par la f-mesure est élevé, plus la zone représentant ce score est claire. Ainsi nous pouvons noter que la valeur optimale de λ_l est comprise entre 0,00005 et 0,0002.

Si on observe à présent les f-mesures acquises en combinant les règles spécifiques et les règles larges, nous obtenons les résultats figurants dans la figure 5.4

Nous pouvons noter dans cette dernière figure que la valeur optimale de λ_l est plus étendue. Elle est comprise entre 0,00005 et 0,0005 en fonction de la valeur de λ_s puisqu'elle est cette fois-ci comprise dans l'évaluation. Ce constat est néan-

11. Ces courbes de niveau sont produites automatiquement par l'outil Gnuplot à partir de l'ensemble des résultats représentés dans la figure concernée. Les détails de ces courbes ne sont donc pas très significatifs. Consciente de ce problème nous avons lissé ces courbes dans la description de nos résultats finaux.

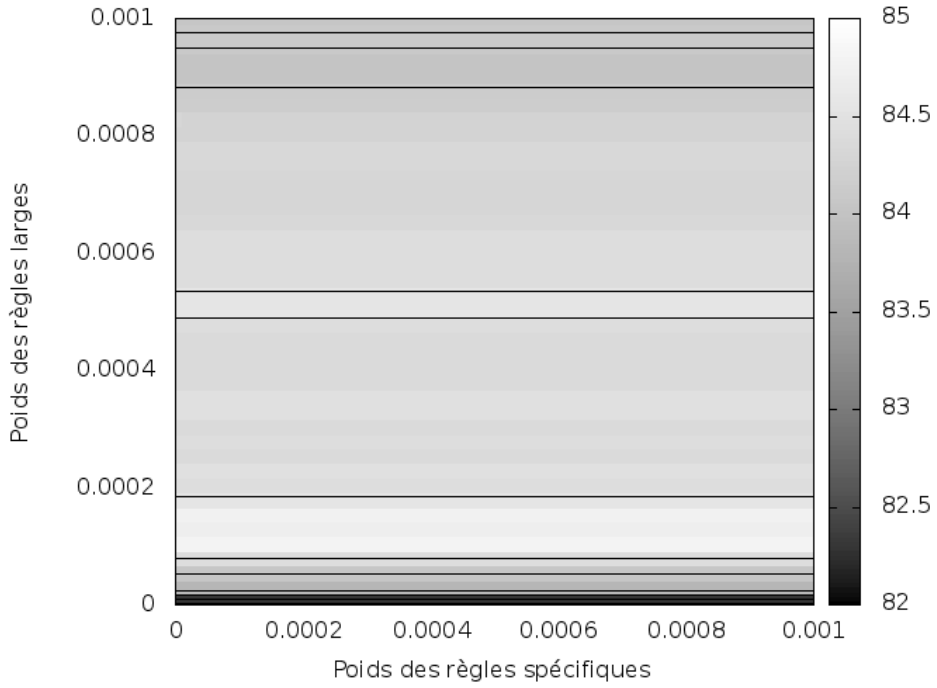

FIGURE 5.3 – F-mesure obtenue en ne conservant que le meilleur candidat produit par les règles larges

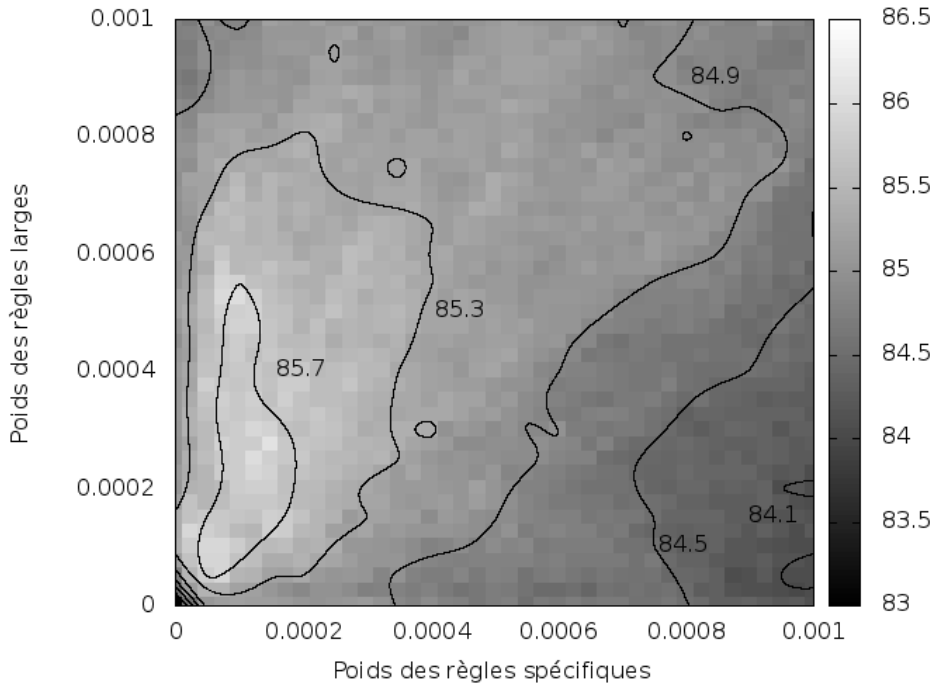

FIGURE 5.4 – F-mesure obtenue en ne conservant que le meilleur candidat produit par les règles spécifiques et larges

moins satisfaisant puisque la fenêtre optimale pour λ_l obtenue dans ce dernier cas couvre celle relevée lorsque nous avions conservé que les règles larges. Il est donc parfaitement possible de conserver la même valeur de λ_l pour calculer le score global d'un candidat, que ce candidat ait été ou non produit aussi par une règle spécifique. Il faut uniquement faire attention à conserver une valeur de λ_l dans les deux fenêtres optimales évoquées ci-dessus.

Nous prenons en compte deux jeux de règles pour calculer notre score global. En fonction de notre manière de combiner ces règles, les résultats que nous obtiendrons différeront. Nous avons pour cela choisi de combiner les impacts que peuvent provoquer les différentes combinaisons possibles entre ces deux jeux de règles. Une première idée serait d'appliquer dans un premier temps uniquement les règles spécifiques pour rechercher les candidats de normalisation d'une altération, puis dans un second temps, d'appliquer les règles larges uniquement dans le cas où les règles spécifiques auraient échoué. Une seconde serait d'appliquer simultanément ces deux jeux de règles. La figure 5.4 illustrait déjà cette seconde idée. La première, quant à elle, est représentée dans la figure 5.5 de combinaisons dans le cas où on ne conserverait qu'un seul candidat. Notons que les figures représentant la précision, le rappel et la f-mesure obtenus en conservant 1, 2 et 3 candidats sont fournies, pour ces deux cas de figure, dans l'annexe A.

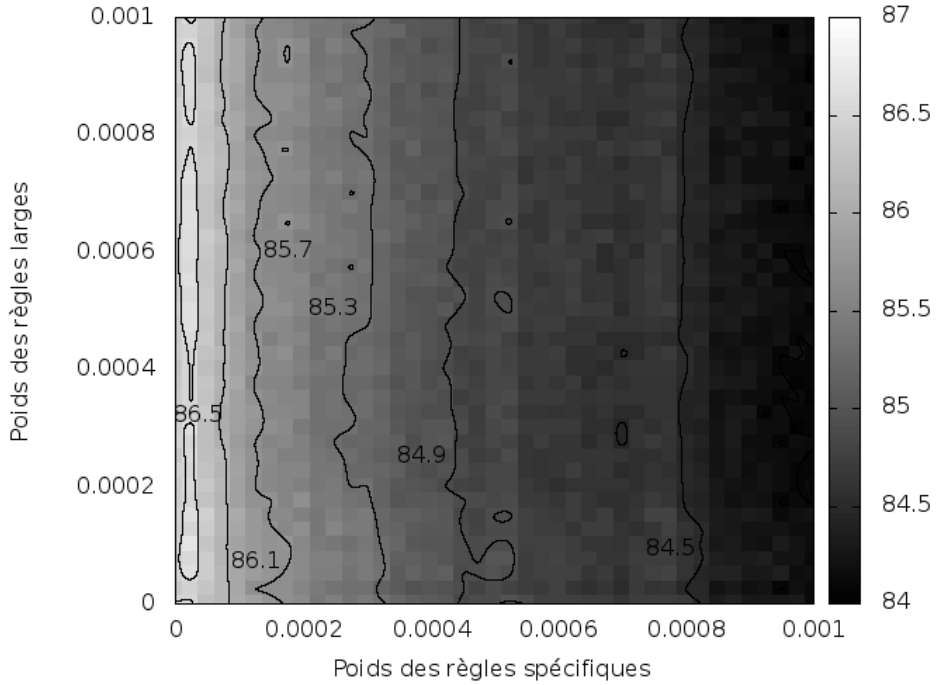

FIGURE 5.5 – F-mesure obtenue en ne conservant que le meilleur candidat produit par les règles spécifiques puis larges

En observant ces deux figures, nous pouvons dans un premier temps constater que nos résultats ne se répartissent pas de la même manière. En effet, on peut

clairement distinguer l'influence du jeu de règles spécifiques dans la figure 5.5 dont les strates verticales restent visibles. La figure 5.4, quant à elle, montre bien le mélange de ces deux jeux de règles. Ainsi dans l'annexe A, ces deux combinaisons comportent certains avantages qui leur sont propres. Les règles spécifiques produisent des candidats de normalisation fiables bien que peu nombreux. Les règles larges suggéreront en parallèle plus de candidats légèrement moins fiables. Appliquer dans un premier temps les règles spécifiques puis larges assure alors une bonne précision au détriment du rappel. Combiner les deux permet par conséquent d'optimiser ce rappel mais diminue légèrement la précision. C'est d'ailleurs pour cette raison que le score optimal obtenu pour cette dernière combinaison est moins bon de 0,5% par rapport à la première combinaison. Toutefois, nous pouvons noter que cet écart est très vite rattrapé puis dépassé dès lors que plus d'un candidat de correction est conservé (cf. figures dans l'annexe A). Le nombre limité de candidats proposés par les règles spécifiques a ainsi un impact sur les résultats pouvant être obtenus par cette dernière combinaison.

C'est ainsi afin de ne pas trop dégrader le rappel de notre système de correction que nous avons préféré opter pour la combinaison des règles larges et spécifiques. Par ailleurs, il est fort possible que nous soyons amenée à conserver plusieurs candidats de normalisation pour une même altération afin de sélectionner le plus probable uniquement dans la dernière étape de désambiguïsation de notre système (cf. chapitre 6). Dans ce dernier cas, c'est bien ce dernier type de combinaison qui est optimal.

5.3.2.2.3 Résultats obtenus Nous avons fait le choix de nous concentrer sur les résultats (précision, rappel et f-mesure) obtenus par notre système quand celui-ci combine nos deux jeux de règles. Ces derniers sont ainsi illustrés dans les figures 5.6 à 5.14. Les résultats de notre système peuvent varier entre les bornes inférieures et supérieures décrites à la section 5.3.2.1, à savoir entre 69,3% et 95,1% pour la précision, 66,7% et 91,6% pour le rappel et 68% et 93,3% pour la f-mesure dans le cas où on conserverait par exemple qu'un seul candidat de normalisation.

Les figures 5.6, 5.7 et 5.8 illustrent la précision, le rappel et la f-mesure obtenus par notre système lorsque l'on ne conserve que le candidat de score maximum pour chaque altération, en fonction des valeurs de λ_s et λ_l. À l'aide des courbes de niveau insérées dans ces figures, on peut constater par exemple que les meilleurs scores pour la précision s'élèvent à plus de 87,2 (la précision maximum atteinte est en réalité de 87,7%), pour le rappel à plus de 84,2% (le rappel maximum atteint est en réalité de 84,5%) et ceux concernant la f-mesure à plus de 85,7 (la f-mesure maximale atteinte est en réalité de 86,1%). Ces scores sont donc bien supérieurs aux bornes inférieures calculées précédemment. La forte similarité entre les figures 5.6 et 5.8 s'explique par le fait que les scores de rappel obtenus, bien que plus faibles que ceux de la précision, se répartissent de la même manière en

fonction des coefficients λ_s et λ_l. À titre indicatif, lorsque λ_s et λ_l varient entre 0 et 0,001, le rappel est presque partout entre 83% et 84%, avec un minimum à 81,4% (lorsque $\lambda_s = \lambda_l = 0$) et un maximum à 84,3%. Ces figures démontrent clairement l'utilité des scores proposés par nos règles lorsque l'on ne garde que le meilleur candidat : si nous ne prenons que la fréquence en compte ($\lambda_s = \lambda_l = 0$), notre f-mesure est de 82,9%, soit 3,2% de moins que la f-mesure maximale). Bien que de façon moins nette, ce constat reste vrai lorsque l'on conserve les deux ou trois meilleurs candidats les mieux pondérés. La f-mesure gagne ainsi 1,5% si on prend en compte les règles pour deux candidats de correction et 0,7% pour trois candidats.

Bien que le passage d'un candidat à deux nous permette d'améliorer nos scores de près de 5% absolus, le passage de deux candidats à trois est légèrement moins significatif. Cela s'explique notamment par le faible nombre de corrections proposées pour chaque altération. Notre système n'attribue trois candidats ou plus qu'à environ 40% de notre corpus de test (cf. figure 5.2). Le nombre d'altérations pour lesquelles nous devons sélectionner les meilleurs candidats est par conséquent automatiquement réduit. Autre conséquence de ce fait : nos scores de précision et de f-mesure se rapprochent de la borne supérieure théorique dès la prise en compte de seulement deux candidats. En effet, avec une précision qui s'élève à 93,3% (cf. figure 5.9), un rappel à 89,9% (cf. figure 5.10) et une f-mesure à 91,6% (cf. figure 5.11), on est à 2% de la borne supérieure pour la tâche de normalisation.

Ces figures montrent que nous obtenons nos meilleurs résultats lorsque l'on donne plus de poids aux règles larges qu'aux règles spécifiques. En effet, bien que les règles spécifiques soient plus précises et plus fiables, elles s'appliquent moins souvent. Les règles larges peuvent ainsi détecter et corriger plus de fautes. Ces deux jeux sont donc complémentaires : nous obtenons de meilleurs scores en associant les deux. Enfin, la prise en compte de la fréquence d'un token pour sa normalisation reste primordiale. Cela est particulièrement visible dans les figures illustrant les résultats obtenus avec deux et trois occurrences (figures 5.9, 5.10, 5.11, 5.12, 5.13 et 5.14) dans lesquelles nos scores se dégradent dès lors que λ_s ou λ_l sont au-dessus de 0,0006 environ, et ce d'autant plus que λ_s ou λ_l sont élevés.

5.4 Conclusion

L'objectif du système présenté dans cette section est de proposer un ou plusieurs candidats de normalisation pondérés pour toutes les altérations inconnues de notre lexique de référence présentes dans un texte. Pour ce faire, nous avons mis en place et évalué un système de normalisation fonctionnant à l'aide de règles de correction induites par analogie. Les résultats obtenus étant satisfaisants, ce système a été conservé afin de constituer un module pouvant

s'intégrer à notre chaîne de traitement. Il prend ainsi en entrée et rend en sortie un texte au format SxPipe. Lorsqu'une altération est détectée et normalisée par ce module elle est annotée avec l'étiquette ◁Corr_ANA suivie de sa pondération et du type des règles qui ont permis cette normalisation. Par exemple, si notre module rencontre l'altération *telephonne*, elle proposera en sortie l'analyse suivante : ({telephonne$_1$◁Corr_ANA_0.361–SLFreq} téléphoné | {telephonne$_1$◁Corr_ANA_0.588–SLFreq} téléphone)

Comme le montre cette sortie, nous avons fait le choix de conserver plusieurs candidats de normalisation. Nous avons pour l'instant autorisé notre système à nous proposer jusqu'à trois candidats. Cette méthode a pour but de lui permettre de conserver les candidats de normalisation les plus probables. Le système de désambiguïsation contextuelle, qui sera introduit au chapitre 6, aura ensuite pour objectif de ne conserver que le candidat le plus probable étant donné le contexte donné et son score de normalisation.

Ce module de normalisation des altérations inconnues a par ailleurs été paramétré en prenant en compte les différentes évaluations réalisées. Nous avons ainsi choisi de fixer les valeurs de λ_s et de λ_l en fonction de leurs valeurs optimales lors qu'elles sont combinées mais aussi lorsqu'elles sont appliquées seules. Nous avons ainsi fixé λ_s à 0,00005 et λ_l à 0,0001. Ainsi qu'illustré dans la table 5.8, ce paramétrage nous permet d'obtenir des scores très proches de la borne supérieure de notre système dans le cas où nous conservons les 3 candidats les mieux pondérés.

Méthode de sélection des candidats	Précision	rappel	F-mesure
Paramétrage conservé pour notre système	94,5%	91,1%	92,8%
Aléatoire	85,3%	82,2%	83,7%
Oracle	95,1%	91,6%	93,3%

TABLE 5.8 – Comparaison des résultats obtenus par notre système de sélection de candidats avec ceux obtenus aléatoirement et à l'aide d'un oracle en conservant les trois meilleurs candidats.

Bien que ce système n'ait pour l'instant été évalué que sur le français, notons qu'il pourrait être adapté à d'autres langues. En effet, nos règles de correction sont extraites automatiquement d'un corpus d'erreurs annotées. Cette normalisation pourrait parfaitement fonctionner pour d'autres langues pour peu qu'il existe une base de fautes corrigées pour cette langue. Cette base pourrait parfaitement, par exemple, être extraite de la Wikipédia correspondante, de la même façon qu'a été construit le corpus WiCoPaCo pour le français ou comme le propose Zesch (2012) pour l'anglais.

Parmi nos perspectives nous aimerions aussi travailler à rendre ce système complètement non supervisé. Actuellement, l'apprentissage de nos règles de correction est effectué à partir d'un corpus de fautes annotées. Il n'existe toutefois que peu de corpus de ce type. C'est pourquoi apprendre nos règles de

façon non-supervisée à partir d'un corpus brut légèrement bruité serait particulièrement intéressant. Cela nous permettrait d'obtenir un système qui ne dépendrait d'aucune ressource particulière mis à part d'un échantillon de la langue et d'un lexique de cette langue. Pour ce faire une solution serait de concevoir automatiquement notre corpus d'apprentissage d'une manière similaire à ce que nous avons décrit dans nos travaux sur l'induction non-supervisée de relations dérivationnelles dans un lexique (Baranes et Sagot, 2014). Dans ce dernier cas, il s'agirait non plus d'induire des règles dérivationnelles mais des règles de corrections orthographiques liant une altération à sa forme correcte.

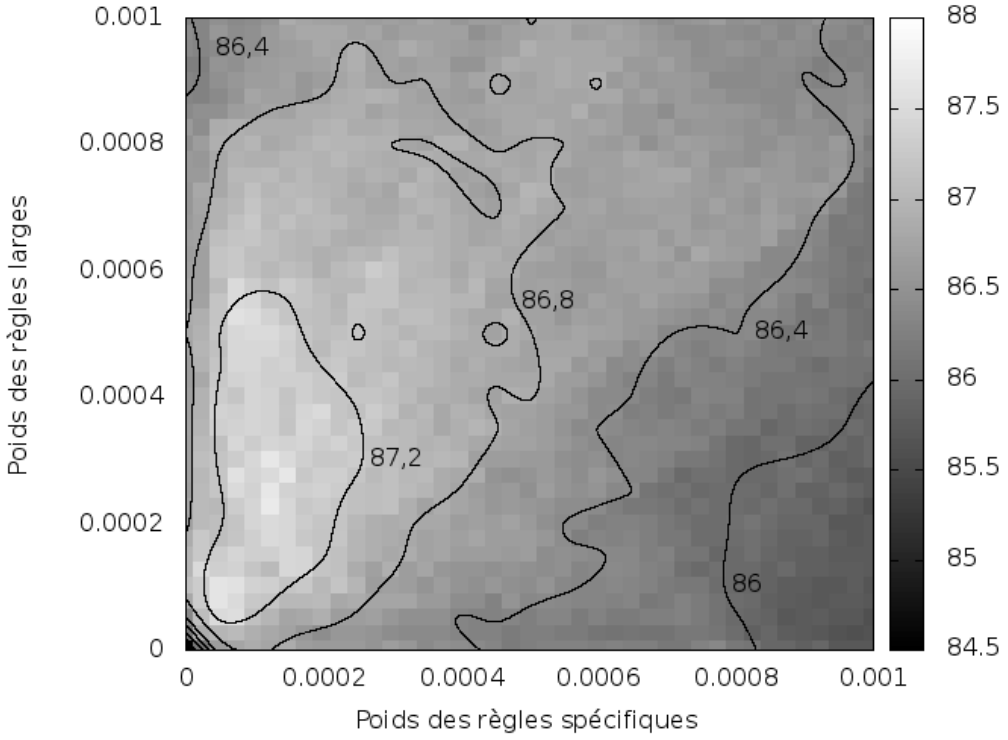

FIGURE 5.6 – Précision pour la meilleure normalisation

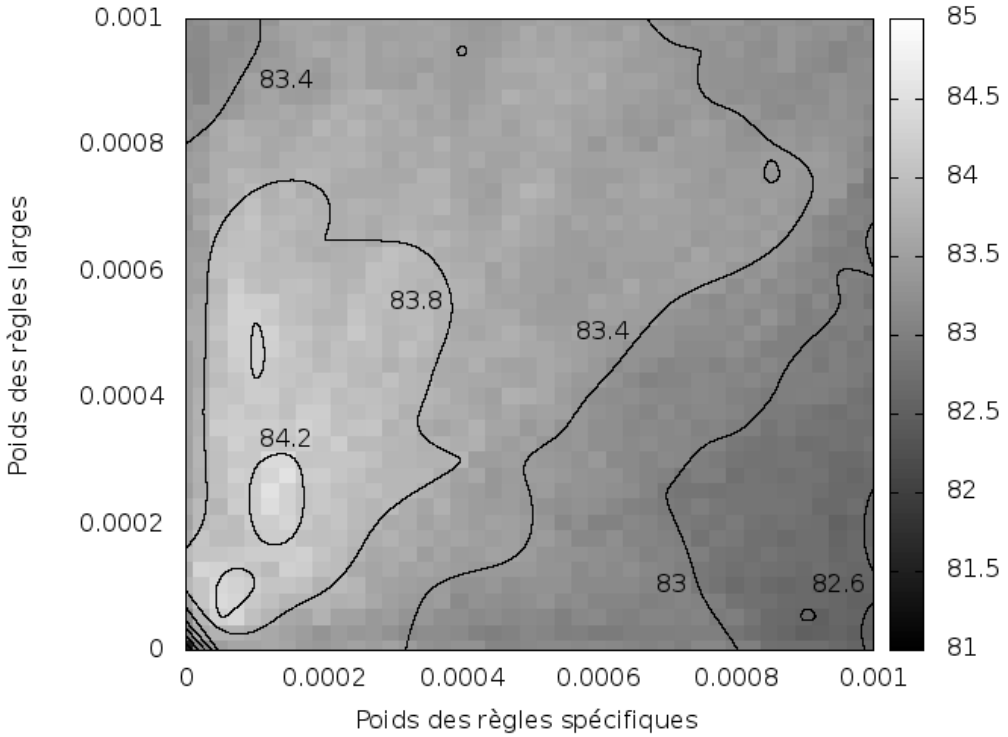

FIGURE 5.7 – Rappel pour la meilleure normalisation

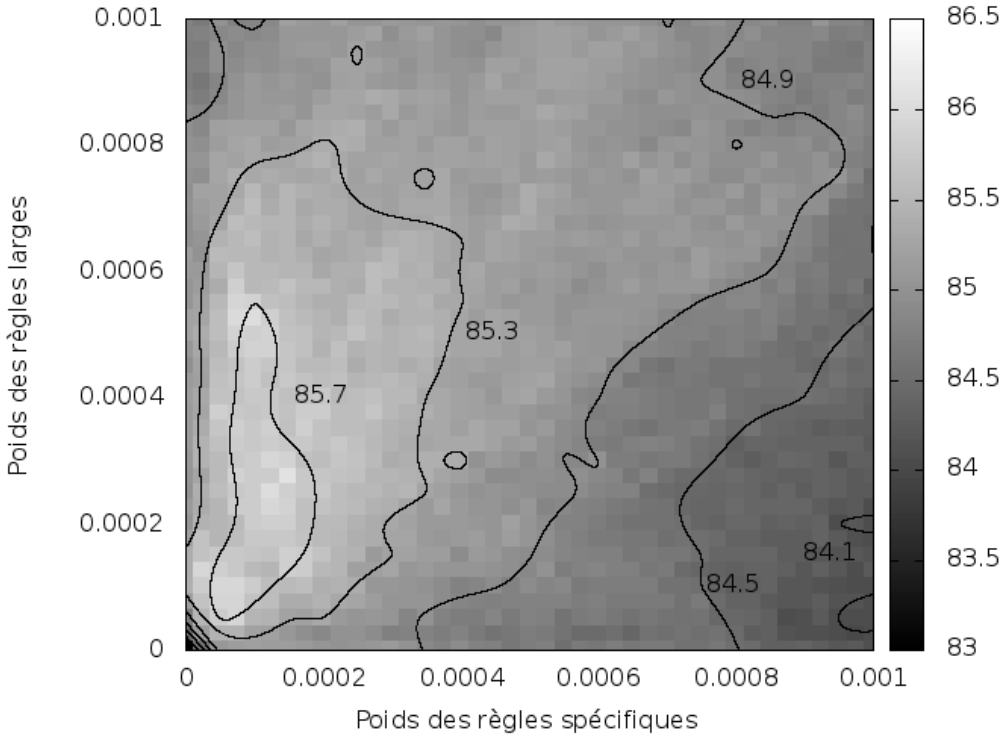

FIGURE 5.8 – F-mesure pour la meilleure normalisation

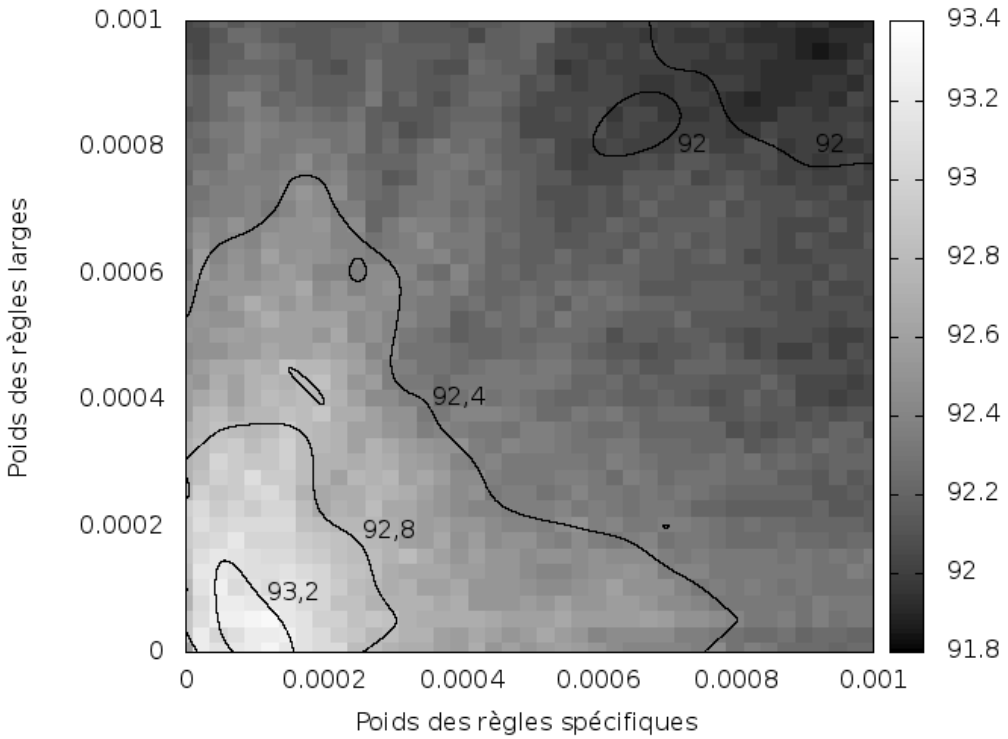

FIGURE 5.9 – Précision pour les 2 meilleures normalisations

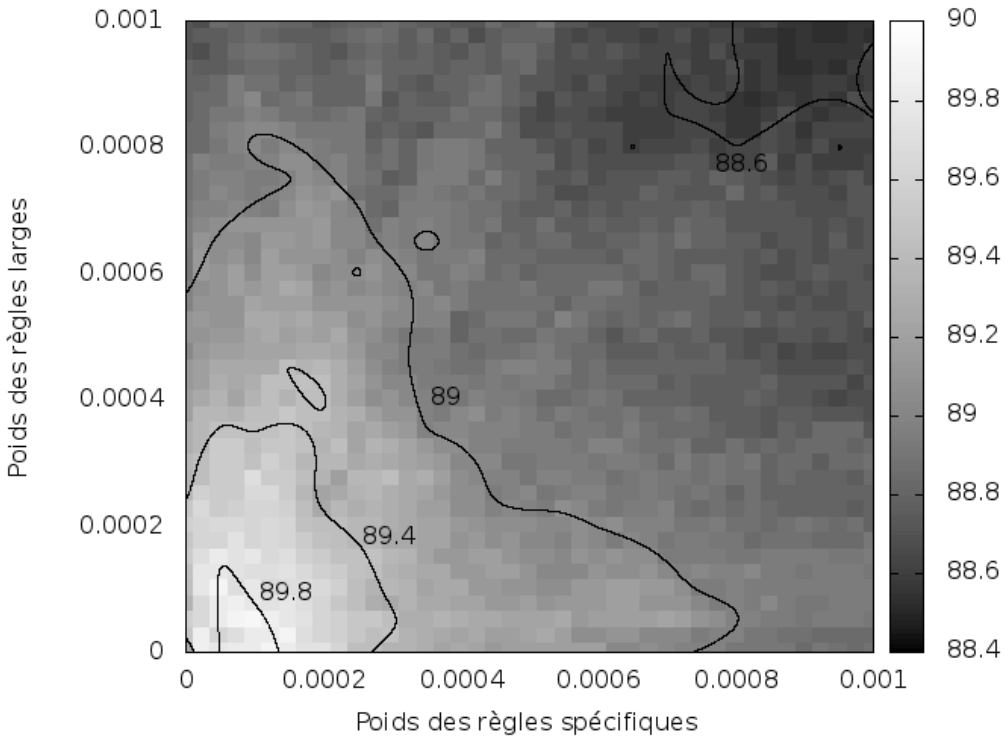

FIGURE 5.10 – Rappel pour les 2 meilleures normalisations

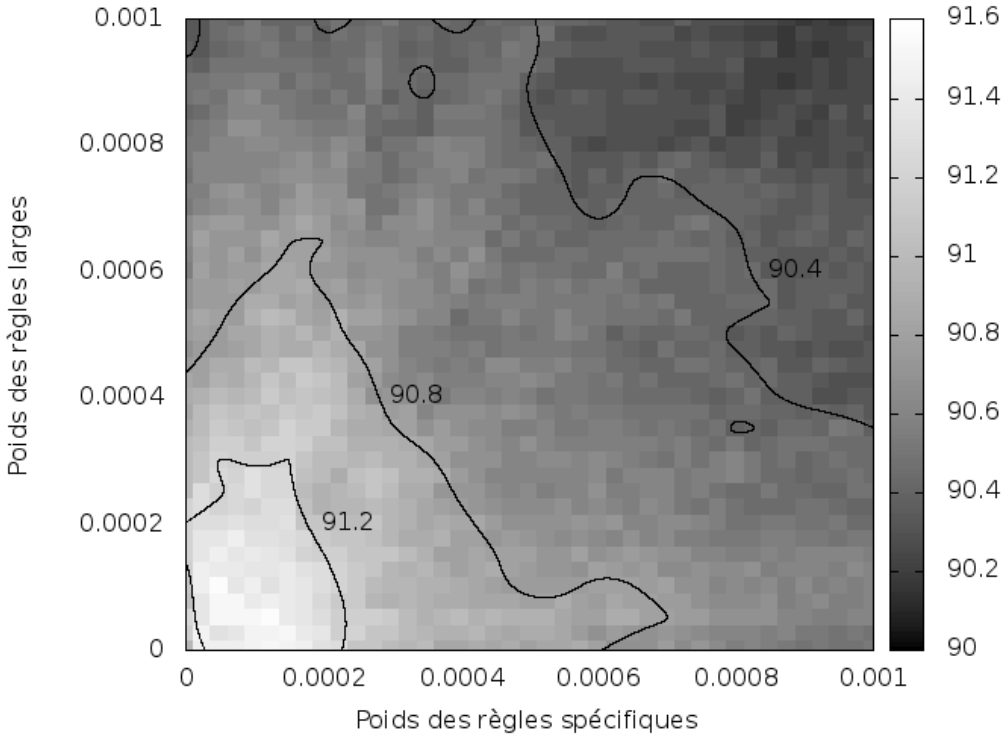

FIGURE 5.11 – F-mesure pour les 2 meilleures normalisations

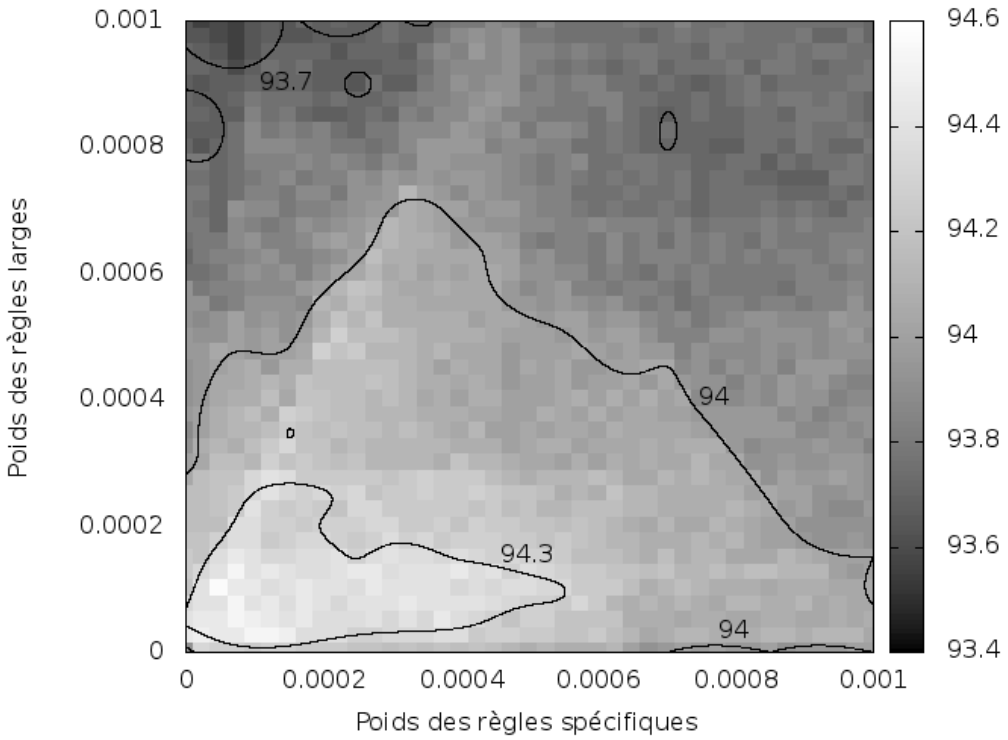

FIGURE 5.12 – Précision pour les 3 meilleures normalisations

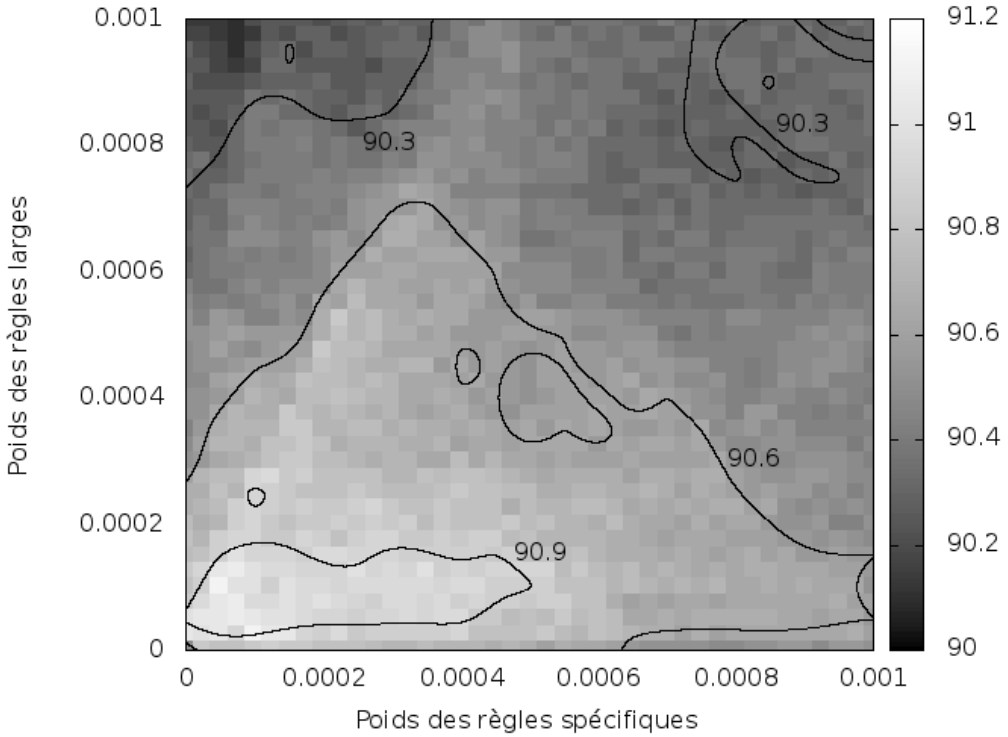

FIGURE 5.13 – Rappel pour les 3 meilleures normalisations

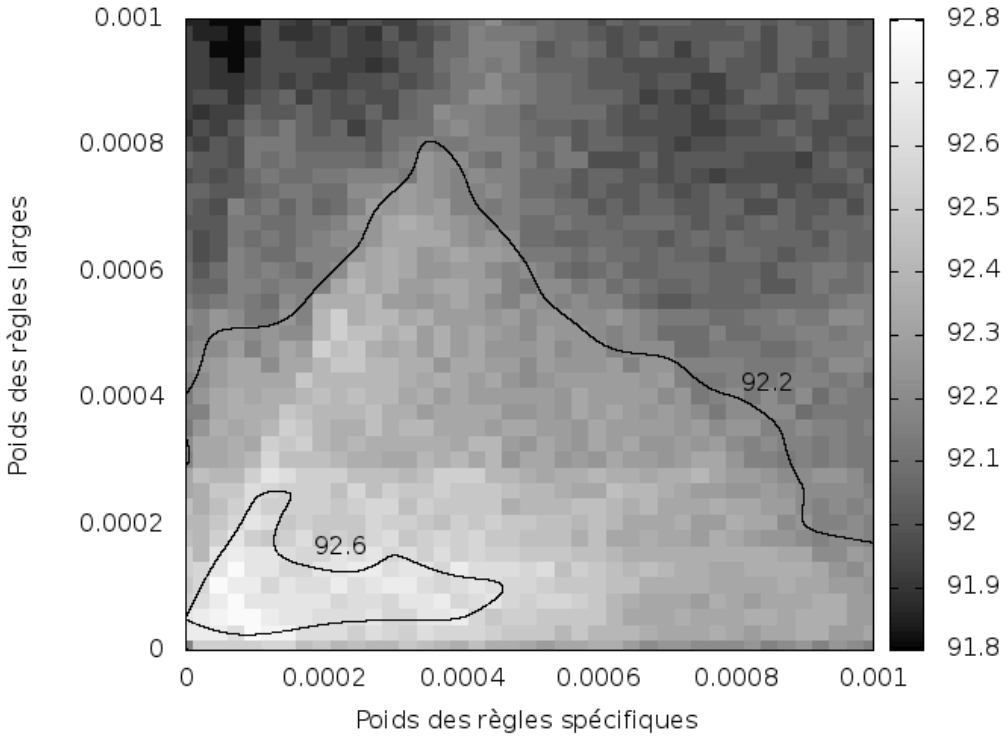

FIGURE 5.14 – F-mesure pour les 3 meilleures normalisations

CHAPITRE 6

Normalisation des tokens connus altérés et désambiguïsation contextuelle

Sommaire

Notre système complet a pour objectif de proposer des candidats de normalisation pour toutes les altérations, que ces dernières produisent des tokens connus ou non. Jusqu'ici, nous nous sommes concentrée sur l'analyse des tokens altérés inconnus, ces derniers ne nécessitant pas d'informations contextuelles. Ils sont ainsi analysés par notre système en deux étapes. Nous les détectons dans un premier temps en écartant les inconnus ne correspondant pas à des altérations (cf. chapitre 4). Puis, nous proposons, dans un second temps, de normaliser les inconnus restants alors considérés alors comme altérés (cf. chapitre 5).

Les tokens connus notre lexique de référence qui ont été altérés de manière non flexionnelle restent donc encore à analyser. Puisqu'il s'agit de tokens connus, on ne peut plus se contenter d'une vérification lexicale. Il est donc à présent nécessaire de recourir à des informations contextuelles.

En outre, nous disposons à ce stade d'une chaîne de traitement qui peut proposer plusieurs analyses pour un seul et même token altéré. À terme, il serait souhaitable de conserver uniquement l'analyse la plus pertinente. Une désambiguïsation du DAG obtenu en sortie de notre chaîne est donc nécessaire, et ce, pour plusieurs points. Tout d'abord, l'outil appliqué en aval de notre chaîne ne traite actuellement pas l'ambiguïté. De plus, dans le cas où nous adapterions cedit outil, il reste préférable de limiter cette ambiguïté. En effet, elle risquerait de générer du bruit et/ou de ralentir ce dernier. C'est pourquoi nous avons choisi de procéder à une étape de désambiguïsation qui ne conserverait que la ou les analyses les plus probables en fonction du contexte. Schématiquement, on peut ainsi représenter l'agencement de ces trois étapes comme illustré dans la figure 6.1.

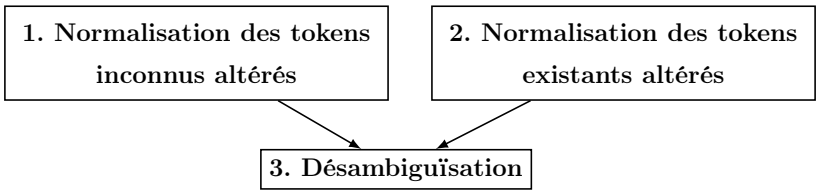

FIGURE 6.1 – Agencement des modules de normalisation et de désambiguïsation

Ces deux dernières tâches ont ainsi en commun qu'elles requièrent la prise en compte du contexte. C'est pourquoi nous avons choisi de les traiter toutes deux dans ce même chapitre. Nous commencerons donc par nous pencher sur la conception de candidats de normalisation pour les tokens existants altérés à la section 6.1, puis nous proposerons un système de désambiguïsation contextuel à la section 6.2.

6.1 Normalisation des altérations existantes

Les altérations produisant des tokens connus de nos lexiques peuvent être réalisées volontairement ou non. Dans le premier cas, elles peuvent par exemple être créées dans le but de faire un jeu de mots (« *pour respirer un peu d'air vrai* » — *Gilbert Cesbron*) comme évoqué à la section 3.1. Dans le second, il s'agira de fautes grammaticales (« *envoyer un colis* ***a*** *une adresse* » ou « *c* ***es*** *pour* ***quant*** *?* »). Dans notre contexte, c'est principalement ce dernier cas qui nous intéresse. La création volontaire d'altérations est plus rare et peu représentée dans nos corpus. Elle sera par conséquent ignorée dans ce travail. Rappelons, par ailleurs, que notre but est de proposer une normalisation à la flexion près. Par conséquent, nous ne nous intéressons pas aux erreurs grammaticales flexionnelles (« *vous* ***avais donner*** *le mauvais numéro.* »).

Afin de mieux cibler les erreurs grammaticales à traiter, nous avons tout d'abord réalisé une rapide étude de corpus nous permettant ainsi de relever les différentes fautes grammaticales [1] attestées dans nos données (section 6.1.1). Nous avons ensuite cherché à détecter automatiquement ces erreurs (section 6.1.2) avant de mettre en place un système de génération de candidats de normalisation (section 6.1.3).

6.1.1 Étude en corpus des fautes grammaticales

L'étude de corpus présentée ci-dessous a été décrite dans (Baranes, 2011). Elle porte uniquement sur les fautes grammaticales et prend en compte notre volonté de travailler sur plusieurs canaux différents (e-mails, forums, réseaux sociaux...). Le corpus étudié a donc été constitué en fonction de l'importance que l'on attribuait à un canal à ce moment et du degré de bruit contenu dans chacun de ces messages. Il est ainsi constitué de :

- 75 e-mails (environ 7500 tokens),
- 120 conversations avec un agent virtuel (environ 10000 tokens) [2],

1. Les termes de *fautes* et d'*erreurs* grammaticales seront utilisés sans distinction dans ce chapitre.

2. Corpus composé des questions de l'utilisateur et des réponses de l'agent virtuel horodatées.

- 120 messages provenant de réseaux sociaux (environ 2000 tokens),
- 55 messages provenant de sites d'avis (environ 3500 tokens),
- 90 messages de forum (environ 5500 tokens).

En moyenne, nous avons trouvé près d'une faute grammaticale tous les deux messages dans les réseaux sociaux et les forums, tous les trois messages dans les e-mails et les avis de satisfaction et tous les cinq messages dans les messages provenant de conversations avec des agents conversationnels. En tout, 148 fautes ont ainsi été annotées. La répartition de ces fautes et de leur type en fonction des différents canaux figure dans le tableau 6.1.

Canal	Répartition des fautes par canal	Répartition des types de fautes par canal : homophonie	Paronymie	autres
E-mails	8,8%	85%	15%	—
Agent virtuel	10,2%	67%	20%	13%
Réseaux sociaux	34,3%	90%	10%	—
Sites d'avis	18,3%	82%	7%	11%
Forums	28,4%	98%	2%	—
Totalité corpus	100%	87,8%	8,8%	3,4%

Table 6.1 – Répartition des fautes grammaticales et de leur type par canal

On constate ainsi que les fautes se répartissent principalement en deux catégories : les erreurs d'homophonie et celles de paronymie. Rappelons que l'homophonie est une relation entre deux tokens de la langue qui partagent la même phonétique, mais qui ne véhiculent pas le même sens (ex : ***Sa*** *coûte cher !*). La paronymie, quant à elle, est une relation lexicale qui regroupe des tokens proches graphiquement ou phonétiquement (ex : *Ça* ***voûte*** *cher !*). On peut aussi parler d'homophonie proche ou d'homonymie approximative. Dans ce chapitre, on utilisera aussi la notion de proximité phonétique pour évoquer l'homophonie et de proximité clavier pour la paronymie. La section autres de ce tableau correspond à des cas particuliers de tokens dont l'utilisation peut être controversée, non fixée ou non partagée par tous (ex : *bogue/bug* ou *ah/ha*).

Les résultats de cette étude montrent que, quel que soit le canal utilisé, ce sont systématiquement les fautes d'homophonie qui sont les plus fréquentes. En effet, près de 88% des fautes grammaticales sont dues à la proximité phonétique. En outre, ces 141 erreurs peuvent être rassemblées en 57 fautes distinctes. Parmi ces dernières, 46 erreurs ne sont produites qu'une seule ou deux fois (ex. : *mètre/mettre*, *vol/vole*, *censé/sensé*, *plus/plu* ou encore *moi/mois*), une dizaine d'entre elles apparaissent entre 3 et 10 fois (ex. : *ai/est/et*, *ça/sa*, *ce/se/ceux*, *peu/peut* ou encore *conseil/conseille*) et seule une est réalisée plus de 10 fois (*a/à*). Ainsi sur 141

fautes grammaticales, près de la moitié correspondent à des fautes peu ou pas réitérées.

6.1.2 Détection des altérations existantes en contexte

De même que pour les tokens inconnus altérés, nous voulons procéder en trois étapes pour normaliser les fautes grammaticales. Cela suppose de détecter dans un premier temps les tokens qui ont subi une altération en nous appuyant sur leur contexte puis, dans un second temps de générer des candidats de normalisation. Une des manières de procéder à cette généralisation est par exemple de réappliquer notre système de correction par analogie sur ces tokens possiblement altérés, ce qui permettrait l'induction de candidats de normalisation pour les erreurs grammaticales de proximité clavier, et de rechercher en parallèle les homophones de ce même token pour couvrir aussi les erreurs de proximité clavier. Enfin, dans un dernier temps, il est nécessaire de sélectionner le ou les candidats plus pertinents en nous appuyant sur leur contexte.

Afin de répondre à la première de ces trois étapes, nous voulions mettre en place un système capable de détecter ce type d'altérations automatiquement en s'appuyant sur le contexte. Notre intuition est la suivante : si un token connu est altéré, alors la probabilité qu'il se trouve à cette position dans une séquence avec ce contexte est faible. Dès lors, nous avons supposé possible sa détection par exemple à l'aide d'un modèle de langage.

Pour ce faire, nous avons tenté d'utiliser notre système contextuel qui s'appuie sur des modèles de langue (décrit ci-dessous en section 6.2) afin de repérer, à l'aide de leur pondération, les fautes grammaticales. Cette expérience ne s'est toutefois pas révélée concluante. En effet, les scores proposés par notre système n'étaient pas constants. Bien que ce système parvienne généralement à mieux pondérer une analyse valide plutôt qu'une erronée, les scores qu'il propose ne semblent pas significatifs de la qualité orthographique de la phrase traitée. Un token bien orthographié pouvait ainsi être doté d'un même score qu'un autre altéré. Cela peut s'expliquer par le fait que nous n'avons pu trouver de corpus d'entraînement assez proche des données que nous voulons traiter.

Cette technique n'étant pas concluante, nous avons abandonné cette première étape afin de nous concentrer sur la seconde, la génération de candidats de normalisation. Toutefois en mettant de côté cette première étape, nous sommes dans l'obligation de considérer tous les tokens connus de notre lexique de référence comme potentiellement altérés. Or, si nous tentons de normaliser toutes les erreurs grammaticales possibles, cette opération peut se révéler très coûteuse . C'est pourquoi nous nous sommes concentrée dans la suite de ce travail sur les fautes grammaticales les plus fréquentes.

6.1.3 Généralisation de candidats de normalisation pour les erreurs d'homophonie

Nous avons ainsi choisi de traiter en priorité les erreurs les plus courantes trouvées dans nos corpus, à savoir les erreurs d'homophonie. Même si notre étude de corpus permet de constater que certaines fautes reviennent parfois plus que d'autres, elle montre surtout que ces dernières ne se produisent pas systématiquement sur les mêmes tokens. Par conséquent, nous ne pouvons pas établir une liste finie de tokens pouvant être mal orthographiés et devons partir du principe qu'ils sont tous susceptibles de l'être. Pour mettre en place notre système, il est donc nécessaire d'obtenir simplement les homophones de chaque token. Une manière simple de réaliser cette opération est d'utiliser un lexique d'homophonie. Nous commencerons ainsi par décrire la ressource que nous utiliserons pour rassembler les tokens partageant un même lien d'homophonie avant de nous concentrer sur le système de génération de candidats mis en place pour ce type de fautes.

6.1.3.1 Ressource lexicale utilisée

Afin de détecter et de normaliser les erreurs d'homophonie en rattachant un token à ses homophones, il est nécessaire de s'appuyer sur un dictionnaire phonétique. Nous utilisons, dans ce travail, le lexique phonétique Lexique 3, décrit en section 1.2.2.1. Ce dernier, doté d'environ 135 000 entrées, est principalement constitué de tokens usuels. Nous aurions certes pu nous appuyer sur un lexique plus fourni tel que le GLÀFF (Hathout et al., 2014) mais ce nombre restreint nous permettra notamment de nous focaliser sur les tokens les plus fréquents pour réaliser notre génération de candidats sans nous attarder sur ceux, plus rares, qui font moins souvent l'objet de fautes. En effet, comme nous l'avons constaté en corpus, les erreurs d'homophonie sont majoritairement réalisées sur des tokens usuels. Un lexique plus large risquerait ainsi de nous induire en erreur.

Pondérer chacune des entrées de Lexique 3 en fonction de sa fréquence dans la langue peut, par ailleurs, nous aider à sélectionner plus précisément les tokens que nous souhaitons analyser. En outre, cela nous permet d'ordonner les candidats de corrections à conserver. Nous avons donc associé à chacune de ces entrées un poids situé entre 0 et 1. Ce poids a été calculé en normalisant de façon affine le logarithme de la fréquence de chaque token dans la Wikipédia [3].

Ainsi, nous pouvons à présent nous appuyer sur une ressource lexicale qui associe chacune de ces entrées à sa retranscription phonétique (au format Lexique 3), à son lemme, à sa catégorie grammaticale et à sa fréquence normalisée ainsi qu'illustré à la table 6.2.

3. Réaliser ce calcul de fréquence sur un corpus à la première personne, plus proche de ceux que nous voulons traiter, serait préférable. Toutefois, il n'en existe pas à notre connaissance qui soit non bruité.

Phonétique [4]	Forme fléchie	Lemme	Cat. Gram.	Fréq. norm.
§	on	on	NOM	0,812
§	on	on	PRO:per	0.812
§	ont	avoir	VER	0,799
§	ont	avoir	AUX	0,799
§	hon	hon	ONO	0,454
m@Z	mange	manger	VER	0,522
m@Z	manges	manger	VER	0,309
m@Z	mangent	manger	VER	0,478
p2	peut	pouvoir	VER	0,777
p2	peu	peu	ADV	0,747
p2	peu	peu	NOM	0,747
p2	peuh	peuh	ONO	0,238
p2	peux	pouvoir	VER	0,524

Table 6.2 – Exemple d'entrées contenues dans Lexique 3 accompagnées de leurs fréquences extraites de la Wikipédia

6.1.3.2 Génération des candidats

Avant toute chose, rappelons que notre chaîne de traitement est construite de manière à ce que toutes les ambiguïtés produites par chacun de nos modules puissent être résolues en nous appuyant sur leurs contextes. À ce stade, nous proposons donc de créer une ambiguïté supplémentaire en ajoutant des candidats de normalisation à tous les tokens dotés d'homophones que nous jugerons pertinents. Cette ambiguïté sera ainsi retirée à l'aide d'un système de désambiguïsation (comme nous l'expliquerons à la section 6.2). C'est cette dernière étape qui nous permettra de réaliser une sélection des candidats conservés parmi ceux créés par le module décrit dans cette section.

Si nous insérons une ambiguïté pour chaque homophonie détectée, le DAG représentant notre texte risque de devenir très, voire trop, ambigu. En effet, sans filtre, une phrase bien orthographiée comme « *il est très cher* » risquerait de devenir un DAG comme celui-ci : « (il|ils|ile|île|îles|hile) (et|hé|ait|est|eh|ai|é) (trait|trais|traits|traie|très) (cher|chair|chairs|chaire|chaires|cherres|chère|chères|chers) ». Tenter de désambiguïser une telle phrase risque d'augmenter la complexité de son analyse, son temps de traitement et d'impacter nos résultats. Nous avons ainsi choisi de procéder comme suit. Pour chaque token connu de notre lexique, nous cherchons

4. Le code phonétique employé dans ce tableau est propre à Lexique 3. Chaque phonème est ainsi représenté par un seul caractère. Ce code se rapproche notamment du code X-Sampa (eXtended Speech Assessment Methods Phonetic Alphabet). Ainsi § correspond [ɔ̃], m@Z à /mɑ̃ʒ/ et p2 à /pø/. Bien que l'on désigne ce code comme phonétique, il s'agit plutôt d'un code phonologique.

ses homophones. S'il en possède, nous vérifions dans un premier temps les lemmes des homophones concernés puis, dans un second temps leurs fréquences. Ce qui nous permettra à terme de déterminer si nous souhaitons les conserver.

1. Tout d'abord, rappelons que nous ne nous intéressons qu'aux erreurs qui ne sont pas flexionnelles. Par exemple, le token *mange* ne partage sa phonétique [mɑ̃ʒ] qu'avec les tokens *manges* et *mangent*. Par conséquent, nous ne le suspecterons pas de nécessiter une normalisation puisqu'il ne s'agit ici que de flexions du lemme *manger*. Si nous prenons à présent le token *peux*, on constate qu'il peut correspondre à une faute flexionnelle (*peux*→*peut*), mais aussi à une erreur d'homophonie. En effet, dans ce cas le token partage la phonétique [pø] aussi avec l'adverbe et le nom *peu* ainsi qu'avec l'onomatopée *peuh*. Ainsi la relation d'homophonie entre deux tokens sera pertinente que s'ils ne partagent pas le même lemme.

2. Ensuite, il est très rare qu'une erreur d'homophonie soit réalisée sur un token peu fréquent ou que son candidat de normalisation soit lui-même peu fréquent. Si par exemple le token *peux* correspond dans un texte à une faute grammaticale, on ne s'attendra pas à ce qu'il soit normalisé par l'onomatopée *peuh* qui est rarement employée. Nous proposons donc d'ajouter un SEUIL MINIMUM DE FRÉQUENCE compris entre 0 et 1 afin d'autoriser ou non la normalisation d'un token et la validité d'un homophone donné. Plus ce seuil est élevé, plus la fréquence des tokens concernés sera importante. Ce seuil aura toutefois les défauts de ses qualités. En effet, imposer une limite parmi les tokens que nous souhaitons prendre en compte nous permettrait d'améliorer la précision de notre système puisqu'on ne conserverait que les tokens les plus fréquents et limiterait en plus de cela la complexité de notre système. Toutefois, cette limite diminuera automatiquement le rappel de notre système. En nous restreignant à un seuil donné, nous admettons que nous ne tenterons jamais de normaliser certains tokens qui auraient pu, malgré cela, être mal orthographiés. La valeur de ce seuil sera ainsi choisie en conséquence lors de l'évaluation de ce système.

Enfin, précisons qu'en fonction de sa provenance, la diction d'un locuteur peut varier. En effet, la phonétique d'un token dépendra souvent de l'accent de celui qui le prononce. Certaines personnes ne feront par exemple aucune distinction phonétique entre les tokens *pomme* /pɔm/ et *paume* /pom/. L'aperture des voyelles notamment peut ne pas être perçue. C'est pourquoi, nous avons fait le choix de ne pas la prendre en compte en autorisant la libre substitution des phonèmes [o] et [ɔ] et des phonèmes [e] et [ɛ]. Les tokens *été* /ete/ et *était* /etɛ/, par exemple, seront ainsi considérés comme homophones.

Ainsi si on rencontre dans notre texte le token *été*, souvent confondu avec *était*, nous chercherons ses homophones. Ce dernier en possédant, il se verra attribuer le label ◁Homophone et obtiendra une sortie au format SXPipe telle que celle-ci : ({été◁Homophone} était|{été◁Homophone} été|{été◁Homophone} étaient)

6.1.4 Évaluation

À ce stade nous ne pouvons pas évaluer notre module sur la normalisation des fautes d'homophonie. Cette évaluation se fera dans la section suivante en le couplant au module de désambiguïsation. Ainsi afin de juger de la qualité de ce module de génération de candidats de normalisation seul, nous avons extrait toutes les fautes grammaticales et leur contexte contenus dans le corpus annoté utilisé pour l'évaluation de notre système de prétraitement (voir en section 4.2.9 la description de ce corpus). Ce corpus contient au total 105 fautes grammaticales.

La capacité de notre module à proposer des candidats de normalisation pertinents pour les tokens existants altérés dépend en grande partie de la valeur que nous aurons fixée pour le seuil minimum de fréquence normalisé. Nous avons ainsi choisi empiriquement d'observer les différents résultats obtenus lorsque ce seuil a les valeurs suivantes : 0,60, 0,65, 0,70, 0,75, 0,80, 0,90. La table 6.3 illustre ainsi la précision, le rappel et la f-mesure obtenus pour la tâche de génération de candidats de normalisation pour les fautes grammaticales.

	Seuil homophonie					
	0,60	0,65	0,70	0,75	0,80	0,90
Précision	1,00	1,00	1,00	1,00	1,00	1,00
Rappel	0,78	0,68	0,62	0,55	0,52	0,01
F-Mesure	0,88	0,81	0,76	0,71	0,69	0,02

Table 6.3 – Résultats obtenus pour notre module de génération de candidats pour les fautes grammaticales

Les résultats obtenus par ce module sur notre corpus d'erreurs grammaticales en contexte sont satisfaisants. La précision montre que, lorsque ce dernier suggère des candidats de normalisation, le candidat attendu pour la faute grammaticale concernée est systématiquement proposé. Toutefois, obtenir une très bonne précision n'est pas très complexe dès lors que notre module propose pour une altération tous ses homophones. C'est donc principalement le rappel obtenu par notre système qui nous intéressera pour cette tâche précise. En effet, ce dernier illustre bien l'importance du choix de notre seuil limite de fréquence. Si ce dernier est trop élevé, il ne tentera de normaliser que trop peu de fautes grammaticales. Ce constat se reflète d'ailleurs dans le rappel obtenu par notre système lorsque notre seuil minimum est fixé à 0,90. Toutefois, notons qu'un seuil dont la valeur serait trop petite permettrait certes de normaliser plus d'altérations, mais risquerait à terme de générer trop de bruit sur un corpus entier. Sélectionner la valeur la plus adéquate pour ce seuil est délicat. C'est pourquoi nous avons fait le choix de la déterminer non pas à la suite de cette évaluation, mais lors de l'évaluation générale de notre chaîne.

Notons enfin que l'on a tendance à se tromper sur certains types de tokens plus que sur d'autres. Ajouter, en plus d'un seuil de fréquence, un second score représentant la probabilité qu'une personne se trompe dans l'orthographe d'un token aurait donc pu être pertinent. Toutefois, pour rendre cela possible, un corpus annoté à grande échelle, fiable et proche de ceux que nous souhaitons normaliser aurait été nécessaire. Or, nous ne disposons pas de telles ressources [5].

Notre système suggère, malgré ce filtre de fréquence, de nombreux candidats de normalisation à la fois pour les tokens connus, comme expliqué ci-dessus, et pour les tokens inconnus, comme décrits dans les deux chapitres précédents. Il ne peut par conséquent être pertinent et efficace que s'il est suivi d'un bon système de désambiguïsation contextuelle. Ce système de désambiguïsation est décrit dans la section suivante.

6.2 Désambiguïsation contextuelle

Avant de décrire plus en détail ce système de désambiguïsation en section 6.2.2, un module qui vise à simplifier et nettoyer le DAG à traiter a été mis en place (section 6.2.1). Rappelons ici que SxPipe rend en sortie toutes les combinaisons d'analyses possibles qu'il a trouvé, étant donné les tokens contenus dans le texte passé en entrée, sous la forme d'un DAG (*Directed Acyclic Graph* – graphe orienté acyclique).

6.2.1 Nettoyage des DAG

Le but de ce module est d'améliorer la pertinence des chemins contenus dans le DAG obtenu par notre chaîne de traitement. Pour ce faire, nous ne voulons conserver au sein de chaque DAG que les analyses pertinentes et stocker les annotations proposées par nos divers modules uniquement dans les commentaires. En outre, si un inconnu n'a pas été analysé par notre système de normalisation et demeure suspect, nous avons fait le choix de permettre au module de correction de la chaîne de traitement SxPipe de proposer des candidats de correction. De la sorte, cela nous permettra de passer d'un DAG tel que celui proposé pour la figure 6.2 à un DAG comme illustré dans la figure 6.3 si on a la phrase « *le collis est pas làs* » [6]. Revenons plus précisément sur ces différentes étapes de nettoyage et de complétion du DAG.

5. Le corpus WiCoPaCo aurait certes pu être envisagé, mais il diffère grandement de nos corpus.

6. Les tokens annotés « Sxp » correspondent aux corrections proposées par le correcteur intégré à Sxpipe.

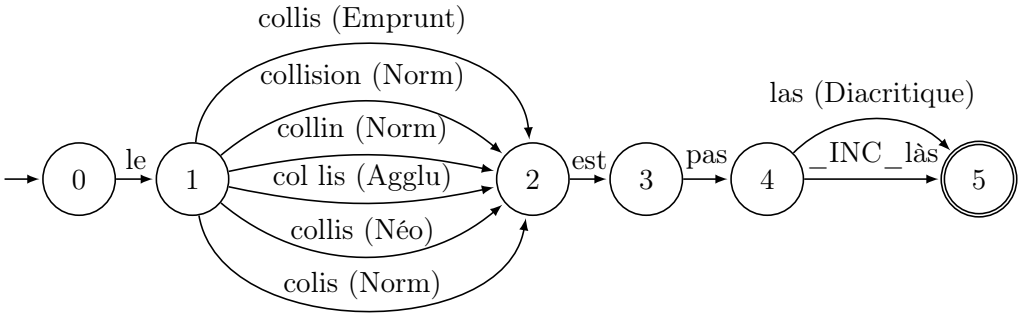

FIGURE 6.2 – DAG avant lissage et ajout des candidats proposés par le module de correction de SXPipe « *le collis est pas làs* »

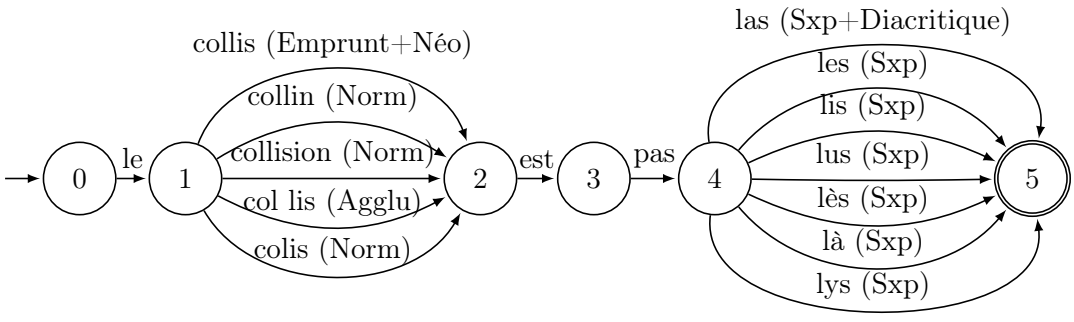

FIGURE 6.3 – DAG après lissage « *le collis est pas làs* »

6.2.1.1 Réécriture des formes annotées

Dans SxPipe, chaque forme composant un texte est représentée par deux éléments : (i) le ou les tokens initiaux la composant indiqués en commentaire (entre accolades) et (ii) la représentation de la forme en question à l'état actuel. À la suite de l'application de notre chaîne de traitement sur un texte, il peut arriver que certaines représentations des formes aient été modifiées. Elles peuvent avoir été normalisées et dans ce cas leurs commentaires ont été annotés en conséquence à l'aide du symbole $\triangleleft$. Ou encore, elles peuvent avoir été étiquetées dans la représentation même de leur forme afin de conserver ou de faire ressortir certaines informations. Un inconnu non catégorisé, par exemple, sera annoté à l'extérieur des commentaires comme suit {token$_1$} _INC_Forme. Afin de ne pas gêner toutes les analyses à venir et principalement notre futur module de désambiguïsation, il est nécessaire de rendre à la forme de chaque token sa forme soit initiale dans le cas soit analysée en conséquence. Toutes les étiquettes conçues par la chaîne de traitement SxPipe adaptée sont ainsi déplacées dans les commentaires. Par exemple, si le smiley :) a été détecté, il aura été étiqueté comme ceci : {:$_1$ \)$_2$} _SMILEY et sera ensuite réécrit ainsi : {:$_1$ \)$_2\triangleleft$SMILEY} :\)

6.2.1.2 Suppression des analyses redondantes

Il peut arriver qu'un inconnu se voit attribuer des étiquettes par de nombreux modules. Par exemple, prenons le token *novelle*, il sera :

- soupçonné par le module de détection des tokens étrangers d'être un emprunt non adapté,
- considéré comme une possible agglutination des tokens *novel*[7] et *le* par notre module prévu à cet effet,
- reconnu par notre module de détection de néologismes comme une possible création lexicale faite à partir de *novelliser*,
- normalisé par notre système de normalisation des tokens inconnus altérés en *nouvelle* (avec un score de 0,74), *novel* (avec un score de 0,50) ou *novella* (avec un score de 0,44).

Si on conserve toutes ces analyses on se retrouve avec une liste de 6 réécritures possibles pour ce token dont deux identiques : *novelle, novel le, novelle, nouvelle, novel et novella*. On fait donc le choix de les rassembler en une seule et même analyse ainsi que l'illustre le DAG représentant notre exemple à la figure 6.4.

7. Notons que les tokens *novel* et *novella* correspondent à des entités nommées prise en compte par notre système. L'absence de majuscule en début de ce mot s'explique par le fait que l'on conserve la casse choisie par l'utilisateur.

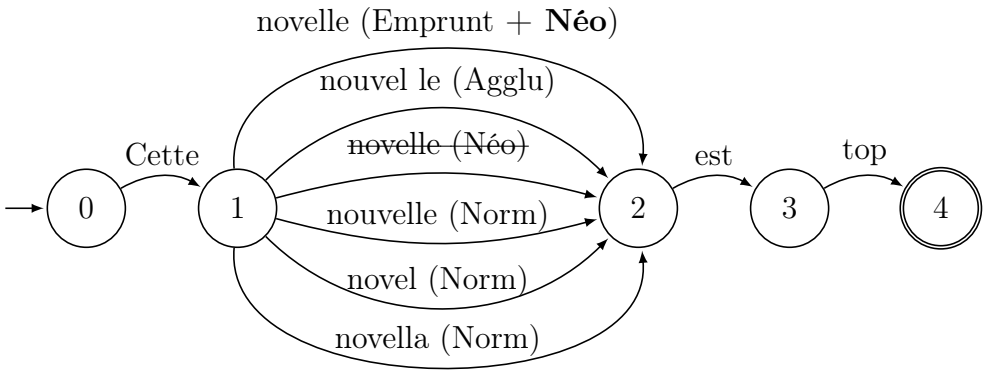

FIGURE 6.4 – Nettoyage du DAG représentant la phrase « *Cette novelle est top* »

À ce stade, nous voulons uniquement réunir les analyses portant sur les tokens inconnus non altérés. Sans contexte, nous ne pouvons pas décréter objectivement quelle est l'analyse la plus pertinente. En effet, bien que l'exemple proposé ci-dessus soit plus favorable à notre système de normalisation des altérations inconnues aux dépens du module de détection des agglutinations, l'inverse peut aussi arriver. Nous ne priorisons donc pas les différents modules de normalisation entre eux.

6.2.1.3 Ajout des candidats de correction proposé par SxPipe

Notons enfin qu'il arrive qu'un token reste considéré comme inconnu en fin de chaîne. Dans ce cas, afin d'optimiser les chances de normalisation de notre système, nous utilisons le correcteur ambigu text2dag intégré à SxPipe. Comme expliqué précédemment, ce correcteur fonctionne à l'aide de règles de substitution pondérées manuellement permettant de remplacer un ou plusieurs caractères. Il a l'avantage d'être paramétrable. On peut ainsi définir le poids de correction maximum souhaité ou encore le nombre maximum de candidats suggérés. Soulignons toutefois que plus le poids d'une correction est élevé, moins elle est sûre. En fonction de son paramétrage, ce correcteur peut donc être très bruyant. À l'inverse, si on le restreint trop, il ne proposera qu'un faible nombre de corrections peu coûteuses (telles que des fautes d'accents par exemple), mais fiables. Notre système étant suivi d'une étape de désambiguïsation, nous avons fait le choix de laisser à ce correcteur une marge de manœuvre assez conséquente. Nous l'avons ainsi limité à 10 candidats dont le poids de correction ne dépasserait pas 140 (à titre indicatif, une correction légère est pondérée à 5). Ses corrections, annotées ◁SXP_Poids, pourront être distinguées des autres candidats de normalisation. Nous pourrons ainsi évaluer l'apport de ce correcteur sur nos résultats.

6.2.2 Désambiguïsation d'un DAG

Suite à leur nettoyage, nos DAG demeurent ambigus et peuvent toujours contenir pour un même token plusieurs candidats de normalisation, qu'il s'agisse d'un token inconnu identifié comme une altération ou d'un token connu pour lequel les candidats homonymes ont été proposés. Prenons la phrase : « *Le **pri** est élevé !* » illustrée figure 6.5. Nous savons que *pri* peut être normalisé par *pris, prie, prix...* Toutefois, sans informations supplémentaires, il est difficile de prédire quel candidat de normalisation est le plus adéquat. Nous avons donc besoin de connaître la probabilité d'avoir un candidat plutôt qu'un autre en fonction du contexte dans lequel il s'inscrit.

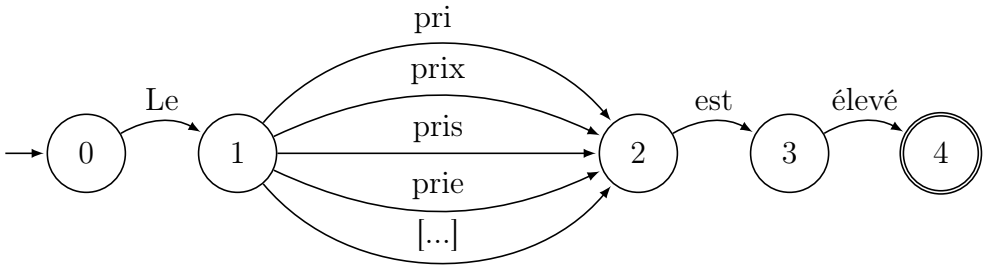

FIGURE 6.5 – Probabilité pour chaque candidat de normalisation d'apparaître dans le contexte ... « *Le pri est élevé* »

Pour cela, nous proposons de modéliser la langue de manière stochastique. Ce type de modélisation permet entre autres de calculer la probabilité qu'a un système de rencontrer une séquence donnée dans une langue donnée.

6.2.2.1 Outil choisi pour la génération du modèle de langue

Il existe d'ores et déjà plusieurs outils qui implémentent des modèles de langue et qui les utilisent de manière efficace afin de calculer la probabilité d'une séquence. Nous avons choisi d'utiliser l'un d'entre eux afin de ne pas avoir à implémenté un système qui serait possiblement moins performant que ceux déjà existant. Cet outil doit toutefois répondre à plusieurs critères. En effet, il doit tout d'abord être libre de droits et capable de traiter de gros volumes de données en peu de temps. En outre, nous avons besoin d'un système qui peut prendre des DAG en entrée. C'est notamment pour ces raisons que notre choix s'est porté sur l'outil KenLM [8] (Heafield, 2011), un outil qui était état de l'art au moment où ce choix a été fait. KenLM permet notamment la construction d'un modèle de langue à partir d'un corpus donné et l'estimation d'une ou plusieurs phrases en fonction de ce modèle.

8. `http://kheafield.com/code/kenlm/`

Bien que KenLM ait été initialement développé pour traiter du texte brut, cet outil est à présent apte à prendre des DAG en entrée. Cet ajout a été réalisé récemment afin de répondre à un besoin émis par le projet PACTE [9]. Ainsi une version de KenLM nommée *Lazy* est disponible sur le site de l'auteur [10]. Il prend donc un texte sous un format de DAG défini et renvoie les X chemins les plus probables. En outre, cet outil a l'avantage d'être paramétrable. Il nous permet entre autres de pondérer nos candidats de normalisation au sein du DAG. Cette option est d'autant plus intéressante que nous pouvons ainsi réutiliser les poids de nos candidats de normalisation et les prendre en compte lors de la désambiguïsation [11].

6.2.2.2 Modèles de langue construits

Les résultats qui seront obtenus par ce système de désambiguïsation dépendront principalement du modèle de langue appris via *Lazy*. Deux questions doivent essentiellement être posées lors de la construction d'un modèle de langue : (1) sur quel corpus doit-on l'entraîner ? et (2) comment doit-on l'entraîner ? Nous reviendrons donc ci-dessous sur les choix que nous avons faits concernant ces deux critères.

6.2.2.2.1 Quel corpus d'entraînement ? Le corpus d'entraînement est censé représenter la langue de la manière la plus complète possible tout en prenant en compte l'utilité qu'il aura par la suite. Par exemple, si nous souhaitons traiter uniquement des textes portant sur l'automobile, utiliser un corpus d'entraînement spécialisé sur le sujet serait parfaitement pertinent. Dans notre cas, nous ne pouvons pas prédire les sujets susceptibles d'être abordés dans nos messages. Le style d'écriture adopté dans ces derniers est variable ce qui fait que nous ne pouvons pas non plus cibler un type de corpus particulier.

9. Le projet PACTE (Projet d'Amélioration de la Capture TExtuelle) (2011-2015) avait pour objectif l'amélioration de la performance des processus de capture textuelle (OCR, reconnaissance d'écriture manuscrite, saisie manuelle, rédaction) par l'utilisation du traitement de la langue, et plus spécifiquement des méthodes linguistiques statistiques et hybrides. Il concerne le domaine applicatif des projets de numérisation du patrimoine écrit, dans un contexte multilingue. Il réunissait trois entreprises (Numen, A2IA, Isako) et deux laboratoires (Alpage, LIUM).

10. `https://github.com/kpu/lazy`

11. Notons que cet outil peut présenter un défaut pour une application comme la nôtre. Le temps de chargement d'un modèle peut prendre un léger temps et doit être rechargé à chaque nouvel appel de l'outil. Pour une utilisation occasionnelle, cela n'est pas gênant, mais, à terme, cette solution ne peut pas être viable pour un système comme le nôtre qui risque d'être relancé à chaque nouvelle arrivée de texte. Pour pallier cela, notre modèle de langue sera chargé en arrière-plan afin que cela ne soit fait qu'à une reprise. Pour ce faire, nous utilisons un second outil, *KendecServeur*, développé par le projet Pacte.

La sélection de notre corpus d'entraînement est d'autant plus importante que ce dernier influencera les choix que fera notre module de filtrage. Il faut donc qu'on ait un corpus :

1. qui ressemble un minimum aux données qu'on veut traiter,
2. qui soit bien orthographié,
3. qui traite de sujets très différents afin qu'il soit le plus couvrant possible.

Un corpus comme celui-ci n'existe pas à notre connaissance. Le second point est particulièrement primordial dans notre cas. En effet, nous avons absolument besoin d'un texte très peu bruité si nous souhaitons que notre modèle de langue soit pertinent pour une tâche de normalisation. Les textes écrits par les utilisateurs ne répondent malheureusement que très rarement à ce critère. Nous avons donc choisi, pour apprendre nos modèles de langue, le corpus Wikipédia qui couvre nos deux derniers points.

Notons que le choix de ce corpus a été effectué à défaut de mieux. Il serait possible de tenter de mélanger plusieurs types de corpus distincts afin d'obtenir un corpus d'apprentissage plus proche de nos données de tests par exemple. En effet, avoir par exemple plusieurs textes écrits à la première personne au sein de ce corpus d'entraînement optimiserait nos résultats. De même, nous pourrions envisager de mettre en place à terme plusieurs corpus d'entraînement en fonction des textes que nous voulons analyser afin que ces derniers soient plus performants en fonction du sujet traité.

Enfin, la qualité du modèle de langue va dépendre du corpus choisi, mais aussi de la quantité de texte contenu dans ce dernier. Plus la taille du corpus sera conséquente plus notre modèle de langue sera volumineux. Ne sachant pas quelle quantité serait adéquate, nous avons choisi d'apprendre et d'utiliser différents modèles de langue à partir de sous-corpus de la Wikipédia de tailles variés. Ces corpus sont illustrés dans la table 6.4.

Corpus	nombre de tokens
100% Wiki	534 124 387
50% Wiki	257 574 283
25% Wiki	169 575 482
33% Wiki	134 004 732
17% Wiki	93 853 896

Table 6.4 – Tailles de corpus d'entraînement possibles

6.2.2.2.2 Quel contexte apprendre ? Une fois, un corpus d'apprentissage sélectionné, il est important de bien choisir la manière dont on veut le modèle de langue. Ne prendre en compte que le nombre de tokens qu'il contient reviendrait

à apprendre la fréquence d'un token dans une langue sans conserver son contexte. À notre stade, conserver des informations sur les voisinages possibles pour chaque token de la langue est essentiel. Il est donc nécessaire de définir plus précisément la taille du contexte que nous souhaitons prendre en compte. On s'interroge ainsi sur la taille des n-grammes que notre modèle de langue doit apprendre de notre corpus de référence. Plus la taille de n sera grande, plus les informations stockées seront spécifiques. Au contraire, plus la taille des n-grammes conservés sera petite, plus les informations stockées seront générales (les caractéristiques de la langue risquent alors d'être mal représentées). Usuellement, on s'appuie sur des trigrammes, souvent considérés comme un juste milieu entre ces deux problèmes. Dans ce travail, toutefois nos évaluations seront effectuées en utilisant des modèles de langues contenant uniquement des bigrammes, uniquement des trigrammes ou encore des quadrigrammes. Ce ne sera que suite à ces expériences que la valeur des n-grammes conservés sera statuée.

Passer par un modèle de langue s'appuyant sur des tokens pour sélectionner à terme des candidats de normalisation pertinents peut sembler maladroit voir contradictoire. En effet, il aurait été préférable de travailler plutôt sur la notion de lemme puisque notre tâche est effective au lemme près. Toutefois, apprendre un modèle de langue lemmatisé supposerait de pouvoir ensuite le réappliquer sur un corpus de test lui aussi lemmatisé. La lemmatisation étant rarement concluante sur des textes bruités, procéder de la sorte risquerait de dégrader grandement nos résultats. Par ailleurs, au vu des différents résultats que nous avons obtenus jusqu'à présent, il est fort probable que le token qui était attendu pour une altération figure parmi les candidats de normalisation valides que nous aurons proposés. Cette hypothèse donc confortée par le faible différentiel constaté au chapitre précédent entre les scores obtenus par notre système pour une tâche de correction et pour une tâche de normalisation.

6.2.2.3 Application du modèle de langue sur nos textes

L'application du modèle de langue sur nos données est ensuite assez simple à réaliser. En effet, cela consiste à récupérer notre DAG obtenu en sortie de chaîne de traitement et à le convertir au format accepté par *Lazy*. *Lazy* récupère ainsi nos données et nous renvoie, pour chaque DAG, les n phrases les plus probables. Ces dernières contiennent ainsi les candidats de normalisation les plus pertinents étant donné leur contexte. La valeur de n sera déterminée lors de notre évaluation. Afin de ne pas avoir à charger notre modèle de langue pour chaque nouvelle phrase traitée, rappelons que nous utilisons en parallèle *KendecServeur*. Ce second outil nous permet de lancer *Lazy* comme un processus et de stocker ainsi notre modèle en arrière-plan. Une fois que notre modèle de langue a extrait de notre DAG de sortie les n candidats de normalisation les plus pertinents, nous filtrons notre DAG initial afin de ne conserver que les candidats retenus par *Lazy*.

Ainsi, si nous prenons par exemple la phrase : « *le syte et claire* », sa normalisation se fera en trois temps :

1. analyse de la phrase par notre chaîne de traitement adaptée (cf. figure 6.6),
2. pondération des différents chemins possibles (cf. figure 6.7),
3. sélection de la meilleure analyse (cf. figure 6.8).

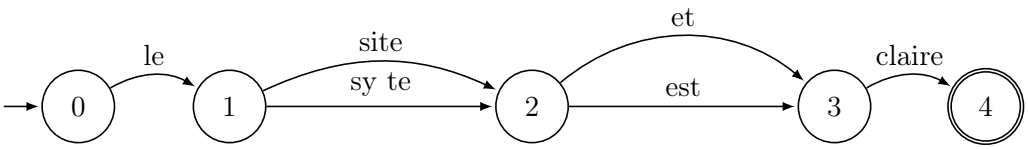

FIGURE 6.6 – Analyse de la phrase « *le syte et claire* » avant la désambiguïsation

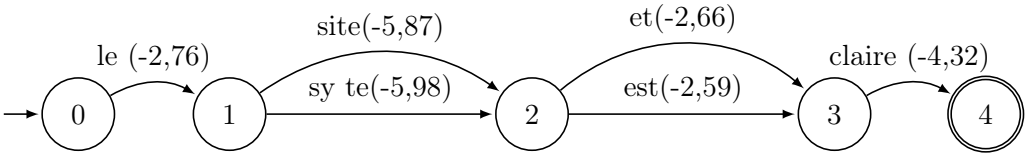

FIGURE 6.7 – Pondération de l'analyse obtenue pour la phrase « *le syte et claire* » grâce au modèle de langue "k"-gramme entrainé sur "n"% de la Wikipédia

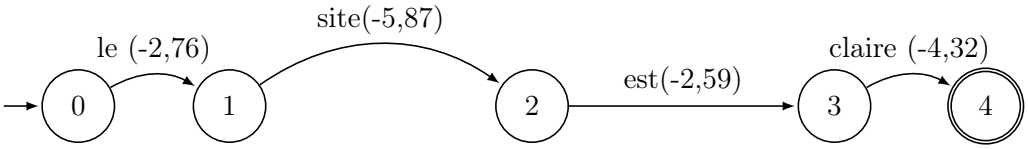

FIGURE 6.8 – Sélection de la meilleure analyse de la phrase « *le syte et claire* »

L'analyse ainsi obtenue est une normalisation correcte de la phrase de départ (la seule forme dont l'orthographe est incorrecte est « *claire* », mais la normalisation se faire à la flexion près).

6.3 Évaluation

6.3.1 Corpus d'évaluation

Notre chaîne de traitement vise à améliorer les résultats qu'obtiennent les outils de viavoo sur les textes auxquels l'entreprise est confrontée. Nous avons donc fait le choix de nous évaluer uniquement sur de tels textes. Pour ce faire, nous utiliserons trois corpus qui ont tous été manuellement annotés.

1. ***Corp1*** *:* Le premier corpus utilisé est un corpus composé de messages extraits aléatoirement des données viavoo. Nous avons annoté ce dernier au début de notre travail. Il est composé de 13 896 tokens, dont 250 altérations non flexionnelles.

2. ***Corp2*** *:* Le second corpus est celui présenté en section 4.2.9 employé afin d'évaluer notre système de prétraitement. Il est ainsi constitué de messages provenant de canaux distincts. Il est composé de :

   - messages extraits de site d'avis contenant 2016 tokens dont 28 altérations non flexionnelles,
   - e-mails représentant près de 2170 tokens, dont 18 altérations non flexionnelles,
   - messages provenant de forums contenant 2013 tokens dont 22 altérations non flexionnelles,
   - messages extraits de formulaires qui ont été ajoutés par la suite contenant 2046 tokens dont 38 altérations non flexionnelles,
   - enquêtes de satisfaction d'avis contenant 2010 tokens, dont 59 altérations non flexionnelles,
   - conversations tchats contenant 1914 tokens dont 47 altérations non flexionnelles.

   Au total, ce corpus contient 12200 tokens, dont un peu plus de 200 altérations non flexionnelles.

3. ***Corp3*** *:* Enfin, un troisième corpus a été ajouté. Ce dernier est composé d'une centaine de phrases (1749 tokens) sélectionnées manuellement qui ont pour caractéristiques d'être particulièrement mal orthographiées. Ce corpus est quant à lui doté de près de 200 altérations non flexionnelles.

6.3.2 Borne supérieure de notre système

Avant d'évaluer notre système, nous avons calculé les scores maximaux que pourrait obtenir notre module de désambiguïsation s'il ne se trompait jamais, étant donné les candidats entre lesquels il doit choisir. Cette première évaluation est particulièrement intéressante pour deux raisons. Tout d'abord elle nous permet d'obtenir des scores de référence auxquels nous pourrons nous comparer par la suite. De plus, cela nous permet d'évaluer notre système de génération de candidats (tout type d'altération confondu) avant que notre système de désambiguïsation tente de ne conserver que les plus pertinents.

N'ayant pas déterminé à ce stade le seuil de fréquence minimal conservé pour notre module d'homophonie, nous avons pris en compte chaque cas de figure possible. Les précisions, rappels et f-mesures obtenus suite à ces diverses évaluations figurent respectivement dans les tables 6.5, 6.6 et 6.7.

De manière générale, on peut constater que plus le seuil d'homophonie est élevé, plus nos résultats décroissent. Cela s'explique par le fait que ce seuil, en augmen-

Corpus testé	Seuil homophonie 0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,904	0,898	0,885	0,893	0,891	0,845
Corp2	0,973	0,972	0,970	0,970	0,969	0,959
Corp3	0,869	0,867	0,870	0,861	0,860	0,860

Table 6.5 – Précision maximale théorique pouvant être obtenue par notre système de désambiguïsation

Corpus testé	Seuil homophonie 0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,703	0,656	0,613	0,621	0,609	0,406
Corp2	0,834	0,797	0,761	0,744	0,734	0,548
Corp3	0,679	0,665	0,651	0,633	0,628	0,600

Table 6.6 – Rappel maximal théorique pouvant être obtenu par notre système de désambiguïsation

Corpus testé	Seuil homophonie 0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,791	0,758	0,724	0,733	0,723	0,548
Corp2	0,898	0,876	0,853	0,842	0,835	0,697
Corp3	0,762	0,753	0,745	0,730	0,726	0,707

Table 6.7 – F-Mesure maximale théorique pouvant être obtenue par notre système de désambiguïsation

tant, restreint le nombre de fautes grammaticales détectées par notre système et pouvant ainsi être corrigées.

En ce qui concerne la précision obtenue (cf. table 6.5), on constate que nos scores varient en fonction du contenu de chacun de nos corpus. Notre système a principalement failli sur :

- des noms propres de sociétés ou de produits qui n'étaient pas présents dans notre lexique de référence (ex : *Sephora* ou *Nespresso*),
- des abréviations spécifiques (ex : *fdp* → *frais de port*) ou des réductions non connues et traitées par notre système (ex : *K*→*que*)
- des erreurs lexicales particulièrement complexes pour notre système (ex : *esque* → *est-ce que*, *paimen*→*paiement* ou *fr* qui dans un même corpus aurait dû être réécrit une fois *faire* et autre fois *français*),
- des tokens mêlant plusieurs procédés d'altération (ex : *rapel* → *rappelle* qui correspond à la fois à une faute grammaticale et lexicale ou *nee*→*n'est* qui correspond à une altération produite sur l'agglutination de deux tokens)
- notre système ne traitant pas l'ajout d'espace au sein d'un même token il n'a pas su normaliser des altérations telles que *cordial ement*.

Notre rappel (cf. table 6.6) est dû, quant à lui, à des raisons similaires. Il s'explique notamment par la présence de tokens inconnus de notre lexique qui ont été trop altérés pour que notre système parvienne à proposer un candidat de normalisation (ex : *comtante* → *contente*, *aispere* → *espère*[12] ou *sisthematiquement* → *systématiquement*). On note, par ailleurs, la présence de quelques fautes grammaticales qui, non détectées, n'ont pas pu se voir attribuer des candidats de normalisation. Ces dernières peuvent être provoquées par une proximité clavier (ex : *soir* → *soit* ou *causse* → *cause*) et ne sont donc pas traitées par notre système de génération de candidats. Elles peuvent aussi concerner une erreur de proximité phonétique qui n'aura alors pas été détectée car sa fréquence aura été en dessous notre seuil défini pour la sélection des candidats d'homophonie (ex : *septique* → *sceptique* dont la fréquence est respectivement de 0,39 et 0,47).

6.3.3 Évaluation de la chaîne entière

Notre système de désambiguïsation a été évalué en faisant varier plusieurs critères. En effet, nous l'avons testé en appliquant différents modèles de langue et en proposant différents seuils de fréquence afin de limiter la détection des homophones. La totalité des résultats obtenus figure en annexe B.

12. Ce type d'altérations pourrait être détecté en passant par une étape de phonétisation des inconnus. Une extension naturelle de notre travail pourrait ainsi être l'entraînement d'un phonétiseur afin de pouvoir prédire la phonétique d'un token inconnu et rechercher ses homophones.

6.3.3.1 Sélection du modèle de langue utilisé

Afin de réaliser notre évaluation, nous nous sommes appuyée sur différents modèles de langue. Ces modèles se distinguent les uns des autres par la quantité de texte qui a servi à leur entraînement et par la taille des n-grammes appris au sein de ce corpus d'apprentissage. Nous avons fait varier la taille du corpus d'entraînement en conservant soit la Wikipédia entière, soit sa moitié, soit son tiers, soit son quart ou soit un sixième de cette dernière. La taille des n-grammes conservés fluctue quant à elle entre 2 et 4 grammes. Cela nous a permis de mettre en place 15 modèles de langue. Parmi ces derniers, nous n'avons conservé pour des raisons techniques que ceux qui faisaient moins de 4 gigaoctets, soit 11 d'entre eux, comme le montre le schéma 6.8.

Corpus	2-gramme	3-gramme	4-gramme
100% Wiki	✓	×	×
50% Wiki	✓	✓	×
33% Wiki	✓	✓	×
25% Wiki	✓	✓	✓
17% Wiki	✓	✓	✓

Table 6.8 – Modèles de langues conservés

Après avoir testé tous ces modèles sur nos trois corpus, en faisant varier les différents seuils minimaux de fréquence pour le module des homophones, nous avons constaté qu'ils donnaient des résultats très proches (cf. la totalité de ces résultats en annexe B). En effet, aucun modèle ne se démarque réellement des autres. C'est pourquoi nous avons choisi d'utiliser arbitrairement le modèle trigramme entraîné sur la moitié de la Wikipédia. Ce choix a notamment été motivé par la qualité de ses résultats qui font très régulièrement partie des meilleurs.

6.3.3.2 Sélection du seuil minimum de fréquence utilisé pour la détection des fautes d'homophonie

Notre module d'homophonie a pour but de proposer des candidats de normalisation pour tous les tokens connus de notre lexique qui dépassent un certain seuil de fréquence normalisée. Ce score est compris entre 0 et 1. Plus ce dernier est élevé plus le token concerné sera considéré comme fréquent. Afin d'évaluer ce module, nous avons paramétré notre module en faisant varier ces seuils en nous limitant aux valeurs suivantes : 0,60, 0,65, 0,70, 0,75, 0,80, 0,85 et 0,90.

6.3.3.2.1 Résultats obtenus en ne conservant que la meilleure analyse :

Les résultats obtenus par notre système complet en ne conservant que la meilleure

analyse proposée par notre modèle de langue sont illustrés dans les tables 6.9, 6.10 et 6.11.

Corpus testé	Seuil fixé pour module d'homophonie 0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,764	0,756	0,761	0,749	0,747	0,660
Corp2	0,912	0,905	0,919	0,901	0,900	0,874
Corp3	0,740	0,740	0,741	0,732	0,735	0,736

Table 6.9 – Précision de la meilleure normalisation produite par notre système complet en fonction du seuil de fréquence fixé pour le module d'homophonie

Corpus testé	Seuil fixé pour module d'homophonie 0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,601	0,571	0,547	0,545	0,527	0,341
Corp2	0,754	0,728	0,718	0,698	0,688	0,505
Corp3	0,604	0,593	0,583	0,569	0,565	0,544

Table 6.10 – Rappel de la meilleure normalisation produite par notre système complet en fonction du seuil de fréquence fixé pour le module d'homophonie

Corpus testé	Seuil fixé pour module d'homophonie 0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,693	0,651	0,636	0,631	0,617	0,450
Corp2	0,826	0,807	0,806	0,783	0,780	0,640
Corp3	0,665	0,658	0,653	0,640	0,639	0,626

Table 6.11 – F-mesure de la meilleure normalisation produite par notre système complet en fonction du seuil de fréquence fixé pour le module d'homophonie

Tout d'abord, nous pouvons noter la nette progression de nos résultats au fur et à mesure que l'on fait croître le seuil minimal de fréquence du module de production de candidats de normalisation par homonymie. L'évolution de ces scores est attendue puisque moins ce seuil est élevé, plus notre module considère comme potentiellement altérés les tokens existants dans nos lexiques. Nous aurions pu penser qu'une telle méthode ajouterait beaucoup de bruit dans notre système. Toutefois, on note que notre précision s'améliore lorsque l'on fait baisser notre seuil de 90 à 50 et que notre rappel permet la détection et la normalisation de 25% d'altérations supplémentaires pour les corpus *Corp1* et *Corp2*. Le corpus *Corp3* étant un corpus très bruité, il contient en grande majorité des tokens inconnus très altérés ce qui explique que son rappel ne varie quant à lui que de

5% environ. Ces résultats sont ici satisfaisants puisqu'initialement nous voulions favoriser la précision au rappel [13].

On constate par ailleurs qu'en conservant, à l'aide de notre modèle de langue, un seul chemin dans les DAG produits par notre chaîne de traitement, nos résultats sont à environ à 10% en dessous du score optimal pouvant être obtenu à partir des candidats précédemment produits (cf. section 6.3.2).

6.3.3.2.2 Résultats obtenus en ne conservant que les trois meilleures analyses : Les résultats obtenus par notre système en ne conservant que les trois meilleures analyses proposées par notre modèle de langue sont illustrés dans les tables 6.12, 6.13 et 6.14.

Corpus testé	Seuil fixé pour module d'homophonie					
	0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,805	0,806	0,805	0,805	0,804	0,737
Corp2	0,920	0,922	0,941	0,931	0,931	0,909
Corp3	0,787	0,788	0,802	0,791	0,795	0,799

Table 6.12 – Précision des trois meilleures normalisations produites par notre système complet en fonction du seuil de fréquence fixé pour le module d'homophonie

Corpus testé	Seuil fixé pour module d'homophonie					
	0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,644	0,618	0,598	0,591	0,580	0,386
Corp2	0,771	0,751	0,738	0,721	0,714	0,532
Corp3	0,634	0,623	0,623	0,600	0,595	0,572

Table 6.13 – Rappel des trois meilleures normalisations produites par notre système complet en fonction du seuil de fréquence fixé pour le module d'homophonie

En ne conservant que les trois meilleures analyses proposées par notre modèle de désambiguïsation, on peut faire le même constat que précédemment : nos résultats s'améliorent dès lors que notre seuil diminue. En outre, on peut noter que ces derniers sont cette fois-ci beaucoup plus proches des optimums pouvant être obtenus par notre système de désambiguïsation. On conserve toutefois un écart allant environ de 3% à 6% avec les scores maximaux que notre système de désambiguïsation peut avoir.

13. Pour répondre parfaitement à ce besoin, il faudrait pouvoir mesurer combien de sur correction ou de détection d'informations erronées notre système induit sur la solution proposer par viavoo.

Corpus testé	Seuil fixé pour module d'homophonie 0,60	0,65	0,70	0,75	0,80	0,90
Corp1	0,716	0,700	0,686	0,682	0,674	0,507
Corp2	0,839	0,828	0,827	0,813	0,808	0,671
Corp3	0,702	0,696	0,701	0,682	0,681	0,667

Table 6.14 – F-mesure des trois meilleures normalisations produites par notre système complet en fonction du seuil de fréquence fixé pour le module d'homophonie

6.3.3.2.3 Choix du seuil minimal de fréquence choisi pour le module de détection d'homophonie On pourrait supposer qu'en diminuant encore ce seuil, notre système obtiendrait de meilleurs résultats. Néanmoins plus ce seuil est faible, plus nos DAG deviennent ambiguës et sont complexes à analyser pour notre système de désambiguïsation. Nous souhaitons privilégier l'efficacité de notre système favorisant la précision au rappel. Pour deux de nos trois corpus, c'est le seuil à 70% qui obtient les meilleurs scores de précision. C'est pourquoi, nous avons choisi de fixer notre paramétrage à ce seuil donné.

6.3.3.2.4 Analyse manuelle des erreurs de normalisation réalisées sur les altérations Comme expliqué ci-dessus, la précision est primordiale pour nous. C'est pourquoi nous avons analysé les cas où notre système proposait des normalisations fautives aux tokens altérés. Cette observation a été faite sur tous nos corpus d'évaluation confondus. Nous constatons ainsi que notre système suggère des candidats erronés principalement sur les cas suivants :

- Près d'un cinquième des altérations sur lesquelles notre système se trompe sont des tokens qui ont subi un processus de réduction. Il peut ainsi s'agir d'abréviations (ex. :*maxi* → *maximum, tél*→*téléphonique/téléphone* ou *déo* → *déodorant*), de squelettes consonantiques (ex. : *cc* → *coucou* ou *cdt* → *cordialement*), de siglétisation propre à un domaine (ex. : *fdt* → *frais de port*), de réductions phonétiques (ex. : *g* → *j'ai* ou encore Q → cul) ou encore de noms d'enseignes abrégés (ex. : *PIX* →*Pixmania*). Notre système ne traitant que partiellement ces différents cas de figure à l'aide d'un lexique de substitution, ce constat n'est pas étonnant. Notons par ailleurs que certaines altérations peuvent correspondre à des réductions elles-mêmes altérées (ex : *anive* → *anniversaire*).

- Un autre cinquième de nos erreurs de normalisation se porte sur les altérations phonétiques. Bien que notre système sache traiter une altération phonétique simple, elle se révèle impuissante lorsque le token altéré est trop éloigné du token initial (ex. : *comtante* → *contente*, *debarah* → *débarras*, *erbague* → *airbag* ou encore *fahy* → *failli*). Une solution simple qui permet-

trait de pallier ce problème serait d'entraîner un phonétiseur afin de suggérer les homophones de ces tokens inconnus comme candidats de normalisation.

- Un troisième cinquième de nos erreurs de normalisation concerne les tokens inconnus dont l'altération est trop complexe pour notre système. En effet, ces cas concernent généralement soit la concaténation de plusieurs altérations séparées dans un même token (ex. : *paimen* → *paiements* ou *cotran* → *contrat*), soit des altérations qui modifient plusieurs lettres juxtaposées l'une à l'autre (ex. : *conrtable* → *confortable*). Le premier cas d'altération pourrait être normalisé en autorisant l'application de plusieurs règles de normalisation sur un même token inconnu.

- On compte quelques cas d'erreurs de suppression d'apostrophes non gérés actuellement par notre module (ex. : *senlève* → *s'enlève* ou *tai* → *t'ai*). Ces dernières pourraient être réduites en élargissant nos grammaires locales

- Par ailleurs trois cas de fautes grammaticales ont été normalisées par le module d'homophonie alors qu'il s'agissait de faute due à de la paronymie (ex. : *on* → *un* ou *de* → *des*). La présence d'erreurs de ce type est normale dans la mesure où nous ne les traitons pas.

- Enfin, précisons que notre système tente, sans succès, de normaliser les noms d'enseignes et de produits qui lui sont inconnus. Il propose par exemple de normaliser *smiles* en *miles* alors que le token attendu était *S'Miles*. Pour éviter ces fautes, une possibilité serait d'ajouter un lexique spécifique.

En ce qui concerne les mauvais candidats de corrections proposés restants, nous relevons enfin trois cas de figure : (1) le cas où la normalisation attendue est inconnue de notre lexique (ex. : l'altération *pshiiiiit* n'est ainsi pas normalisée en *pshit*), (2) les cas où notre système de normalisation pour les altérations inconnues n'avait pas de règles de normalisation adaptées de par leurs restrictions du contexte (ex. : *maudique* → *modique*) et, enfin (3) les cas où notre système de désambiguïsation ne choisit pas le candidat valide parmi ceux proposés [14].

6.3.3.3 Pertinence de la correction proposée par SxPipe

Nous nous sommes interrogée sur la pertinence d'avoir intégré le correcteur de SxPipe à notre système. Une évaluation a donc été réalisée sans les candidats proposés par ce correcteur avec le modèle de langue trigramme entraîné sur la moitié de la Wikipédia et en fixant notre seuil minimum de fréquence pour l'homophonie à 70. Il apparaît que nos résultats pour nos corpus *Corp1* et *Corp2* ne varient que très peu (notre système perd approximativement 1% de précision, de rappel et de F-Mesure). Cela s'explique par le fait que les altérations laissées

14. Cette erreur de sélection peut être due à un token peu, voire pas, utilisé dans le corpus d'entraînement, ou encore, au contexte entourant ce token.

comme inconnues par notre système, dans ces corpus, sont peu nombreuses et correspondent soit à du langage SMS, par conséquent trop éloignées de leurs tokens d'origine pour être normalisées sans l'aide d'un lexique de substitution, soit trop altérées pour se voir attribuer un candidat de normalisation valable. Les autres altérations inconnues de nos lexiques ayant été traitées par nos modules, le correcteur de SxPipe n'y a plus accès. L'intérêt d'ajouter un système comme celui-ci est toutefois plus visible sur un corpus comme *Corp3*. En effet, ce corpus particulièrement bruité contient principalement des altérations réalisées sur des inconnus. Le système de correction proposé par SxPipe peut être certes plus bruyant que le nôtre, mais parvient ainsi à rattraper de nombreuses erreurs. Ainsi sur un corpus tel que *Corp3*, notre système perdrait 2% de précision, 5% de rappel et ainsi près de 4% de f-mesure s'il n'appliquait pas ce correcteur.

6.3.3.4 Apport des informations contextuelles

Enfin de mesurer l'apport des informations contextuelles sur nos résultats, nous avons évalué notre système complet sans prendre en compte le contexte dans lequel apparaît chaque token susceptible d'être altéré.

Une première approche possible est de nous appuyer uniquement sur la fréquence de chaque candidat de normalisation pris isolément dans la langue pour déterminer lequel nous voulons conserver. Cela revient alors à prendre uniquement des unigrammes en compte pour réaliser notre désambiguïsation. Bien qu'une telle méthode permette la normalisation de près de la moitié des altérations annotées [15], elle produit beaucoup de bruit et normalise une grande partie des tokens non altérés. En effet, le module de détection des fautes réalisées par homophonie propose des candidats de normalisation pour tous les tokens fréquents dans la langue. Ainsi, en ne conservant que le token le plus fréquent nous remplaçons chaque token étant au-dessus d'un certain seuil de fréquence par son homophone le plus fréquent dans la Wikipédia (qui est notre corpus d'entraînement). C'est pour cette raison que nous avons éliminé la possibilité de nous appuyer sur la fréquence d'un token sans prendre en compte son contexte. La prise en compte du contexte est donc bien essentielle pour désambiguïser notre système complet.

Une autre approche, plus pertinente, serait de sélectionner nos candidats de normalisation en deux temps. Ainsi, dans un premier temps, notre système de normalisation par règles induites par analogie (cf. chapitre 5) sélectionnerait à l'aide de son système de pondération le candidat de normalisation le plus probable étant donné l'altération réalisée. Puis, dans un second temps, notre système de désambiguïsation contextuel serait appliqué pour résoudre les ambiguïtés restantes. Les résultats obtenus pour cette approche figurent dans la table 6.15.

15. Cette estimation correspond à la moyenne des rappels obtenus si on prend en compte les différents seuils minimums de fréquence utilisés pour la détection des fautes d'homophonie.

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
Corp1	0.743	0.549	0,631	0.786	0.579	0.667
Corp2	0.887	0.703	0.784	0.896	0.717	0.797
Corp3	0.632	0.520	0.571	0.676	0.541	0.601

Table 6.15 – Résultats proposé par notre système lorsque la sélection des candidats de normalisation pour des altérations inconnues est faites au préalable

Si l'on se concentre sur les résultats obtenus par les corpus *Corp1* et *Corp2*, on constate qu'ils sont très semblables à ceux précédemment obtenus avec la prise en compte du contexte. Ce constat est satisfaisant dans la mesure où il démontre que notre système de pondération proposé au chapitre 5 est suffisamment bon pour obtenir des résultats équivalents à un modèle de langue. Cela illustre par ailleurs le fait qu'il n'est pas systématiquement nécessaire d'avoir recours au contexte pour normaliser des tokens inconnus altérés. En ce qui concerne le corpus *Corp3*, on peut noter que nos résultats sont cette fois-ci nettement moins bons. On perd ainsi 11% de précision et 6% de rappel. Cette différence de résultats avec les premiers corpus peut s'expliquer par leur différence de contenu, ce dernier corpus étant nettement moins bien orthographié que les précédents. Notre système de pondération n'est plus suffisant. Il est ainsi préférable de rester avec notre modèle de langue qui obtient des résultat soit équivalent soit meilleurs selon la qualité orthographique du texte à traiter.

6.4 Conclusion

Les résultats obtenus lors du calcul de la borne supérieure de notre système illustrent les capacités de notre système dans le cas où l'on choisirait de conserver tous les candidats suggérés par notre chaîne de traitement. La précision obtenue montre ainsi la qualité des candidats de normalisation suggérés pour une altération donnée. Par ailleurs, son rappel varie entre 61% et 76% de bonne normalisation en fonction du corpus évalué, ce qui est satisfaisant. Les erreurs de correction effectuées sont notamment dues à la présence de noms propres inconnus de nos lexiques, de tokens peu courants propres au langage sms et, à la présence de tokens inconnus altérés contenant généralement, non pas une erreur en leur sein, mais deux ou plus. Ces deux premiers cas pourraient être améliorés en complétant nos lexiques. Le dernier cas quant à lui pourrait être résolu en limitant à notre système de correction par analogie à non pas une erreur, mais deux ou trois. Toutefois, cette manœuvre bien que possible risquerait de ralentir notre système et de le complexifier. Nous préférons actuellement le conserver comme tel. Enfin, certains tokens altérés nécessiteraient l'application de notre système de correction par analogie suivi de notre module de détection des fautes de proximité phoné-

tique (ex : *rapel* → *rappel* → *rappelle*). Toutefois ces cas sont trop rares dans nos corpus pour être pris en compte.

La sélection du candidat le plus probable parmi ceux proposés est ensuite faite par notre module de désambiguïsation en s'appuyant sur le contexte dans lequel est réalisée chaque altération. Bien que notre système obtienne de moins bons scores que ceux obtenu pour le calcul de la borne supérieure de notre système, ces derniers restent satisfaisants et peu bruyants.

Suite à ce chapitre, nous obtenons une chaîne de normalisation complète capable de proposer des candidats de normalisation aussi bien pour les tokens altérés connus qu'inconnus. Les DAGs proposés en figure 6.9 et 6.10 ainsi qu'en figure 6.11 et 6.12 illustrent les analyses qui seront proposées par notre système avant et après l'étape de désambiguïsation pour les phrases « *ESPACE ABITACLE REDUIT PAR RAPPORT A LA 307CC* » et « *c'est miux d'aller en magazin* ».

Quelques exemples de phrases bruitées et normalisées par notre système figurent par ailleurs en annexe C. Ces normalisations exemplifient uniquement le cas où nous ne conserverions que le chemin ayant obtenu le meilleur score.

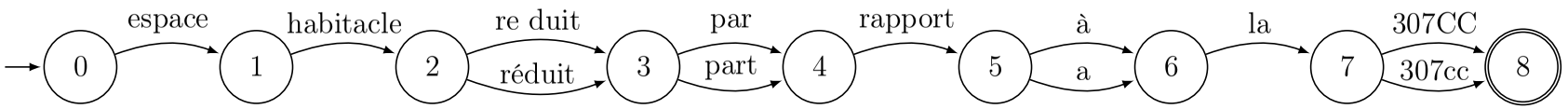

FIGURE 6.9 – DAG de la phrase « *ESPACE ABITACLE REDUIT PAR RAPPORT A LA 307CC* » avant désambiguïsation

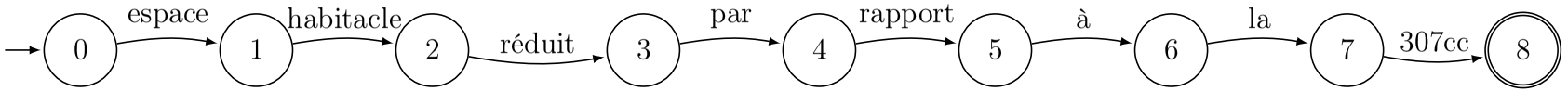

FIGURE 6.10 – DAG de la phrase « *ESPACE ABITACLE REDUIT PAR RAPPORT A LA 307CC* » après désambiguïsation

---

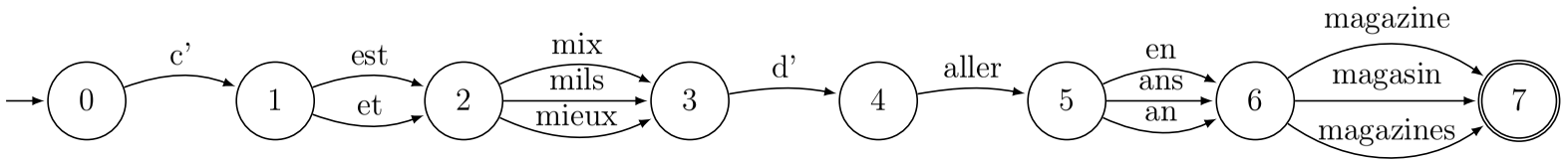

FIGURE 6.11 – DAG de la phrase « *c’est miux d’aller en magazin* » avant désambiguïsation

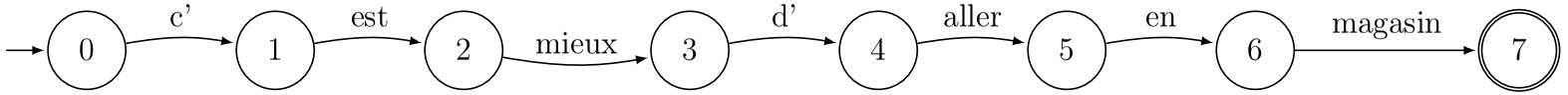

FIGURE 6.12 – DAG de la phrase « *c’est miux d’aller en magazin* » après désambiguïsation

CHAPITRE 7

Conclusion générale

7.1 Bilan

La présence de formes altérées au sein d'un texte peut être particulièrement gênante pour de nombreux systèmes en traitement automatique des langues. En effet, ces derniers tiennent souvent pour acquise la qualité orthographique du texte à analyser. Si l'on se cantonne à des textes journalistiques ou encyclopédiques, cela n'a rien de problématique. Ces textes sont dans leur grande majorité peu altérés et syntaxiquement corrects. Si toutefois nous voulons traiter des données réelles, produites par l'utilisateur, la tâche se révèle plus ardue. Les altérations sont fréquentes dans ce type de textes et peuvent induire en erreur de nombreux systèmes. Les travaux détaillés dans cette thèse proposent de pallier ce problème pour les outils qui ne sont pas à la flexion près. La solution suggérée ici est de passer par un système de normalisation orthographique qui aurait pour objectif de détecter la majorité des altérations contenues dans un texte et de les normaliser, c'est-à-dire de les corriger à la flexion près.

À travers l'état de l'art présenté dans la première partie de cette thèse, nous proposons des éléments permettant de caractériser plus précisément les différents types de tokens constituants la langue. Nous suggérons ainsi de distinguer ces derniers en s'appuyant sur deux critères : (1) Le token est-il connu de notre lexique de référence ? (2) Le token a-t-il été altéré ? Cela peut donner lieu à quatre cas de figure que nous avons tenté de prendre en compte. Si le token est non altéré et connu du lexique, il n'est pas dérangeant pour une analyse automatique de la langue. A contrario, les trois cas restants le sont. En effet, si un token non altéré

n'est pas connu du lexique, un système aura toutes les raisons de le croire mal orthographié. Nous avons recensé les divers tokens pouvant être contenus dans cette catégorie et les différents procédés de détection pouvant être mis en place pour pallier leur absence dans le lexique. Enfin, nous nous sommes concentrée sur les formes altérées, connues et inconnues du lexique, afin préciser leurs mécanismes de création. Puis nous avons proposé un éventail des différentes techniques existantes pour détecter et traiter ces types de tokens particuliers.

Détailler ces trois derniers cas de figure nous a permis d'affiner notre approche et de décomposer notre problématique en plusieurs sous-parties. Cela nous a poussée à mettre en place un système modulaire, qui peut de la sorte être aisément paramétrable en fonction de l'utilisation que l'on souhaite en faire. La figure 7.1 offre une vision schématisée de notre chaîne de traitement finale, qui propose l'application de plusieurs techniques afin de procéder à la normalisation totale d'un texte. La détection des tokens inconnus non altérés (première case du schéma) repose principalement sur des méthodes employant des grammaires locales, des techniques de classification et sur de l'apprentissage par analogie. La création de candidats de normalisation pour les inconnus altérés (seconde et troisième cases du schéma) utilise une approche de génération de règles de correction pondérées apprises par analogie couplée à des informations sur la fréquence des tokens d'une langue [1]. La génération de candidats de normalisation pour les tokens altérés figurants tout de même dans notre lexique de référence (quatrième case du schéma) est réalisée à l'aide d'informations lexicales associées à des informations de fréquence des tokens de la langue. Enfin la désambiguïsation des candidats proposés est, quant à elle, effectuée à l'aide d'un modèle de langue probabiliste.

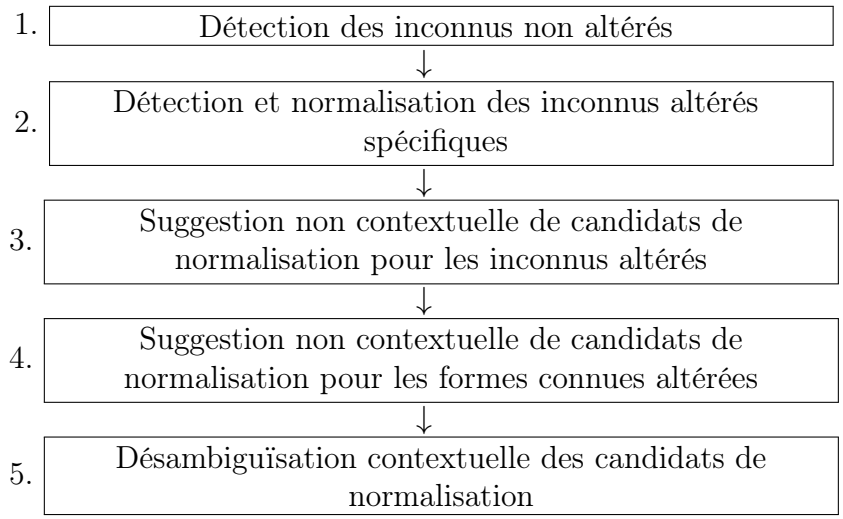

FIGURE 7.1 – Modules composants notre système de normalisation

1. Notons toutefois que pour les altérations spécifiques nous nous appuyons uniquement sur des grammaires locales.

L'assemblage de ces modules nous permet d'obtenir un système assez complet qui peut être accessible via SxPipe. Chaque module est ainsi compatible avec la chaîne de traitement et partage la même licence que lui, à savoir la licence CeCiLL-C. Notre système modulaire est donc accessible librement.

Initialement, ce système a été implémenté afin de répondre à un besoin émis par l'entreprise viavoo. En effet, lorsque la solution proposée par cette dernière est confrontée à des verbatim dont les informations principales sont altérées, elle ne peut les détecter. Notre système a donc pour principal objectif de réduire le nombre de silences en détectant et normalisant ces altérations. À l'heure actuelle, notre chaîne de normalisation orthographique n'a pas encore été intégrée à la solution proposée par viavoo. Ainsi, malgré ses résultats plutôt satisfaisants, nous ne pouvons pas émettre de pronostics quant à l'amélioration qu'elle apportera à la solution proposée par l'entreprise. Son intégration ainsi qu'une évaluation à grande échelle de son impact sur la solution viavoo devraient être réalisées prochainement. Notre chaîne a toutefois été évaluée sur de nombreuses ressources provenant de multiples canaux et d'entreprises différentes. Les résultats obtenus et présentés tout au long de cette thèse sont donc réellement représentatifs des données traitées par viavoo.

Une limite possible de ce travail pourrait être son caractère endogène dans la mesure où il a été mis en place pour une tâche spécifique et évalué en conséquence. Il est donc important de préciser que ce travail est tout d'abord difficilement comparable avec les autres travaux du domaine dans la mesure où il veut répondre à une tâche bien précise, la normalisation des altérations à la flexion près, et a été mis en place pour un type de corpus spécifiques, les textes bruités. Nous pourrions donc envisager deux types de comparaisons :

1. Comparer les travaux existants sur notre tâche en évaluant leurs répercussions et leurs apports dans la solution proposée par viavoo. Cette option est toutefois difficilement envisageable étant donné qu'actuellement les outils les plus complets [2] disponibles sur le marché sont des outils d'aide à la rédaction qui ont un format de sortie difficilement intégrable dans l'état à la solution proposée par viavoo et qui sont peu adaptés aux textes bruités.

2. Se comparer aux autres systèmes existants en ajustant notre système afin qu'il puisse traiter le texte avec le même objectif et les mêmes données que les autres. Notre système pouvant facilement être adaptable pour une tâche de correction dans la mesure où il se trompe peu quant à la flexion du candidat proposé, cette option est plus aisément réalisable. Toutefois, bien que ce travail supplémentaire soit réellement pertinent, il n'est pas intéressant d'un point de vue opérationnel pour une entreprise telle que viavoo et risque d'être trop coûteux en temps.

2. Nous entendons par *outil complet*, les outils qui traitent à la fois les fautes lexicales et grammaticales.

Bien que cette chaîne de traitement ait été réalisée au sein d'une entreprise et pour cette dernière, notons toutefois qu'elle est assez générique pour être appliquée à un large panel de textes différents [3].

7.2 Perspectives

Nous donnons ici quelques pistes pour aller plus loin dans la continuité de ces travaux ou sur des problématiques connexes liées à la normalisation de texte bruité. Ces perspectives touchent quatre points distincts de cette thèse.

Dans un premier temps, la génération de candidats réalisée par notre système pourrait être améliorée. En effet, notre évaluation générale a mis en avant la présence plus nombreuse qu'attendu d'altérations réalisées de manière phonétique. La manière la plus simple serait donc de passer, en plus de notre approche par règles, par une approche par phonétisation. Une solution serait donc d'entraîner automatiquement un phonétiseur afin de phonétiser tous les inconnus de notre lexique (non détectés comme création, emprunt ou autre) et suggérer ainsi tous les homophones d'un token inconnu donné. Cela nous permettrait ainsi de rattraper toutes les altérations du même type que *aispaire* (*espère*). Le phonétiseur pourrait alors être appris sur des ressources librement disponible comme le Wiktionnaire par exemple qui contient pour chacune de ses entrées sa transcription phonétique. Cette dernière ressource existant dans le même format pour de nombreuses langues, cela favoriserait l'adaptation de notre système à d'autres langues européennes. Par ailleurs, notre système de normalisation pour les formes connues altérées ne permet actuellement que la normalisation des fautes grammaticales produite par homophonie. Celles produites par paronymie pourraient être par exemple corrigées en s'appuyant sur le même système de correction par règles que celui appliqué pour la normalisation des formes inconnues altérées.

Dans un second temps, c'est au niveau de la sélection du candidat de normalisation le plus pertinent que nous pourrions nous pencher. À l'heure actuelle, nous laissons notre système par désambiguïsation choisir seul quel candidat conserver. Une seconde piste possible serait de présélectionner les candidats de normalisation pour les formes inconnues altérées en nous appuyant sur leurs scores associés. Ces scores étant relativement fiables (si l'on se fie aux expériences que nous avons menées à ce niveau), il serait donc intéressant de constater l'apport du contexte par rapport des seuls scores de normalisation hors contexte. En outre, une troisième piste possible serait de nous appuyer sur les cooccurrences présentes dans le contexte en plus du contexte proche. De la sorte, le thème du message pourrait ainsi être ciblé et pourrait lui aussi influencer le choix d'un candidat. Ce der-

3. Il est à préciser que pour être réellement efficace sur des données plus bruitées que les nôtres (telles que des SMS par exemple) ce système devrait se voir ajouter un système de phonétisation robuste pour la normalisation des inconnus.

nier procédé serait toutefois à employer avec précaution, en ne l'utilisant que sur certaines catégories grammaticales porteuses de sens telles que les noms.

Dans un troisième temps, soulignons que notre chaîne sera appliquée sur des messages de provenances multiples. Il pourrait donc être intéressant de l'adapter dynamiquement au canal traité ou à l'entreprise concernée. En ce qui concerne le canal, nous pourrions ainsi autoriser des corrections plus ou moins coûteuses selon les cas. Cela permettrait qu'un mail se voie proposer des corrections plus légères qu'un message provenant d'un réseau social. De même, il serait tout à fait possible d'ajuster notre système en fonction de l'entreprise concernée par le message en question. En effet, d'une entreprise à l'autre, le vocabulaire utilisé diffère. Utiliser des lexiques spécialisés propres au domaine de compétence de l'entreprise et la probabilité qu'un token apparaisse dans ce même domaine pourrait ainsi être pertinent.

Dans un dernier temps, rappelons que notre chaîne de traitement a été implémentée dans le souci d'être indépendante de la langue. Bien qu'elle n'ait été actuellement évaluée que sur le français, elle peut être adaptée à d'autres langues à la morphologie majoritairement concaténative. Il est suffisant pour cela de disposer d'un certain nombre de ressources lexicales telles qu'un lexique de la langue, la liste de ses homophones et d'un corpus brut représentatif et d'un corpus de fautes annotées. L'apprentissage de nos règles de normalisation pour les formes inconnues non altérées étant fait automatiquement, cette normalisation pourrait parfaitement fonctionner pour d'autres langues, pour peu qu'il existe une base de fautes corrigées pour cette langue. En outre, cette base pourrait parfaitement être extraite de la Wikipédia correspondante, de la même façon qu'a été construit le corpus WiCoPaCo pour le français ou comme le propose Zesch (2012) pour l'anglais.

De manière plus générale, nous avons constaté qu'utiliser des techniques propres à l'analogie pour étudier et reproduire les régularités de la langue pouvait donner des résultats probants sur plusieurs langue. Nous avons notamment fait l'expérience en parallèle de ces travaux d'apprendre, de façon non supervisée, les règles morphologiques dérivationnelles du français, de l'anglais, de l'espagnol et de l'allemand, nous appuyant uniquement sur des lexiques flexionnels intensionnels (Baranes et Sagot, 2014) et par une approche par analogie très similaire à celle utilisée aux chapitres 4 et 5, respectivement pour détecter les néologismes et pour apprendre des règles de correction. Cette expérience, n'entrant pas directement dans le cadre de notre problématique, n'a pas été décrite dans cette thèse. Elle avait pour principal objectif de rattacher les mots dérivationnellement liés de chaque langue. Les résultats obtenus étant probants, cette expérience a permis la création de quatre ressources linguistiques contenant des liens morphologiques dérivationnels entre mots d'une même langue. Les bons résultats de ces expériences nous confortent sur la généricité de notre approche d'acquisition de règles par analogie et sur ses capacités à être adapté à d'autres langues à morphologie concaténatives, et ce,

pour différents types de tâches. De plus, la mise en œuvre de façon non-supervisée de ce système d'acquisition de relations dérivationnelles, à l'image de ce que nous avons mis en place pour la détection de néologismes (chapitre 4), permet de réfléchir à la possibilité de modifier notre méthodologie de génération de règles de correction (chapitre 5). Il serait possible de tenter d'extraire de telles règles directement à partir de corpus bruts (bruités) de façon là aussi non-supervisée, sans avoir besoin d'un corpus de fautes à utiliser comme corpus d'apprentissage. Cela garantirait par ailleurs le caractère peu dépendant de la langue de notre système. Il n'aurait alors besoin pour fonctionner que d'un corpus brut de la langue et d'un lexique de cette dernière.

Cette approche pourrait s'appliquer à d'autres tâches encore. Nous pourrions par exemple opter pour une étude non plus synchronique mais diachronique de la langue qui tenterait d'induire des règles de dérivation permettant de passer d'une langue peu dotée à une langue dotée (par exemple du français médiéval au français contemporain, ou du picard au français). Pour l'ancien français par exemple, une telle approche pourrait être réalisable avec un corpus numérisé de la langue peu un lexique du français actuel à partir desquels nous tenterions d'extraire des règles de vieillissement d'une même langue [4]. Les tokens liés par ces règles correspondraient ainsi à des cognats. Une telle approche pourrait par exemple apporter un complément à des travaux tels que ceux de Scherrer et Sagot (2014). Une autre idée serait de tenter de produire automatiquement des lexiques flexionnels et dérivationnels pour des langues concaténatives peu dotées en s'appuyant sur des corpus bruts. Nous pourrions ainsi en extraire les règles de flexion et de dérivation principales et reconstruire automatiquement un lexique flexionnel contenant les mots les plus employés de cette langue.

4. Ces règles pourraient aussi être perçues comme des règles de traduction si on considère que l'on traite deux langues plutôt qu'une seule qui aurait évolué.

Troisième partie

Annexes

ANNEXE A

Détails des résultats obtenus pour la normalisation des altérations inconnues

A.1 Résultats obtenus par les règles spécifiques seules

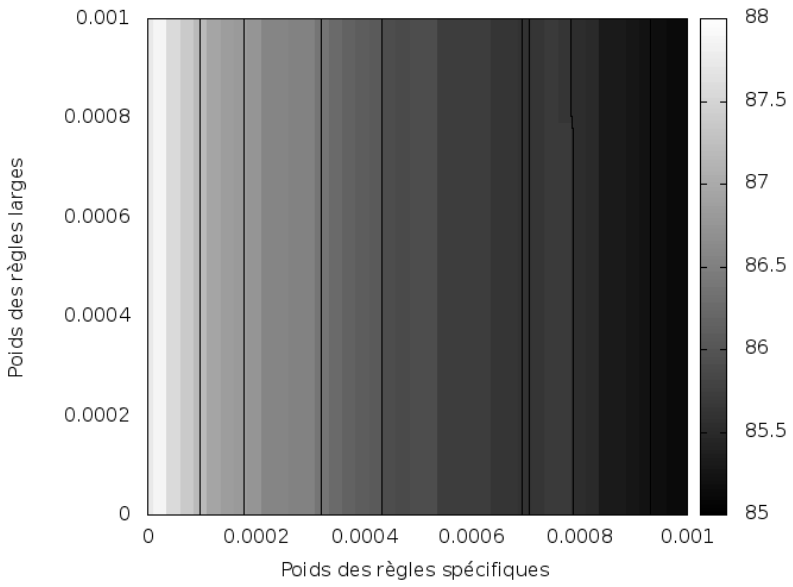

FIGURE A.1 – Précision pour la meilleure normalisation en prenant en compte uniquement les règles spécifiques

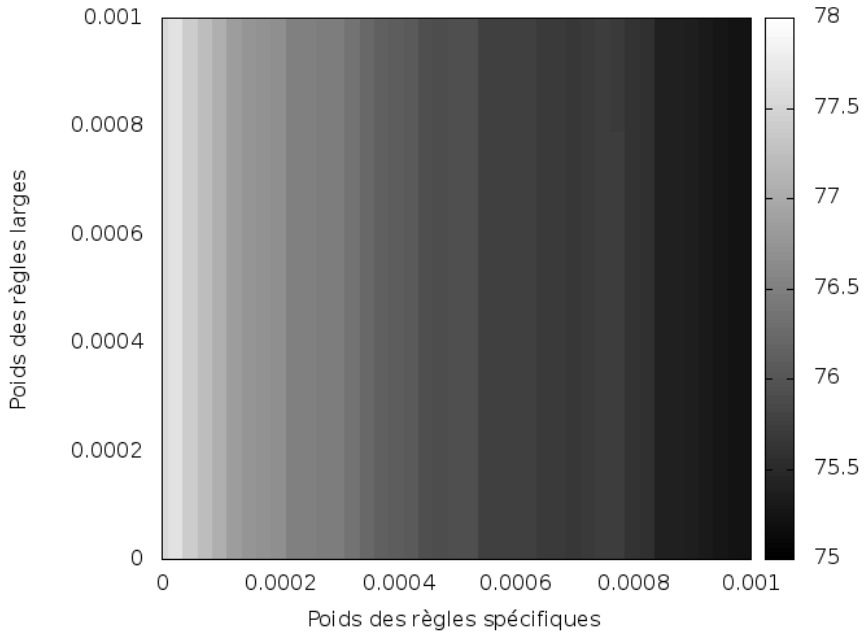

FIGURE A.2 – Rappel pour la meilleure normalisation en prenant en compte uniquement les règles spécifiques

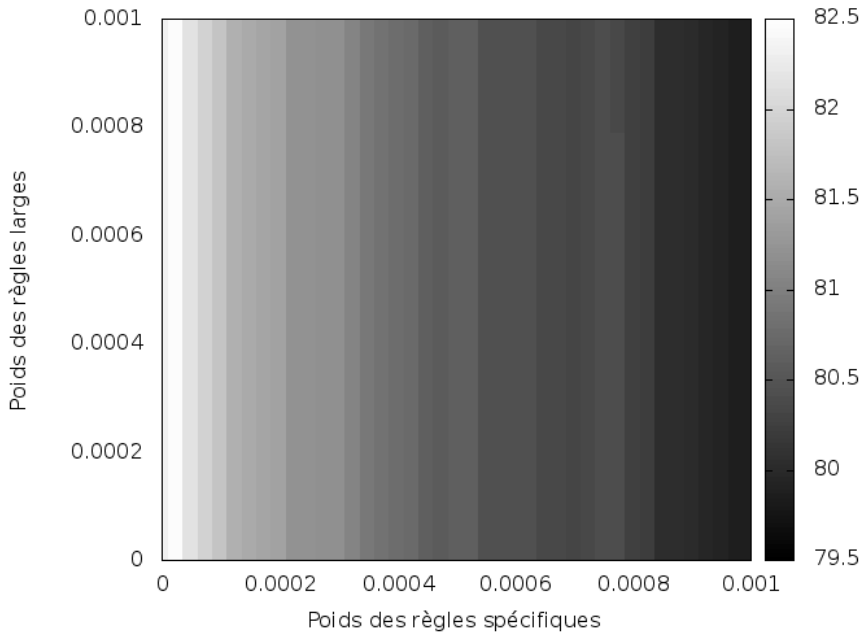

FIGURE A.3 – F-mesure pour la meilleure normalisation en prenant en compte uniquement les règles spécifiques

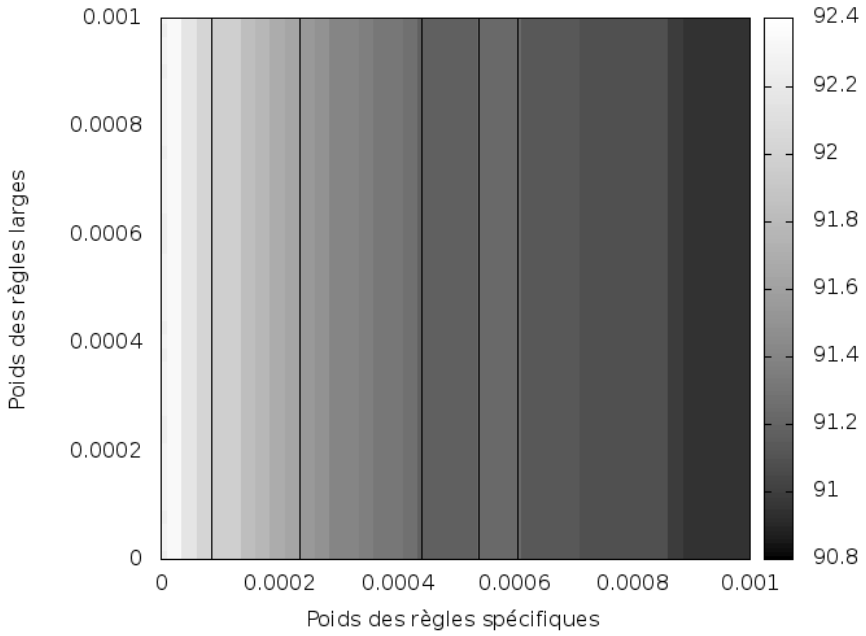

FIGURE A.4 – Précision pour les 2 meilleures normalisations en prenant en compte uniquement les règles spécifiques

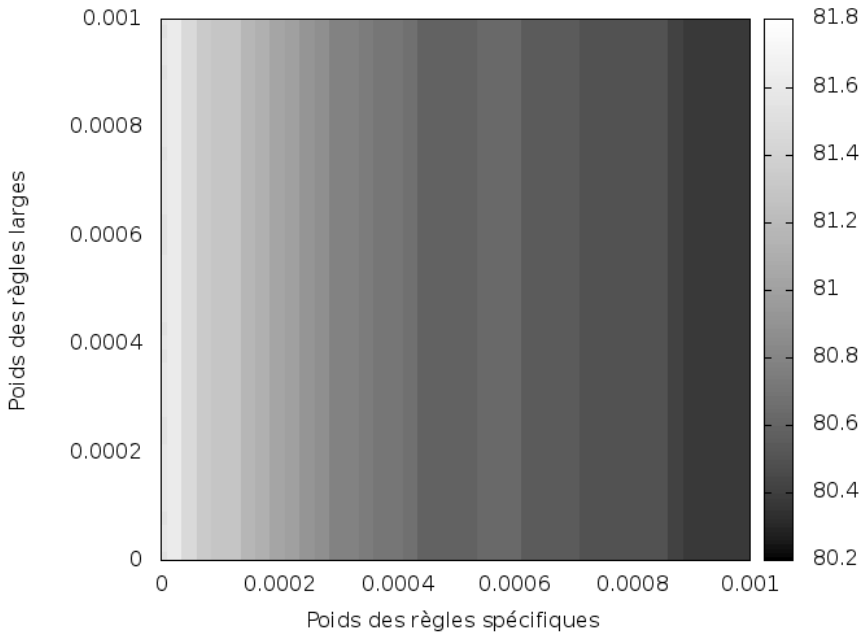

FIGURE A.5 – Rappel pour les 2 meilleures normalisations en prenant en compte uniquement les règles spécifiques

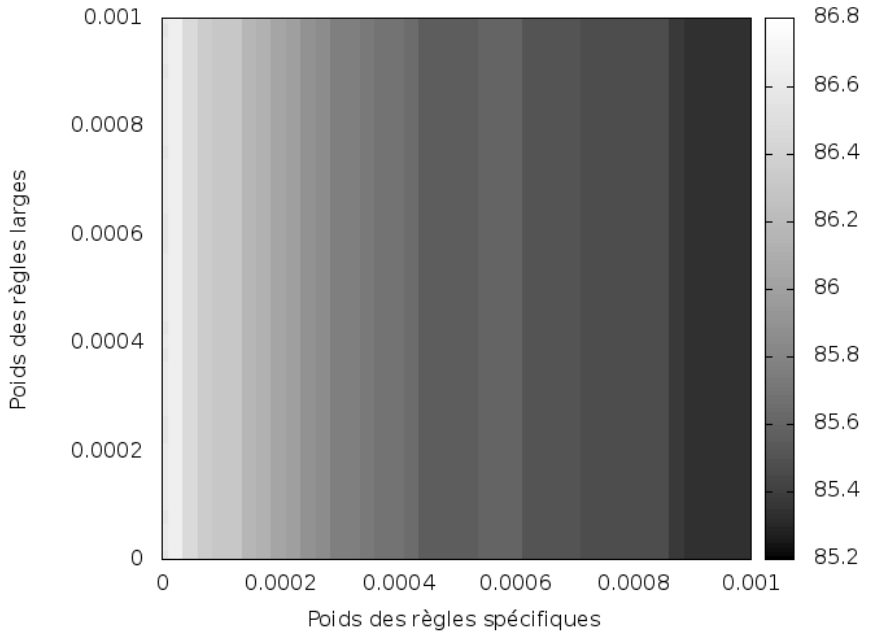

FIGURE A.6 – F-mesure pour les 2 meilleures normalisations en prenant en compte uniquement les règles spécifiques

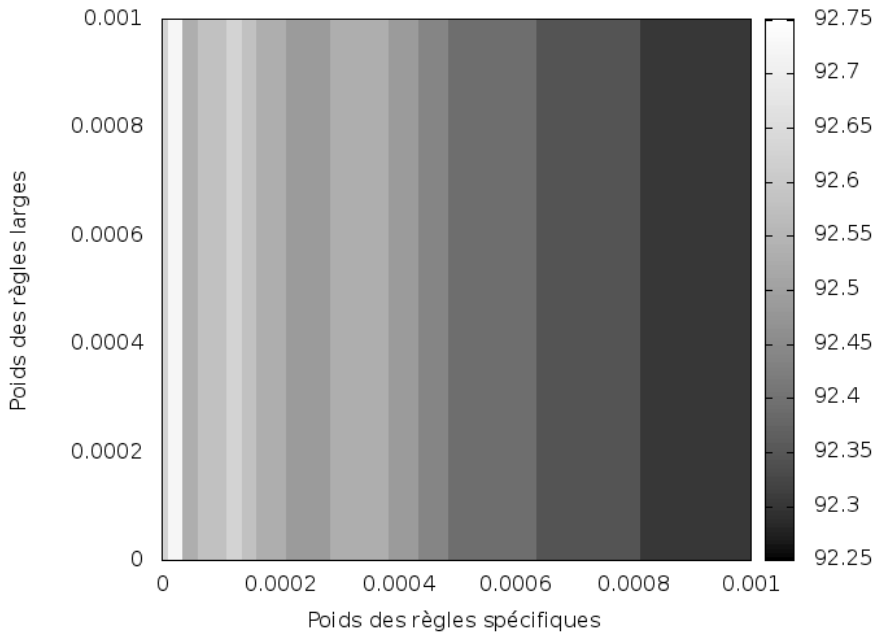

FIGURE A.7 – Précision pour les 3 meilleures normalisations en prenant en compte uniquement les règles spécifiques

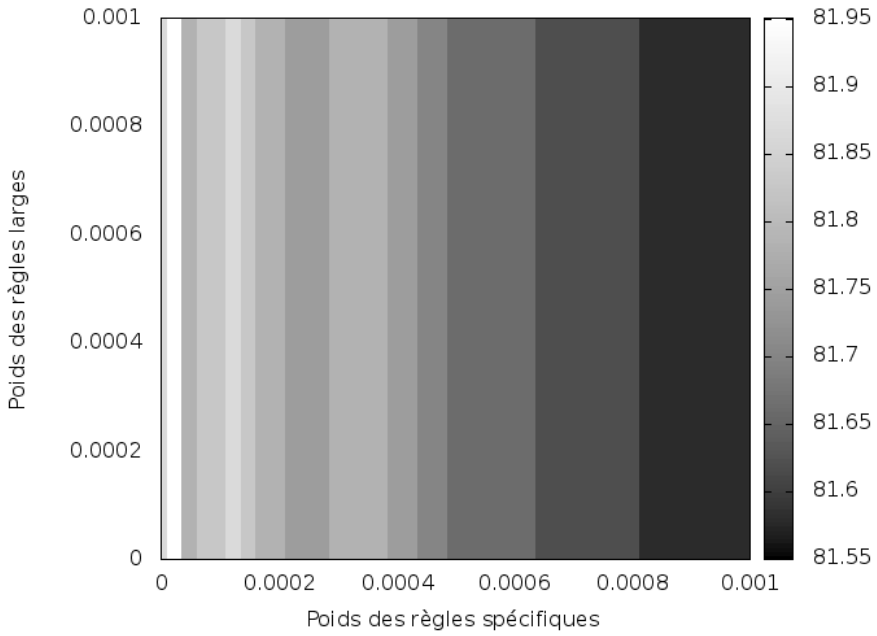

FIGURE A.8 – Rappel pour les 3 meilleures normalisations en prenant en compte uniquement les règles spécifiques

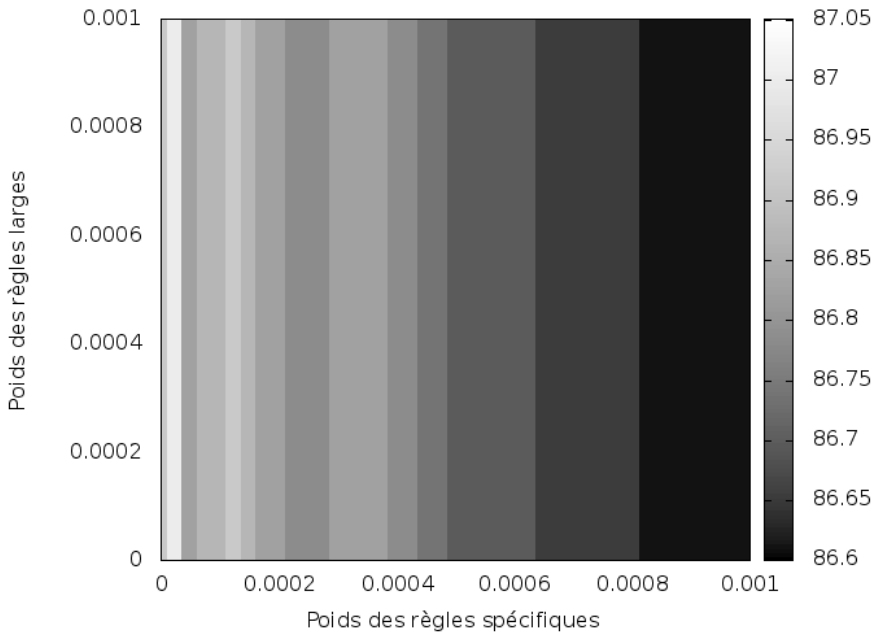

FIGURE A.9 – F-mesure pour les 3 meilleures normalisations en prenant en compte uniquement les règles spécifiques

A.2 Résultats obtenus par les règles larges seules

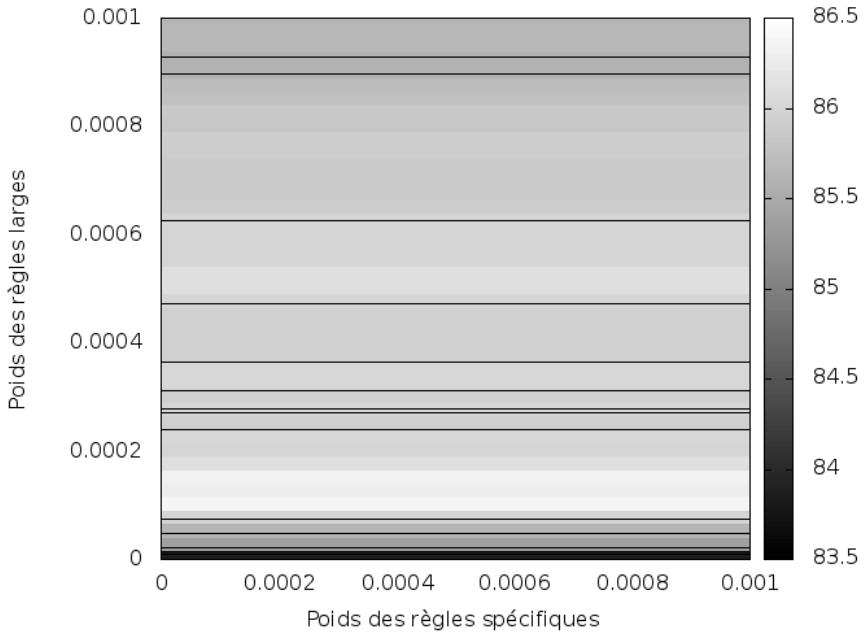

FIGURE A.10 – Précision pour la meilleure normalisation en prenant en compte uniquement les règles larges

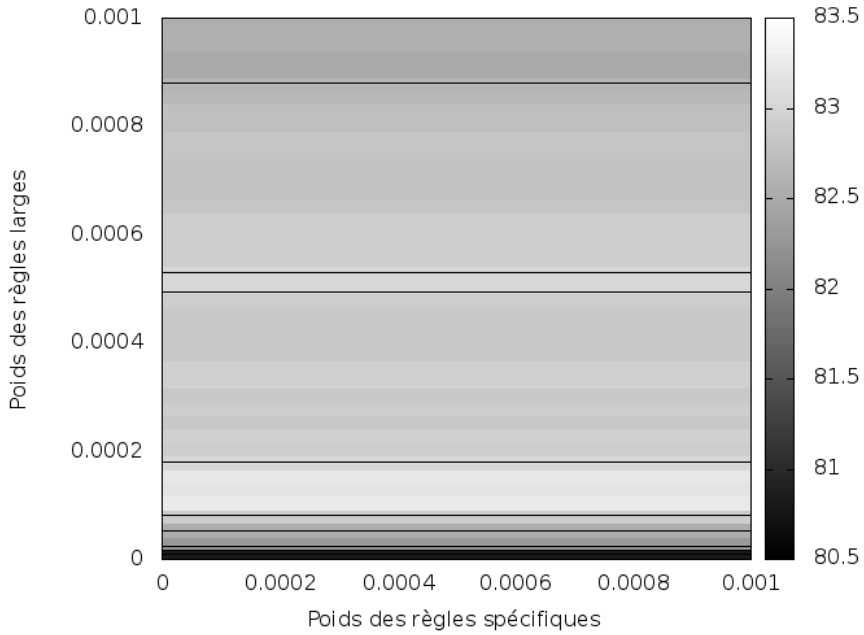

FIGURE A.11 – Rappel pour la meilleure normalisation en prenant en compte uniquement les règles larges

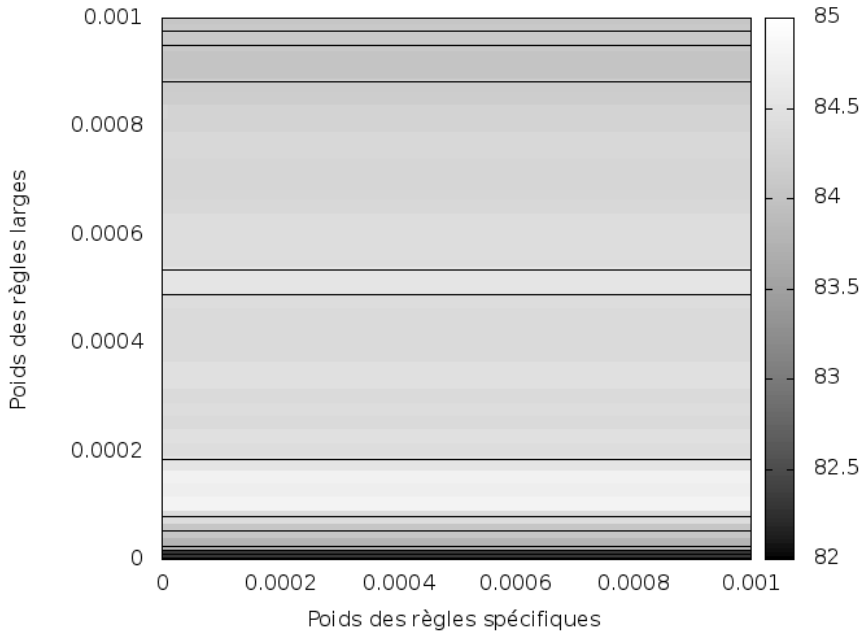

FIGURE A.12 – F-mesure pour la meilleure normalisation en prenant en compte uniquement les règles larges

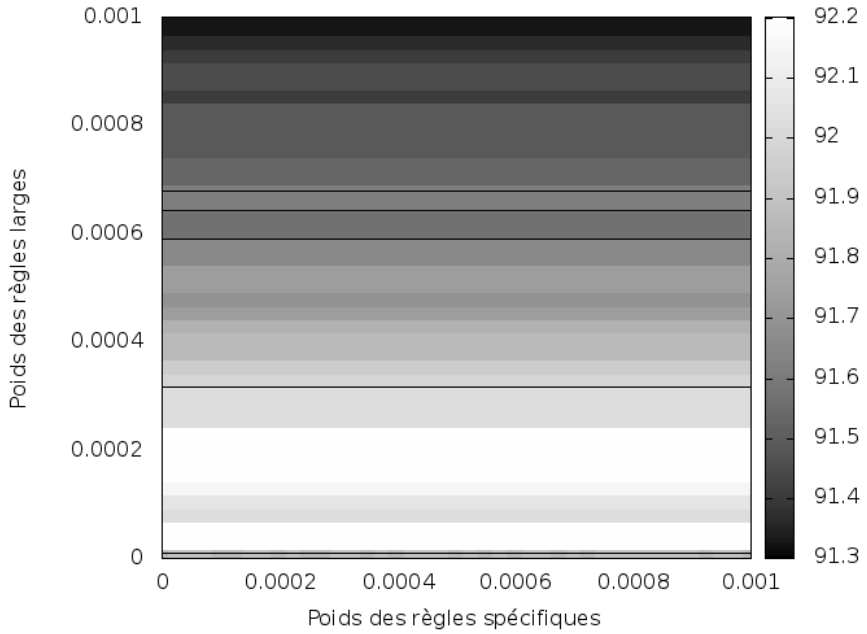

FIGURE A.13 – Précision pour les 2 meilleures normalisations en prenant en compte uniquement les règles larges

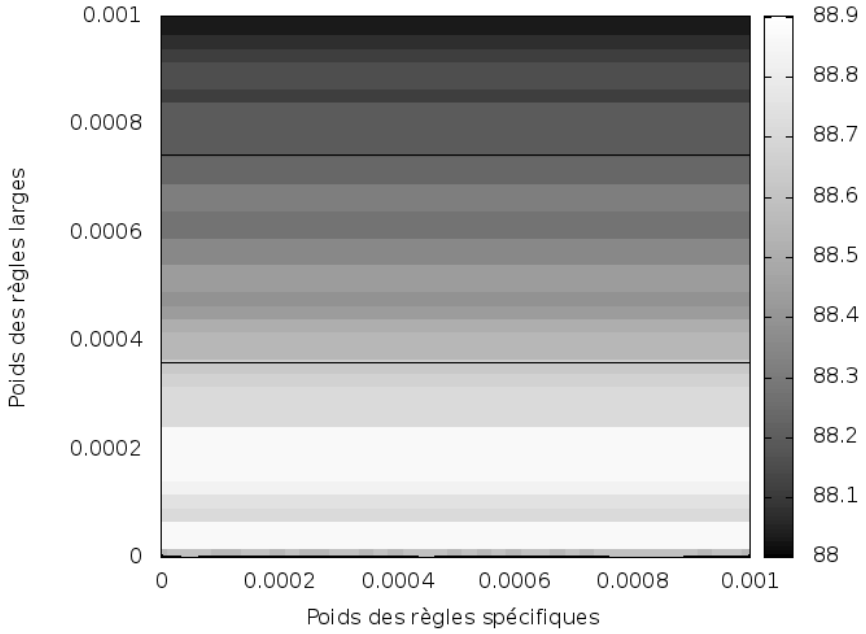

FIGURE A.14 – Rappel pour les 2 meilleures normalisations en prenant en compte uniquement les règles larges

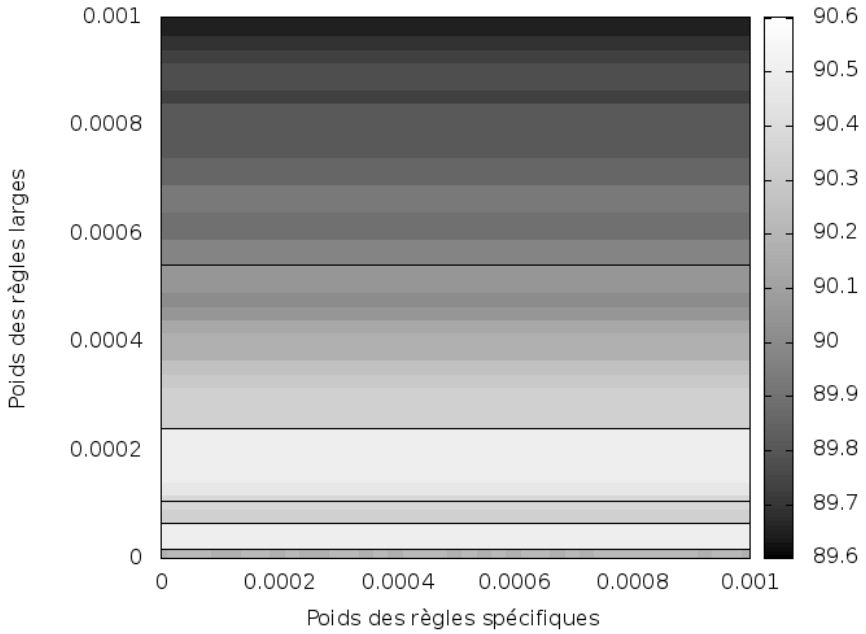

FIGURE A.15 – F-mesure pour les 2 meilleures normalisations en prenant en compte uniquement les règles larges

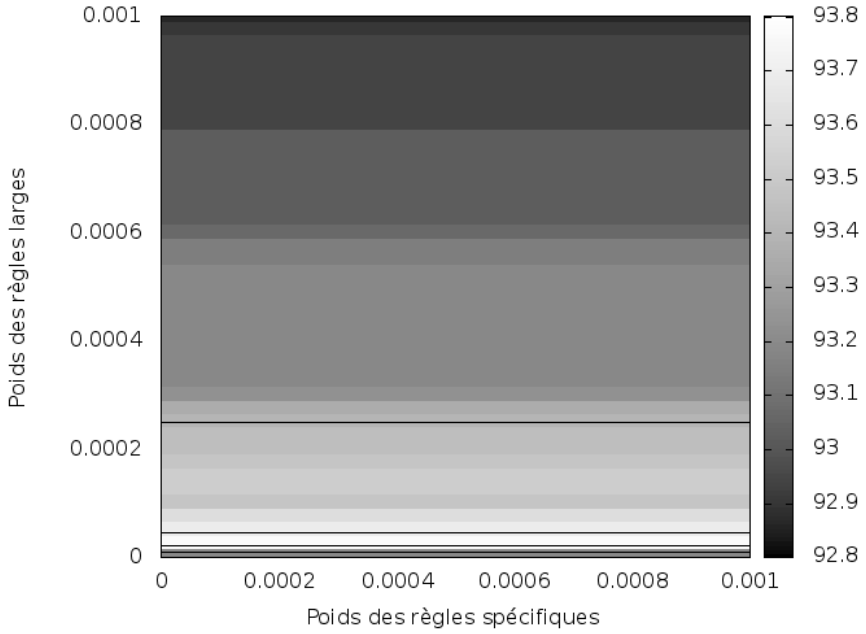

FIGURE A.16 – Précision pour les 3 meilleures normalisations en prenant en compte uniquement les règles larges

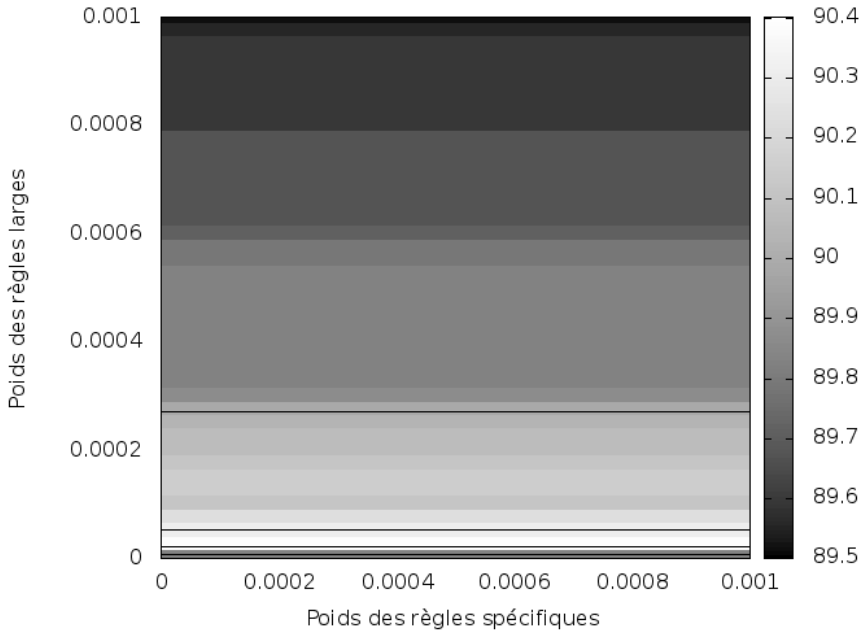

FIGURE A.17 – Rappel pour les 3 meilleures normalisations en prenant en compte uniquement les règles larges

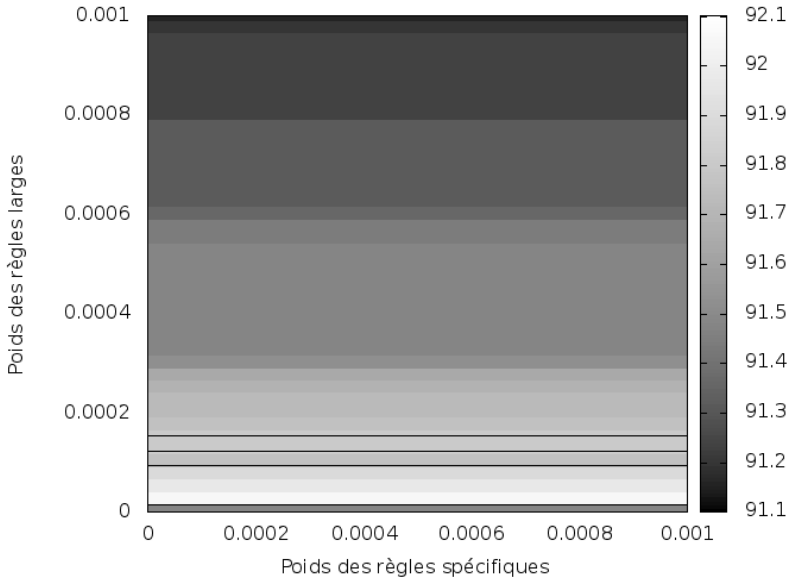

FIGURE A.18 – F-mesure pour les 3 meilleures normalisations en prenant en compte uniquement les règles larges

A.3 Résultats obtenus par les règles spécifiques puis larges

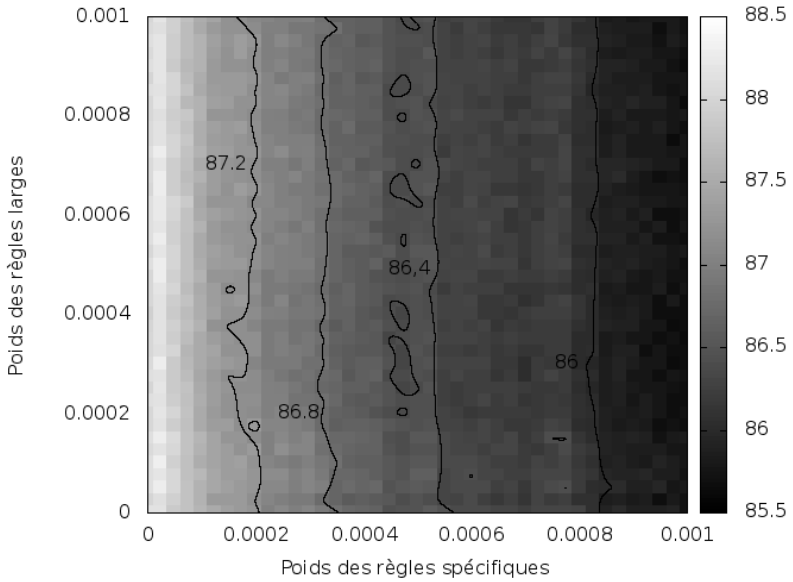

FIGURE A.19 – Précision pour la meilleure normalisation en prenant en compte successivement les règles spécifiques puis larges

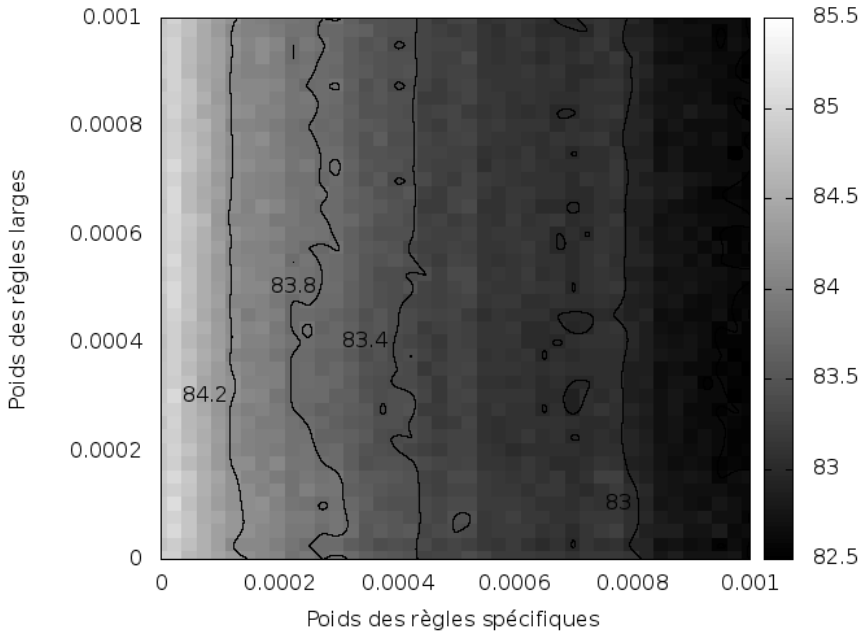

FIGURE A.20 – Rappel pour la meilleure normalisation en prenant en compte successivement les règles spécifiques puis larges

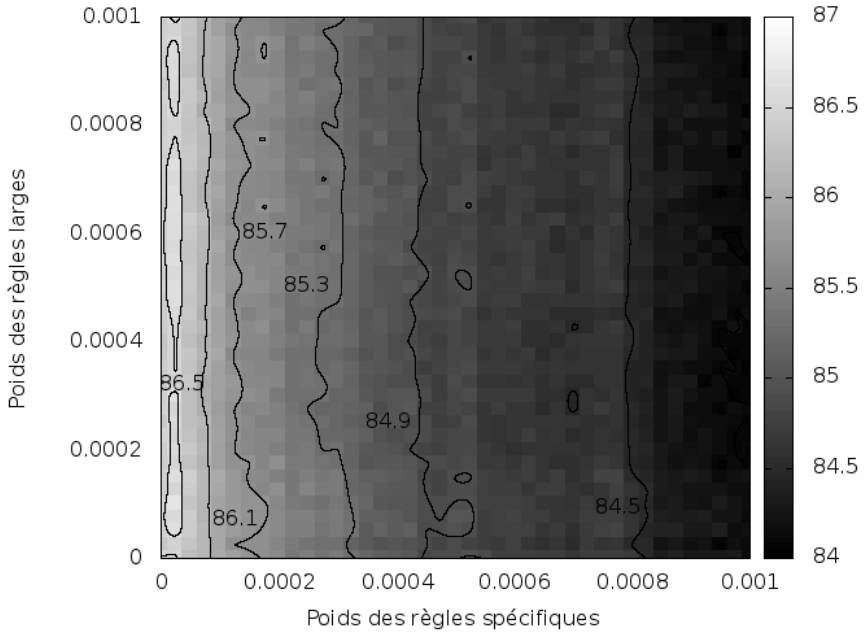

FIGURE A.21 – F-mesure pour la meilleure normalisation en prenant en compte successivement les règles spécifiques puis larges

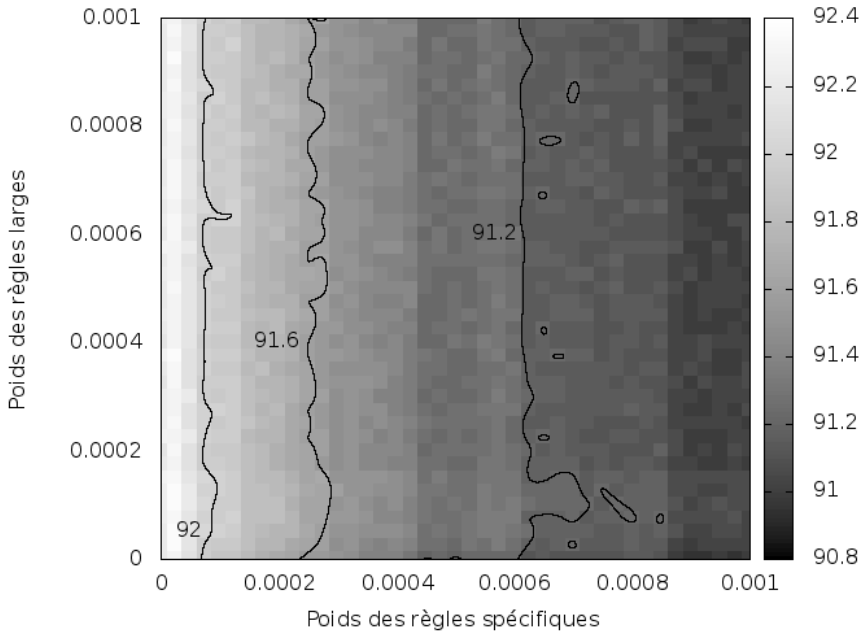

FIGURE A.22 – Précision pour les 2 meilleures normalisations en prenant en compte successivement les règles spécifiques puis larges

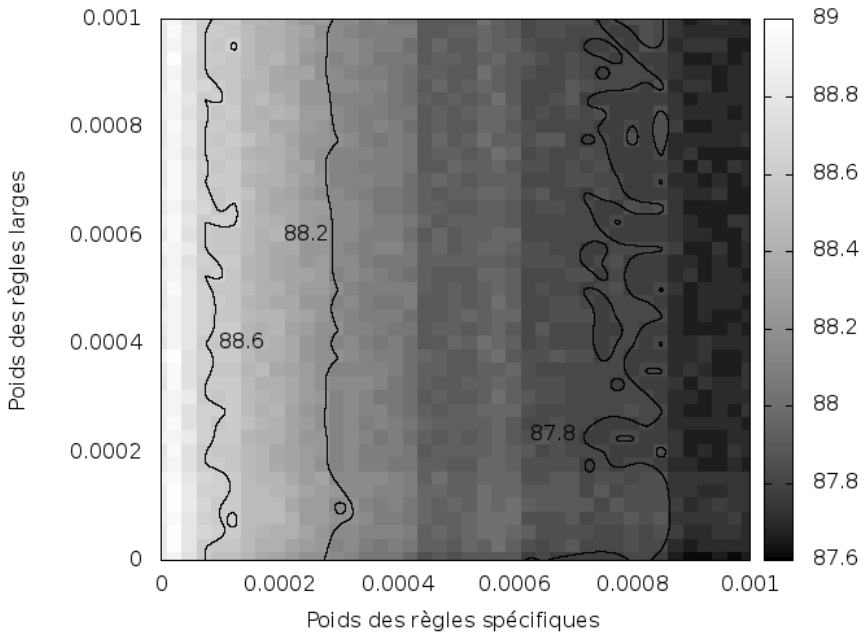

FIGURE A.23 – Rappel pour les 2 meilleures normalisations en prenant en compte successivement les règles spécifiques puis larges

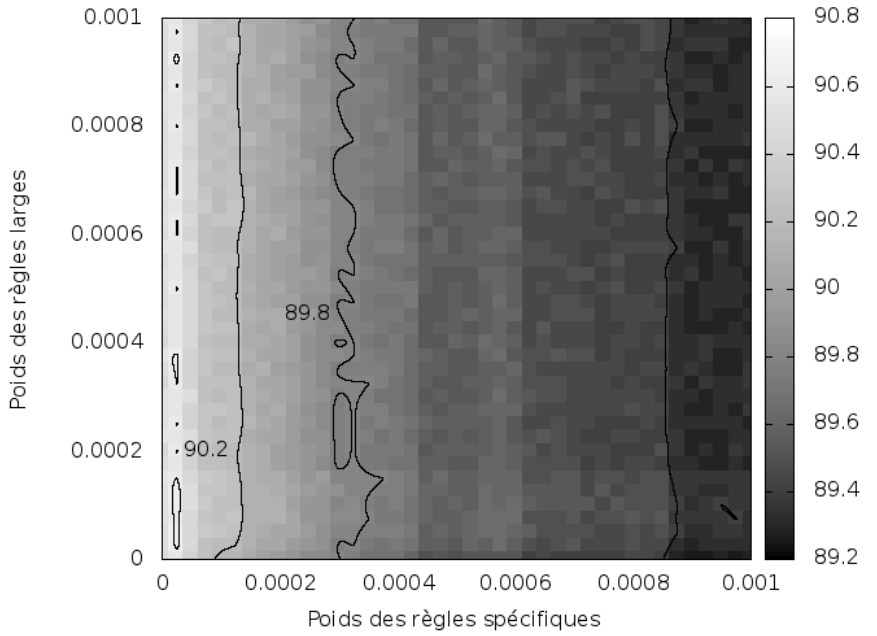

FIGURE A.24 – F-mesure pour les 2 meilleures normalisations en prenant en compte successivement les règles spécifiques puis larges

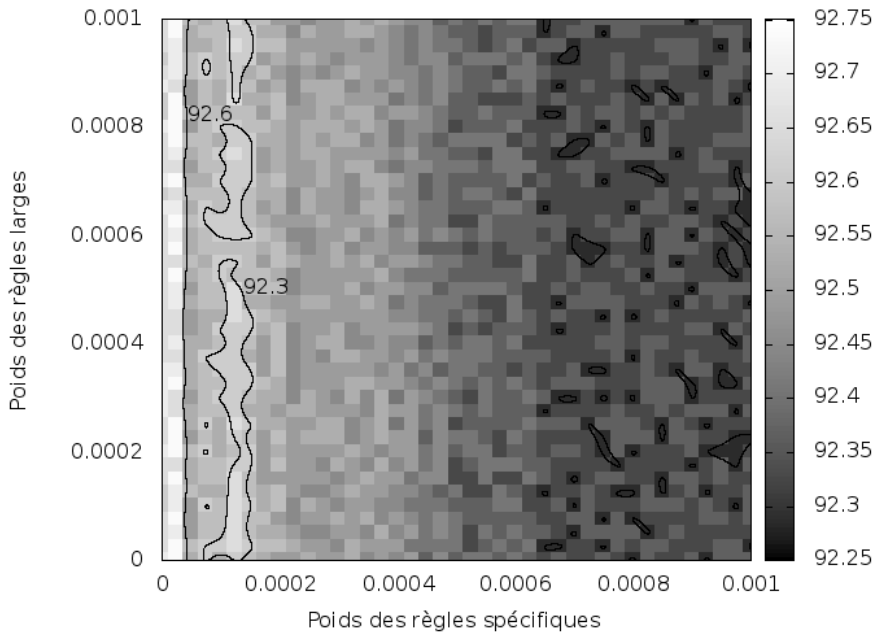

FIGURE A.25 – Précision pour les 3 meilleures normalisations en prenant en compte successivement les règles spécifiques puis larges

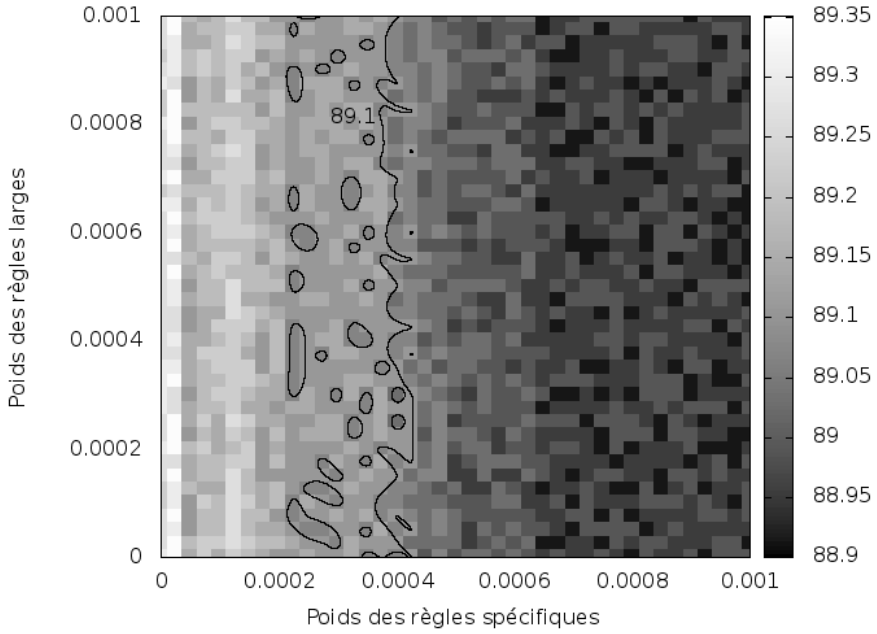

FIGURE A.26 – Rappel pour les 3 meilleures normalisations en prenant en compte successivement les règles spécifiques puis larges

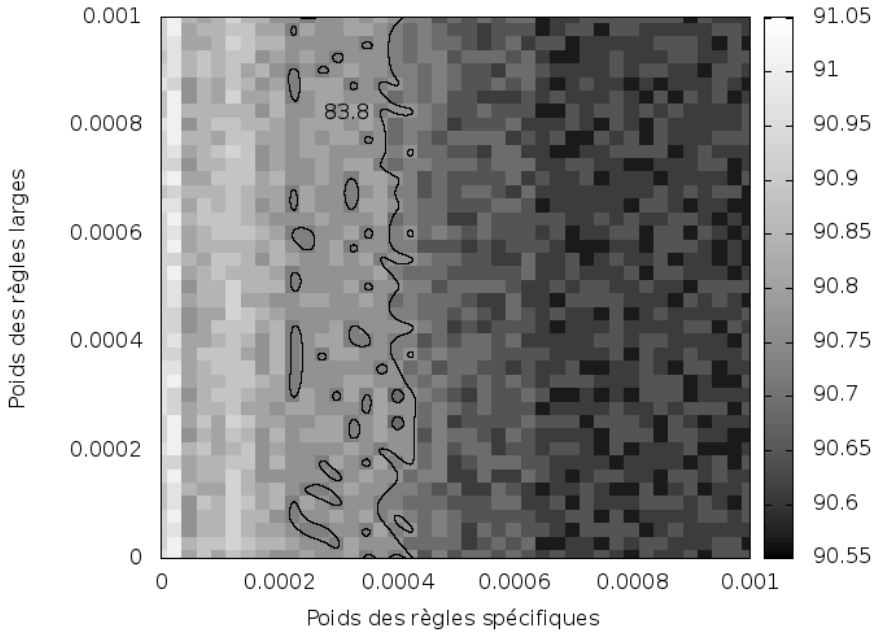

FIGURE A.27 – F-mesure pour les 3 meilleures normalisations en prenant en compte successivement les règles spécifiques puis larges

A.4 Résultats obtenus par les règles spécifiques et larges

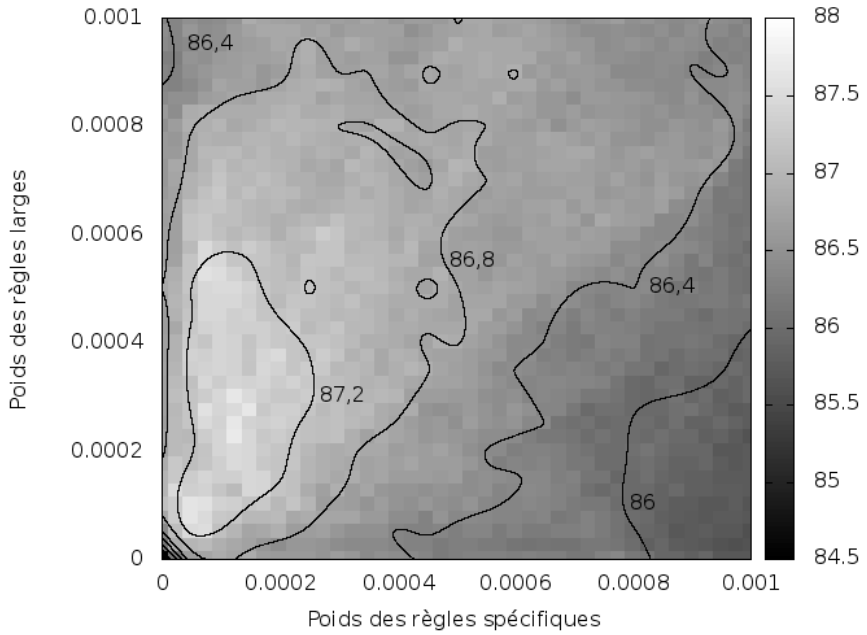

FIGURE A.28 – Précision pour la meilleure normalisation en prenant en compte simultanément les règles spécifiques et larges

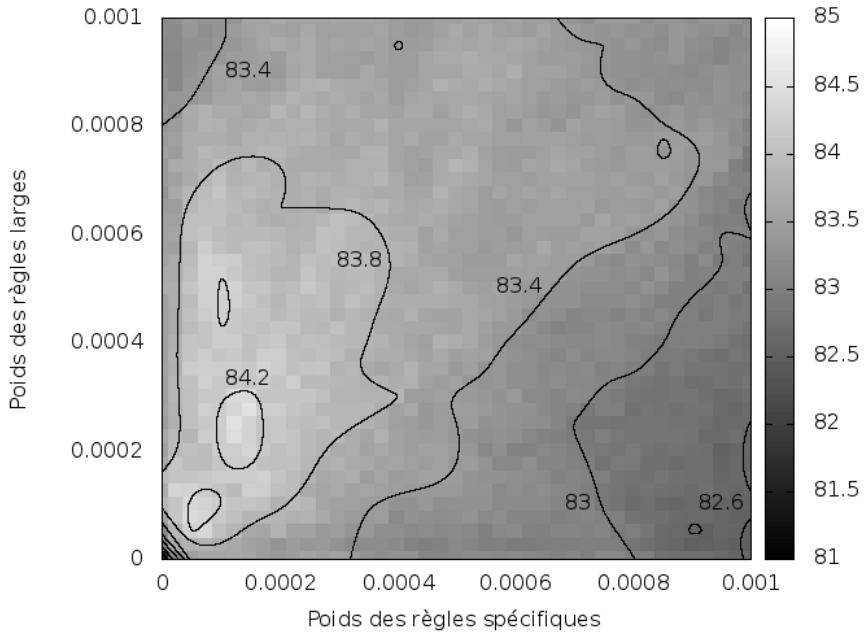

FIGURE A.29 – Rappel pour la meilleure normalisation en prenant en compte simultanément les règles spécifiques et larges

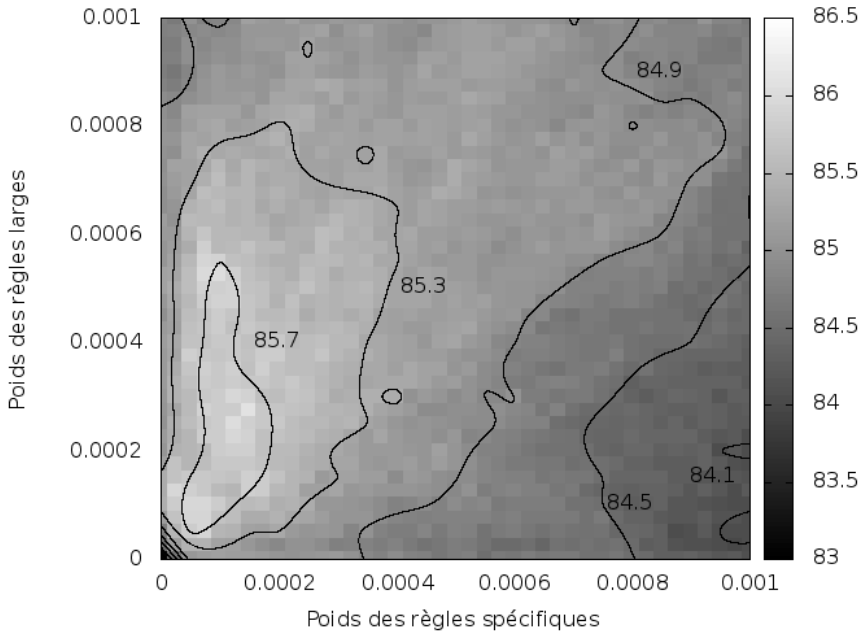

FIGURE A.30 – F-mesure pour la meilleure normalisation en prenant en compte simultanément les règles spécifiques et larges

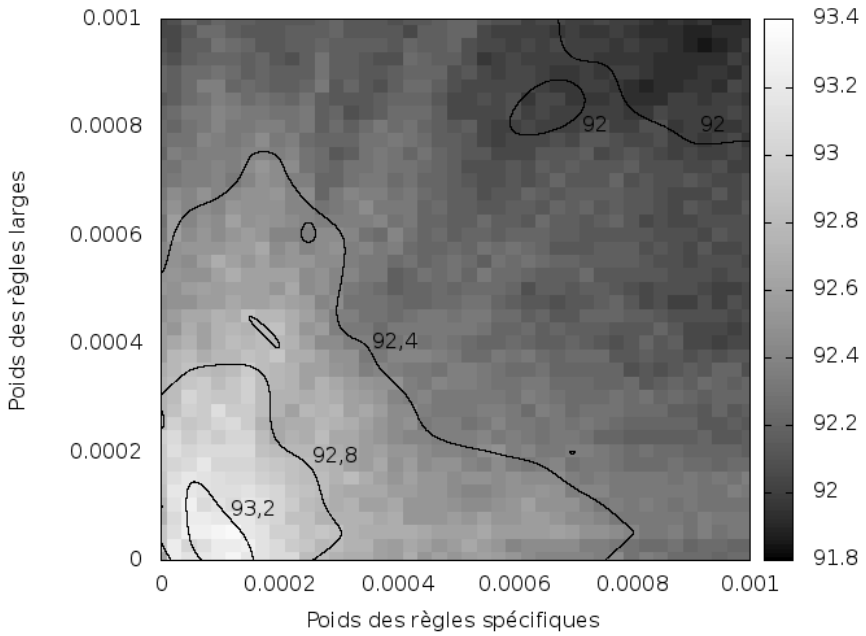

FIGURE A.31 – Précision pour les 2 meilleures normalisations en prenant en compte simultanément les règles spécifiques et larges

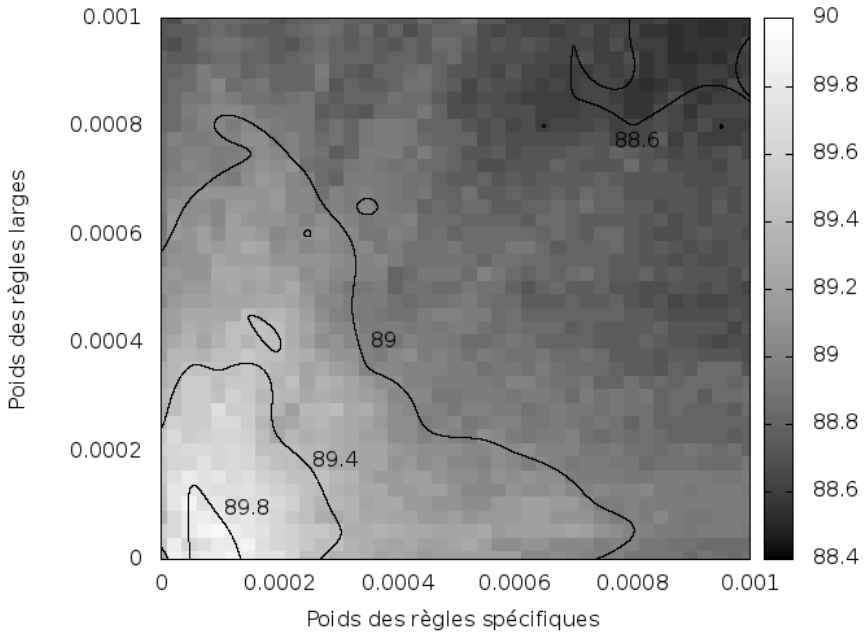

FIGURE A.32 – Rappel pour les 2 meilleures normalisations en prenant en compte simultanément les règles spécifiques et larges

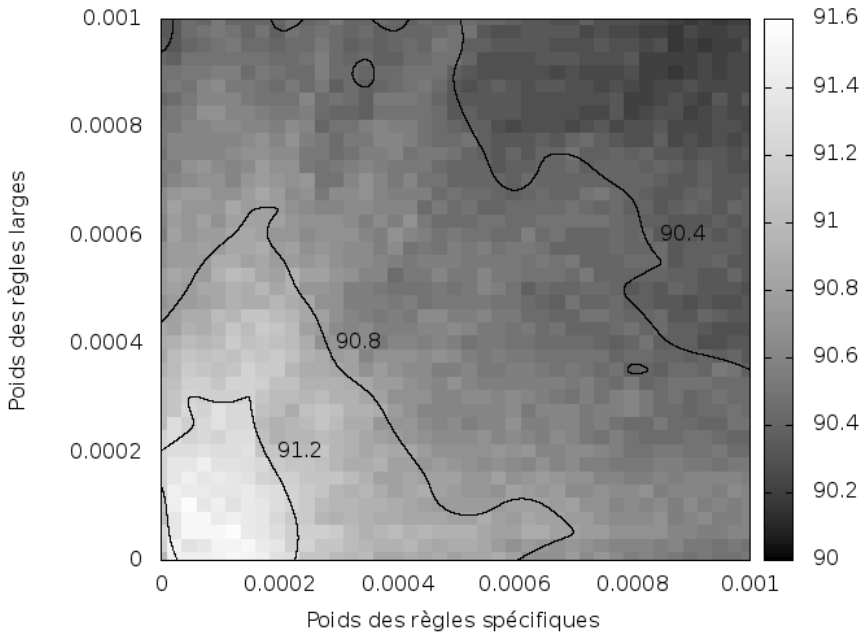

FIGURE A.33 – F-mesure pour les 2 meilleures normalisations en prenant en compte simultanément les règles spécifiques et larges

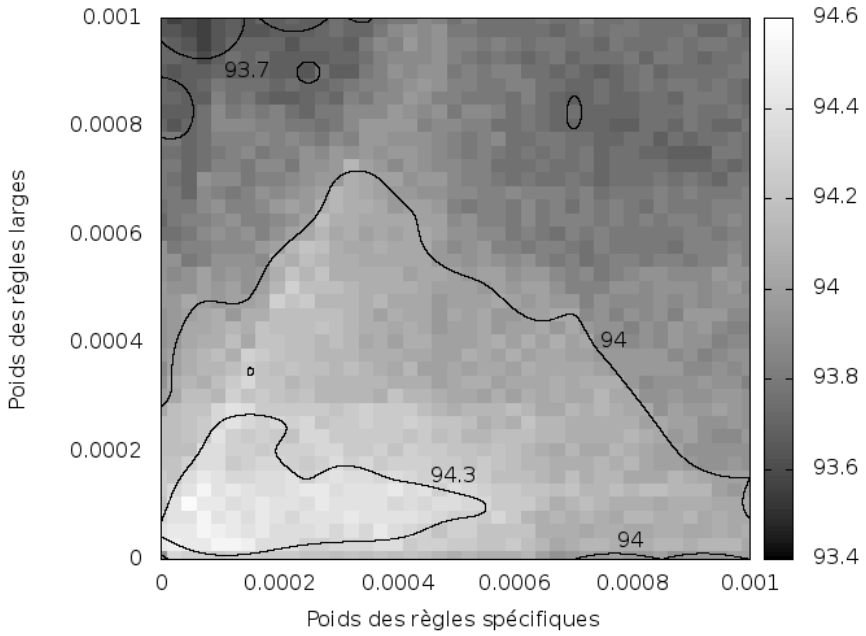

FIGURE A.34 – Précision pour les 3 meilleures normalisations en prenant en compte simultanément les règles spécifiques et larges

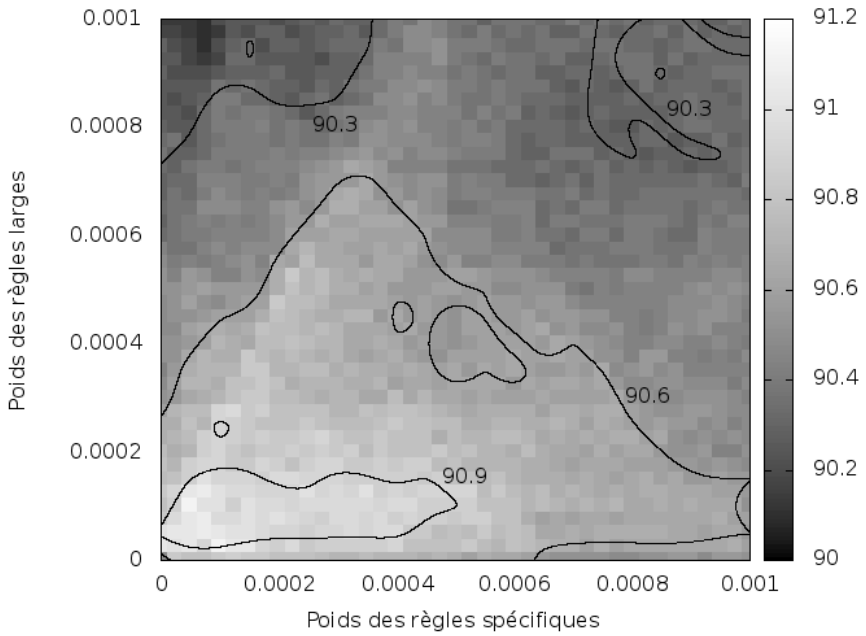

FIGURE A.35 – Rappel pour les 3 meilleures normalisations en prenant en compte simultanément les règles spécifiques et larges

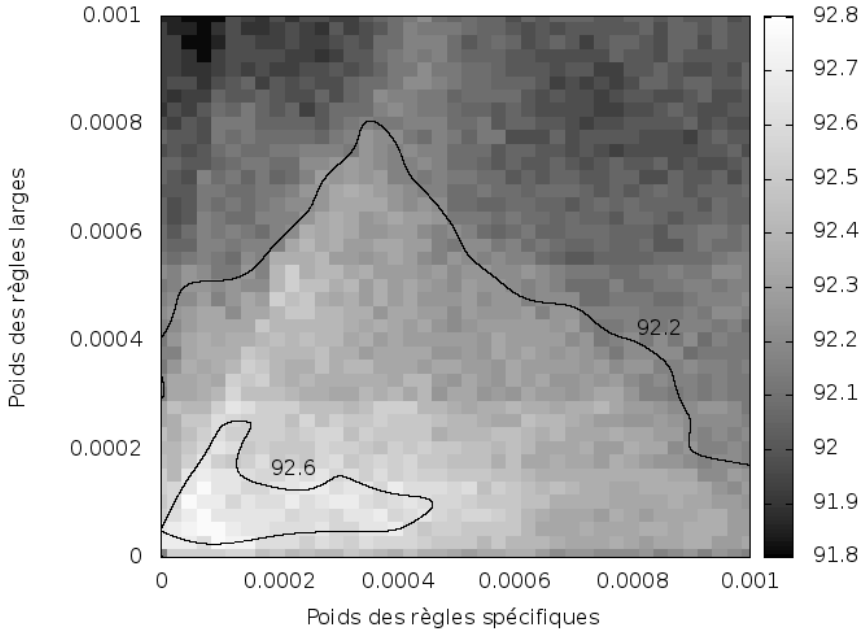

FIGURE A.36 – F-mesure pour les 3 meilleures normalisations en prenant en compte simultanément les règles spécifiques et larges

ANNEXE B

Détails des résultats obtenus par notre module contextuel

Les tableaux de résultats représentés dans cette section correspondent à tous les scores de précision (P), rappel (R) et F-mesure (F-M) obtenus en évaluant notre chaîne de prétraitement quand cette dernière prend en compte le contexte. Cette évaluation a été réalisée sur trois corpus distincts décrits en section 6.3.1 et prend en compte 11 modèles de langue distincts entraîné sur différentes quantités de texte et prenant en compte des bigrammes, des trigrammes ou encore des quadrigrammes. Enfin, ces évaluations ont été réalisée en conservant uniquement le chemin le mieux pondéré par chacun de nos modèles et en conservant les trois meilleurs chemins pour chaque seuil de fréquence envisagé pour la prise en compte de notre module d'homophonie dans notre évaluation (c'est à dire 0,60, 0,65, 0,70, 0,75, 0,80 et 0,90).

B.1 Résultats obtenus pour le corpus Corp1

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,771	0,604	0,677	0,808	0,647	0,719
50% Wiki+2-gram	0,768	0,599	0,673	0,803	0,649	0,718
33% Wiki+2-gram	0,751	0,588	0,660	0,789	0,634	0,703
25% Wiki+2-gram	0,749	0,587	0,658	0,792	0,639	0,707
17% Wiki+2-gram	0,745	0,581	0,653	0,789	0,635	0,704
50% Wiki+3-gram	0,764	0,601	0,673	0,805	0,644	0,716
33% Wiki+3-gram	0,764	0,596	0,670	0,806	0,644	0,716
25% Wiki+3-gram	0,763	0,592	0,667	0,802	0,638	0,711
17% Wiki+3-gram	0,760	0,588	0,663	0,805	0,643	0,715
25% Wiki+4-gram	0,769	0,595	0,671	0,807	0,642	0,715
17% Wiki+4-gram	0,765	0,588	0,665	0,808	0,644	0,717

Table B.1 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,60 pour le corpus Corp1

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,759	0,563	0,646	0,800	0,611	0,693
50% Wiki+2-gram	0,758	0,562	0,645	0,800	0,609	0,692
33% Wiki+2-gram	0,743	0,555	0,635	0,797	0,605	0,688
25% Wiki+2-gram	0,738	0,552	0,632	0,797	0,603	0,687
17% Wiki+2-gram	0,740	0,550	0,631	0,791	0,604	0,685
50% Wiki+3-gram	0,756	0,571	0,651	0,806	0,618	0,700
33% Wiki+3-gram	0,758	0,567	0,649	0,802	0,615	0,696
25% Wiki+3-gram	0,755	0,563	0,645	0,799	0,609	0,691
17% Wiki+3-gram	0,751	0,562	0,643	0,801	0,611	0,693
25% Wiki+4-gram	0,761	0,570	0,652	0,800	0,609	0,692
17% Wiki+4-gram	0,758	0,566	0,648	0,808	0,615	0,698

Table B.2 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,65 pour le corpus Corp1

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,760	0,539	0,631	0,804	0,594	0,683
50% Wiki+2-gram	0,758	0,537	0,629	0,801	0,591	0,680
33% Wiki+2-gram	0,761	0,542	0,633	0,796	0,587	0,676
25% Wiki+2-gram	0,756	0,539	0,629	0,798	0,585	0,675
17% Wiki+2-gram	0,754	0,537	0,627	0,802	0,596	0,684
50% Wiki+3-gram	0,761	0,547	0,636	0,805	0,598	0,686
33% Wiki+3-gram	0,766	0,549	0,640	0,800	0,596	0,683
25% Wiki+3-gram	0,763	0,545	0,636	0,796	0,592	0,679
17% Wiki+3-gram	0,763	0,546	0,637	0,803	0,600	0,687
25% Wiki+4-gram	0,763	0,546	0,637	0,801	0,591	0,680
17% Wiki+4-gram	0,763	0,546	0,637	0,807	0,598	0,687

Table B.3 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,70 pour le corpus Corp1

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,751	0,540	0,628	0,801	0,591	0,680
50% Wiki+2-gram	0,751	0,540	0,628	0,800	0,587	0,677
33% Wiki+2-gram	0,750	0,543	0,630	0,795	0,583	0,673
25% Wiki+2-gram	0,747	0,540	0,627	0,797	0,582	0,673
17% Wiki+2-gram	0,742	0,536	0,622	0,797	0,591	0,679
50% Wiki+3-gram	0,749	0,545	0,631	0,805	0,591	0,682
33% Wiki+3-gram	0,756	0,549	0,636	0,799	0,591	0,679
25% Wiki+3-gram	0,752	0,545	0,632	0,799	0,589	0,678
17% Wiki+3-gram	0,748	0,543	0,629	0,800	0,592	0,680
25% Wiki+4-gram	0,756	0,549	0,636	0,804	0,589	0,680
17% Wiki+4-gram	0,751	0,545	0,632	0,805	0,591	0,682

Table B.4 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,75 pour le corpus Corp1

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,747	0,525	0,617	0,800	0,582	0,674
50% Wiki+2-gram	0,747	0,524	0,616	0,799	0,578	0,671
33% Wiki+2-gram	0,746	0,527	0,618	0,796	0,574	0,667
25% Wiki+2-gram	0,744	0,525	0,616	0,796	0,572	0,666
17% Wiki+2-gram	0,738	0,522	0,611	0,798	0,580	0,672
50% Wiki+3-gram	0,747	0,527	0,617	0,804	0,580	0,674
33% Wiki+3-gram	0,751	0,527	0,622	0,799	0,578	0,671
25% Wiki+3-gram	0,749	0,529	0,620	0,799	0,576	0,669
17% Wiki+3-gram	0,744	0,525	0,616	0,800	0,580	0,672
25% Wiki+4-gram	0,752	0,533	0,624	0,806	0,578	0,673
17% Wiki+4-gram	0,747	0,529	0,619	0,806	0,580	0,675

Table B.5 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,80 pour le corpus Corp1

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,669	0,344	0,454	0,734	0,384	0,504
50% Wiki+2-gram	0,667	0,341	0,451	0,729	0,380	0,500
33% Wiki+2-gram	0,662	0,341	0,450	0,733	0,382	0,502
25% Wiki+2-gram	0,655	0,337	0,445	0,729	0,380	0,500
17% Wiki+2-gram	0,651	0,335	0,442	0,733	0,382	0,502
50% Wiki+3-gram	0,660	0,341	0,450	0,737	0,386	0,507
33% Wiki+3-gram	0,451	0,341	0,451	0,730	0,382	0,502
25% Wiki+3-gram	0,661	0,339	0,448	0,727	0,380	0,499
17% Wiki+3-gram	0,655	0,337	0,445	0,732	0,380	0,500
25% Wiki+4-gram	0,664	0,341	0,451	0,735	0,382	0,503
17% Wiki+4-gram	0,658	0,339	0,447	0,740	0,382	0,504

Table B.6 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,90 pour le corpus Corp1

B.2 Résultats obtenus pour le corpus Corp2

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,908	0,754	0,824	0,922	0,781	0,846
50% Wiki+2-gram	0,904	0,751	0,820	0,921	0,777	0,843
33% Wiki+2-gram	0,904	0,748	0,819	0,917	0,767	0,835
25% Wiki+2-gram	0,904	0,748	0,819	0,913	0,764	0,832
17% Wiki+2-gram	0,907	0,748	0,820	0,917	0,767	0,835
50% Wiki+3-gram	0,912	0,754	0,826	0,920	0,771	0,839
33% Wiki+3-gram	0,907	0,744	0,817	0,917	0,771	0,838
25% Wiki+3-gram	0,899	0,741	0,812	0,917	0,767	0,835
17% Wiki+3-gram	0,907	0,744	0,817	0,920	0,767	0,837
25% Wiki+4-gram	0,908	0,748	0,820	0,921	0,774	0,841
17% Wiki+4-gram	0,911	0,751	0,823	0,922	0,777	0,845

Table B.7 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,60 pour le corpus Corp2

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,905	0,731	0,809	0,927	0,757	0,833
50% Wiki+2-gram	0,898	0,728	0,804	0,923	0,754	0,830
33% Wiki+2-gram	0,897	0,724	0,801	0,918	0,748	0,824
25% Wiki+2-gram	0,897	0,724	0,801	0,918	0,744	0,822
17% Wiki+2-gram	0,901	0,724	0,803	0,926	0,751	0,829
50% Wiki+3-gram	0,905	0,728	0,807	0,922	0,751	0,828
33% Wiki+3-gram	0,901	0,724	0,803	0,922	0,744	0,823
25% Wiki+3-gram	0,893	0,721	0,798	0,914	0,741	0,818
17% Wiki+3-gram	0,897	0,724	0,801	0,918	0,744	0,822
25% Wiki+4-gram	0,902	0,728	0,806	0,927	0,754	0,832
17% Wiki+4-gram	0,901	0,728	0,805	0,927	0,754	0,832

Table B.8 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,65 pour le corpus Corp2

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,919	0,718	0,806	0,945	0,738	0,829
50% Wiki+2-gram	0,915	0,714	0,802	0,945	0,738	0,829
33% Wiki+2-gram	0,915	0,714	0,802	0,936	0,734	0,823
25% Wiki+2-gram	0,915	0,714	0,802	0,936	0,734	0,823
17% Wiki+2-gram	0,919	0,714	0,804	0,945	0,738	0,829
50% Wiki+3-gram	0,919	0,718	0,806	0,941	0,738	0,827
33% Wiki+3-gram	0,915	0,718	0,805	0,936	0,734	0,823
25% Wiki+3-gram	0,907	0,711	0,797	0,940	0,734	0,824
17% Wiki+3-gram	0,911	0,714	0,801	0,940	0,734	0,824
25% Wiki+4-gram	0,911	0,715	0,801	0,940	0,734	0,824
17% Wiki+4-gram	0,911	0,714	0,801	0,940	0,734	0,824

Table B.9 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,70 pour le corpus Corp2

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,905	0,698	0,788	0,935	0,714	0,810
50% Wiki+2-gram	0,897	0,694	0,783	0,926	0,711	0,804
33% Wiki+2-gram	0,897	0,694	0,783	0,922	0,711	0,803
25% Wiki+2-gram	0,897	0,694	0,783	0,926	0,711	0,804
17% Wiki+2-gram	0,901	0,694	0,784	0,934	0,711	0,807
50% Wiki+3-gram	0,901	0,698	0,783	0,931	0,721	0,813
33% Wiki+3-gram	0,897	0,698	0,785	0,927	0,718	0,809
25% Wiki+3-gram	0,889	0,691	0,778	0,927	0,714	0,807
17% Wiki+3-gram	0,893	0,694	0,781	0,931	0,718	0,811
25% Wiki+4-gram	0,897	0,698	0,785	0,935	0,721	0,814
17% Wiki+4-gram	0,897	0,698	0,785	0,935	0,721	0,814

Table B.10 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,75 pour le corpus Corp2

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,904	0,688	0,782	0,934	0,708	0,805
50% Wiki+2-gram	0,896	0,684	0,776	0,930	0,708	0,804
33% Wiki+2-gram	0,896	0,684	0,78	0,926	0,708	0,802
25% Wiki+2-gram	0,896	0,684	0,776	0,930	0,708	0,804
17% Wiki+2-gram	0,900	0,688	0,780	0,934	0,704	0,803
50% Wiki+3-gram	0,900	0,688	0,780	0,931	0,714	0,808
33% Wiki+3-gram	0,896	0,688	0,778	0,926	0,711	0,804
25% Wiki+3-gram	0,887	0,681	0,770	0,926	0,708	0,802
17% Wiki+3-gram	0,892	0,684	0,774	0,930	0,711	0,806
25% Wiki+4-gram	0,896	0,688	0,778	0,930	0,710	0,810
17% Wiki+4-gram	0,896	0,688	0,778	0,935	0,714	0,810

Table B.11 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,80 pour le corpus Corp2

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,879	0,505	0,641	0,920	0,532	0,674
50% Wiki+2-gram	0,868	0,502	0,636	0,914	0,532	0,673
33% Wiki+2-gram	0,868	0,502	0,636	0,909	0,532	0,671
25% Wiki+2-gram	0,868	0,502	0,636	0,914	0,532	0,673
17% Wiki+2-gram	0,874	0,505	0,640	0,920	0,532	0,674
50% Wiki+3-gram	0,874	0,505	0,640	0,909	0,532	0,671
33% Wiki+3-gram	0,869	0,505	0,639	0,904	0,532	0,670
25% Wiki+3-gram	0,857	0,498	0,630	0,909	0,532	0,671
17% Wiki+3-gram	0,863	0,502	0,635	0,915	0,535	0,675
25% Wiki+4-gram	0,869	0,505	0,639	0,915	0,535	0,675
17% Wiki+4-gram	0,869	0,505	0,639	0,915	0,535	0,675

Table B.12 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,90 pour le corpus Corp2

B.3 Résultats obtenus pour le corpus Corp3

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,753	0,618	0,679	0,789	0,642	0,708
50% Wiki+2-gram	0,747	0,613	0,673	0,784	0,639	0,704
33% Wiki+2-gram	0,747	0,613	0,673	0,777	0,630	0,696
25% Wiki+2-gram	0,742	0,608	0,668	0,771	0,625	0,690
17% Wiki+2-gram	0,747	0,613	0,673	0,768	0,630	0,692
50% Wiki+3-gram	0,740	0,604	0,665	0,787	0,634	0,702
33% Wiki+3-gram	0,746	0,608	0,670	0,782	0,630	0,698
25% Wiki+3-gram	0,753	0,618	0,679	0,771	0,625	0,690
17% Wiki+3-gram	0,746	0,608	0,670	0,768	0,630	0,692
25% Wiki+4-gram	0,753	0,618	0,679	0,780	0,639	0,702
17% Wiki+4-gram	0,747	0,613	0,673	0,774	0,634	0,697

Table B.13 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,60 pour le corpus Corp3

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,754	0,611	0,675	0,786	0,633	0,701
50% Wiki+2-gram	0,749	0,606	0,670	0,786	0,633	0,701
33% Wiki+2-gram	0,747	0,602	0,667	0,78	0,628	0,696
25% Wiki+2-gram	0,743	0,602	0,665	0,775	0,623	0,691
17% Wiki+2-gram	0,749	0,606	0,670	0,775	0,623	0,691
50% Wiki+3-gram	0,740	0,593	0,658	0,788	0,623	0,696
33% Wiki+3-gram	0,746	0,597	0,663	0,782	0,619	0,691
25% Wiki+3-gram	0,754	0,611	0,675	0,772	0,614	0,684
17% Wiki+3-gram	0,747	0,602	0,667	0,769	0,619	0,686
25% Wiki+4-gram	0,753	0,606	0,672	0,78	0,628	0,696
17% Wiki+4-gram	0,749	0,606	0,670	0,775	0,623	0,691

Table B.14 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,65 pour le corpus Corp3

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,757	0,606	0,673	0,799	0,628	0,703
50% Wiki+2-gram	0,751	0,602	0,668	0,800	0,633	0,707
33% Wiki+2-gram	0,750	0,597	0,665	0,794	0,628	0,701
25% Wiki+2-gram	0,746	0,597	0,663	0,793	0,623	0,698
17% Wiki+2-gram	0,751	0,602	0,668	0,788	0,623	0,696
50% Wiki+3-gram	0,741	0,583	0,653	0,802	0,623	0,701
33% Wiki+3-gram	0,747	0,588	0,658	0,796	0,619	0,696
25% Wiki+3-gram	0,756	0,602	0,670	0,786	0,614	0,689
17% Wiki+3-gram	0,749	0,593	0,662	0,787	0,619	0,693
25% Wiki+4-gram	0,749	0,593	0,662	0,794	0,628	0,701
17% Wiki+4-gram	0,750	0,597	0,665	0,788	0,623	0,696

Table B.15 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,70 pour le corpus Corp3

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,749	0,593	0,662	0,784	0,609	0,686
50% Wiki+2-gram	0,743	0,588	0,656	0,784	0,609	0,686
33% Wiki+2-gram	0,741	0,583	0,653	0,784	0,609	0,686
25% Wiki+2-gram	0,737	0,583	0,651	0,778	0,605	0,681
17% Wiki+2-gram	0,743	0,588	0,656	0,778	0,605	0,681
50% Wiki+3-gram	0,732	0,569	0,640	0,791	0,600	0,682
33% Wiki+3-gram	0,737	0,569	0,642	0,785	0,595	0,677
25% Wiki+3-gram	0,746	0,583	0,655	0,774	0,591	0,670
17% Wiki+3-gram	0,740	0,579	0,650	0,776	0,595	0,674
25% Wiki+4-gram	0,746	0,583	0,655	0,783	0,605	0,683
17% Wiki+4-gram	0,741	0,583	0,653	0,777	0,600	0,677

Table B.16 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,75 pour le corpus Corp3

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,707	0,564	0,627	0,741	0,581	0,651
50% Wiki+2-gram	0,701	0,565	0,623	0,741	0,581	0,651
33% Wiki+2-gram	0,735	0,555	0,619	0,741	0,581	0,651
25% Wiki+2-gram	0,735	0,555	0,617	0,735	0,576	0,646
17% Wiki+2-gram	0,701	0,560	0,623	0,735	0,576	0,646
50% Wiki+3-gram	0,735	0,565	0,639	0,795	0,595	0,681
33% Wiki+3-gram	0,735	0,565	0,639	0,789	0,591	0,676
25% Wiki+3-gram	0,749	0,579	0,653	0,784	0,591	0,674
17% Wiki+3-gram	0,703	0,555	0,620	0,737	0,567	0,641
25% Wiki+4-gram	0,744	0,579	0,623	0,787	0,600	0,681
17% Wiki+4-gram	0,703	0,555	0,619	0,738	0,571	0,644

Table B.17 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,80 pour le corpus Corp3

Modèles conçus	Meilleur chemin			3 meilleurs chemins		
	P	R	F-M	P	R	F-M
100% Wiki+2-gram	0,707	0,544	0,615	0,753	0,562	0,644
50% Wiki+2-gram	0,701	0,539	0,609	0,747	0,558	0,639
33% Wiki+2-gram	0,699	0,535	0,606	0,747	0,558	0,639
25% Wiki+2-gram	0,695	0,695	0,605	0,741	0,553	0,633
17% Wiki+2-gram	0,701	0,539	0,609	0,741	0,553	0,633
50% Wiki+3-gram	0,736	0,544	0,626	0,799	0,572	0,667
33% Wiki+3-gram	0,736	0,544	0,626	0,792	0,567	0,661
25% Wiki+3-gram	0,748	0,553	0,636	0,786	0,563	0,656
17% Wiki+3-gram	0,703	0,535	0,608	0,738	0,544	0,626
25% Wiki+4-gram	0,744	0,553	0,634	0,788	0,572	0,663
17% Wiki+4-gram	0,699	0,535	0,606	0,739	0,548	0,629

Table B.18 – Résultats obtenus par nos modèles de langue lorsque notre système d'homophonie est fixé à 0,90 pour le corpus Corp3

ANNEXE C

Exemples de sorties proposées par le système

Nous proposons ci-dessous une dizaine d'exemples de phrases bruitées normalisées avec notre système. Cette normalisation a été effectuée en ne conservant que le chemin le mieux pondéré par notre système contextuel. Les normalisations proposées par notre système sont toutes précédées du symbole "→".

1. les prux son interessts
   → *les prix sont intéressants*

2. le syte et klair
   → *le site est clair*

3. les vandeurs son synpa
   → *les vendeurs sont sympa*

4. sa mzrche pa
   → *Sa marche pas*

5. je vous avez deja ecrit pour recevoir mon idantifiant pricipal par courrié postal.
   → *Je vous avez déjà écrit pour recevoir mon identifiant principal par courrier postal.*

6. mes adresse son correctement ortographieés
   → *mes adresse sont correctement orthographies*

7. depuis quelque jour mon creit par tout seul
   → *depuis quelque jour mon crédit par tout seul*

8. Désolée de se contre temps et des probléme que je vous aucasionne
   → *Désolée de ce contre temps et des problèmes que je vous occasionne*

9. exscellant servic mersi beaucoup
   → *excellant service merci beaucoup*

10. spoiling : noooooon pk il meuurt kom ca :'(
    → *spoiling : non pourquoi ils meurt comme ça :'(*

Bibliographie

Jacques Anis. *Parlez-vous texto ? Guide des nouveaux langages du réseau.* Le Cherche Midi, Paris, France, 2001.

Jacques Anis. Communication électronique scripturale et formes langagières. In *Actes des Quatrièmes Rencontres Réseaux Humain / Réseaux Technologiques*, Poitiers, France, 2003.

AiTi Aw, Min Zhang, Juan Xiao, et Jian Su. A phrase-based statistical model for sms text normalization. In *Proceedings of the COLING/ACL on Main Conference (COLING-ACL'06)*, pages 33–40, Sydney, Australie, 2006.

Dominic Balasuriya, Nicky Ringland, Joel Nothman, Tara Murphy, et James R. Curran. Named entity recognition in Wikipedia. In *Proceedings of the 2009 Workshop on The People's Web Meets NLP : Collaboratively Constructed Semantic Resources*, pages 10–18, Singapour, 2009.

Marion Baranes. Vers la mise en place d'un correcteur grammatical. Mémoire, Université Paris Diderot (Paris 7), 2011. Document confidentiel.

Marion Baranes. Vers la correction automatique de textes bruités : Architecture générale et détermination de la langue d'un mot inconnu. In *Actes de la conférence conjointe JEP-TALN-RECITAL 2012*, pages 95–108, Grenoble, France, 2012.

Marion Baranes et Benoit Sagot. A Language-Independent Approach to Extracting Derivational Relations from an Inflectional Lexicon. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2793–2799, Reykjavik, Islande, 2014.

Marco Baroni, Silvia Bernardini, Adriano Ferraresi, et Eros Zanchetta. The Wacky Wide Web : A Collection of Very Large Linguistically Processed Web-Crawled Corpora. *Language Resources and Evaluation*, 43(3) : 209–226, 2009.

Jan Niecisław Ignacy Baudouin de Courtenay. *Versuch einer Theorie phonetischer Alternationen : Ein Kapitel aus der Psychophonetik*. Trübner, Strasbourg, Empire allemand, 1895.

Richard Beaufort. Composition filtrée et marqueurs de règles de réécriture pour une distance d'édition flexible. application à la correction des mots hors-vocabulaire. *Traitement Automatique des Langues (T.A.L)*, 51(1) : 11–40, 2010.

Richard Beaufort, Sophie Roekhaut, Louise-Amélie Cougnon, et Cédrick Fairon. A Hybrid Rule/Model-Based Finite-State Framework for Normalizing SMS Messages. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'10)*, pages 770–779, Uppsala, Suède, 2010.

Kenneth R. Beesley. Language identifier : A computer program for automatic natural-language identification on on-line text. In *Languages as Crossroads : Proceedings of th 29th Annual Conference of the American Translators Association'88*, pages 47–54, 1988.

Delphine Bernhard. Apprentissage non supervisé de familles morphologiques : Comparaison de méthodes et aspects multilingues. *Traitement Automatique des Langues*, 51(2) : 11–39, 2010.

Daniel M. Bikel, Richard Schwartz, et Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3) : 211–231, 1999.

Charles R. Blair. A program for correcting spelling errors. *Information and Control*, 3(1) : 60–67, 1960.

Helena Blancafort, Gaëlle Recourcé, Javier Couto, Benoît Sagot, Rosa Stern, et Denis Teyssou. Traitement des inconnus : une approche systématique de l'incomplétude lexicale. In *Actes de la 17ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2010)*, pages 1–6, Montréal, Canada, 2010.

Léonard Bloomfield. A set of postulates for the science of language. *Language*, 2 (3) : 153–164, 1926.

Léonard Bloomfield. *Language*. H. Holt and Company, 1933.

Philippe Boissière, Jean-Leon Bouraoui, Frédéric Vella, Aurélie Lagarrigue, Mustapha Mojahid, Nadine Vigouroux, et Jean-Luc Nespoulous. Méthodologie d'annotation des erreurs en production écrite. principes et résultats préliminaires. In *Actes de la 14ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2007)*, pages 529–538, Toulouse, France, 2007.

Andrew Borthwick, John Sterling, Eugene Agichtein, et Ralph Grishman. Nyu : Description of the mene named entity system as used in muc-7. In *In Proceedings of the Seventh Message Understanding Conference (MUC-7)*, Fairfax, États-Unis, 1998.

Vincent Bouvier et Patrice Bellot. Amélioration d'un corpus de requêtes à l'aide d'une méthode non-supervisée. In *CORIA*, pages 373–382, Neuchâtel, Suisse, 2013.

Adriane Boyd. Pronunciation modeling in spelling correction for writers of English as a foreign language. In *Proceedings of Human Language Technologies : The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume : Student Research Workshop and Doctoral Consortium (SRWS'09)*, pages 31–36, Boulder, États-Unis, 2009.

Eric Brill et Robert C. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293, Hong Kong, Chine, 2000.

Éric Brunelle. Antidote : Correcteur, dictionnaire et plus. *BULAG (Bulletin de linguistique général et appliquée)*, 29 : 25–31, 2004.

Razvan Bunescu et Marius Pasca. Using encyclopedic knowledge for named entity disambiguation. In *Proceesings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, pages 9–16, Trente, Italie, 2006.

Gavin Burnage. Celex : A guide for users. Technical report, University of Nijmegen, Center for Lexical Information, 1990.

Teresa Cabré et Rogelio Nazar. Towards a new approach to the study of neology. In *Neology and Specialised Translation 4th Joint Seminar Organised by the CVC and Termisti*, Bruxelles, Belgique, 2011.

Aoife Cahill, Nitin Madnani, Joel Tetreault, et Diane Napolitano. Robust systems for preposition error correction using Wikipedia revisions. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, pages 507–517, Atlanta, États-Unis, 2013.

Andrew Carlson et Ian Fette. Memory-based context-sensitive spelling correction at web scale. In *Proceedings of the Sixth International Conference on Machine Learning and Applications (ICMLA'07)*, pages 166–171, Cincinnati, États-Unis, 2007.

Bruno Cartoni. Constance et variabilité de l'incomplétude lexicale. In *Actes de la 8ème Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RECITAL 2006)*, pages 661–669, Louvain, Belgique, 2006.

Bruno Cartoni. *De l'incomplétude lexicale en traduction automatique : vers une approche morphosémantique multilingue*. Thèse de doctorat, Université de Genève, 2008.

William B. Cavnar et John M. Trenkle. N-gram based text categorization. In *In Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval (SDAIR-94)*, pages 161–175, Las Vegas, États-Unis, 1994.

Simon Charest, Éric Brunelle, Jean Fontaine, et Bertrand Pelletier. Élaboration automatique d'un dictionnaire de cooccurrences grand public. In *Actes de la 14ème conférence sur le Traitement Automatique des Langues Naturelles*, pages 283–292, Toulouse, France, 2007.

Monojit Choudhury, Rahul Saraf, Vijit Jain, Sudeshna Sarkar, et Anupam Basu. Investigation and modeling of the structure of texting language. In *In Proceedings of the IJCAI Workshop on "Analytics for Noisy Unstructured Text Data*, pages 63–70, Hyderabad, Inde, 2007.

Grzegorz Chrupała. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL'14*, pages 680–686, Baltimore, États-Unis, 2014.

Kenneth W. Church et William A. Gale. Probability scoring for spelling correction. *Statistics and Computing*, 1(2) : 93–103, 1991.

Vincent Claveau et Marie-Claude L'Homme. Structuring terminology by analogy-based machine learning. In *Proceedings of the 7th International Conference on Terminology and Knowledge Engineering (TKE'05)*, Copenhague, Danemark, 2005.

Lionel Clément et Éric Villemonte de la Clergerie. Maf : a morphosyntactic annotation frame- work. In *Proceedings of the 2nd Language and Technology Conference (LTC'05)*, pages 90–94, Poznań, Pologne, 2005.

Danish Contractor, Tanveer A. Faruquie, et L. Venkata Subramaniam. Unsupervised cleansing of noisy text. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING'10)*, pages 189–196, Pékin, Chine, 2010.

Paul Cook et Suzanne Stevenson. An unsupervised model for text message normalization. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity (CALC'09)*, pages 71–78, Boulder, États-Unis, 2009.

Blandine Courtois. Un système de dictionnaires électroniques pour les mots simples du français. *Langue Française*, 87, 1990.

Mathias Creutz et Krista Lagus. Inducing the morphological lexicon of a natural language from unannotated text. In *Proceedings of the International and Inter-*

disciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05), pages 106–113, Espoo, Finlande, 2005.

Silviu Cucerzan et Eric Brill. Spelling correction as an iterative process that exploits the collective knowledge of Web users. In Dekang Lin et Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 293–300, Barcelone, Espagne, 2004.

Georgette Dal et Fiammetta Namer. Génération et analyse automatiques de ressources lexicales construites utilisables en recherche d'informations. *Traitement Automatique des Langues*, 41(2) : 423–446, 2000.

Robert Dale et Adam Kilgarriff. Helping our own : The hoo 2011 pilot shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*, ENLG'11, pages 242–249, Nancy, France, 2011.

Robert Dale, Ilya Anisimoff, et George Narroway. Hoo 2012 : A report on the preposition and determiner error correction shared task. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*, pages 54–62, Montréal, Canada, 2012.

Fred Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3) : 171–176, 1964.

Hal Daumé III. Notes on CG and LM-BFGS optimization of logistic regression. http ://www.umiacs.umd.edu/ hal/docs/daume04cg-bfgs.pdf, 2004.

Pascal Denis et Benoît Sagot. Coupling an annotated corpus and a lexicon for state-of-the-art pos tagging. *Language Resources and Evaluation*, 46(4) : 721–736, 2012.

Etienne Denoual. Analogical translation of unknown words in a statistical machine translation framework. In *Proceedings of Machine Translation Summit XI, European Association for Machine Translation*, pages 135–141, Copenhague, Danemark, 2007.

Jean-Jacques Didier, Olivier Hambursin, Philippe Moreau, et Michel Seron. Le français m'a tuer. In *Actes du colloque - L'orthographe française à l'épreuve du supérieur*, Bruxelles, Belgique, 2005.

Anne Dister et Cédrick Fairon. Extension des ressources lexicales grâce à un corpus dynamique. *Lexicometrica*, Thema 7, 2004.

Ted Dunning. Statistical Identification of Language. In *Technical report CRL MCCS-94-273*, Computing Research Lab, New Mexico State University, 1994.

Camille Dutrey, Anne Peradotto, et Chloé Clavel. Analyse de forums de discussion pour la relation clients : du Text Mining au Web Content Mining. In *Actes des 11èmes Journées internationnales d'Analyse statistique des Données Textuelles (JADT'12)*, Liège, Belgique, 2012.

Asif Ekbal, Eva Sourjikova, Anette Frank, et Simone Paolo Ponzetto. Assessing the challenge of fine-grained named entity recognition and classification. In *Proceedings of the 2010 Named Entities Workshop (NEWS'10)*, pages 93–101, Uppsala, Suède, 2010.

Cédrick Fairon, Jean Klein, et Sébastien Paumier. Le langage sms : révélateur d'1compétence. In *"Le français m'a tuer". Actes du colloque "L'orthographe française à l'épreuve du supérieur"*, Bruxelles, Belgique, 2006.

Ingrid Falk, Delphine Bernhard, et Christophe Gérard. From Non Word to New Word : Automatically Identifying Neologisms in French Newspapers. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Islande, 2014.

Noura Farra, Nadi Tomeh, Alla Rozovskaya, et Nizar Habash. Generalized character-level spelling error correction. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, (ACL'2014)*, pages 161–167, Baltimore, États-Unis, 2014.

Stefano Federici, Simonetta Montemagni, et Vito Pirrelli. Inferring semantic similarity from distributional evidence : an analogy-based approach to word sense disambiguation. In *Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, pages 90–97, 1997.

Jennifer Foster. "cba to check the spelling" : Investigating parser performance on discussion forum posts. In *Human Language Technologies : The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 381–384, Los Angeles, États-Unis, 2010.

Nathalie Friburger. *Reconnaissance automatique des noms propres : application à la classification automatique de textes journalistiques*. Thèse de doctorat, Université François-Rabelais Tours, France, 2002.

Jianfeng Gao, Xiaolong Li, Daniel Micol, Chris Quirk, et Xu Sun. A large scale ranker-based system for search query spelling correction. In *Proceedings of the 23rd International Conference on Computational Linguistics (CoLing'10)*, pages 358–366, Pékin, Chine, 2010.

Claudia Gdaniec, Esmé Manandise, et Michael C. McCord. Derivational Morphology to the Rescue : How It Can Help Resolve Unfound Words in MT. In *Machine Translation summit VIII, European Association for Machine Translation*, Saint-Jacques-de-Compostelle, Espagne, 2001.

Andrew R. Golding et Dan Roth. Applying Winnow to Context-Sensitive Spelling Correction. In *Proceedings of the 13th International Conference on Machine Learning (ICML'96)*, pages 182–190, Bari, Italie, 1996.

Andrew R. Golding et Yves Schabes. Combining Trigram-based and Feature-based Methods for Context-sensitive Spelling Correction. In *Proceedings of*

the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96), pages 71–78, Santa Cruz, États-Unis, 1996.

John Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2) : 153–198, 2001.

Julien Gosme et Yves Lepage. Structure des trigrammes inconnus et lissage par analogie. In *Actes de la 18ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2011)*, Montpellier, France, 2011.

Fabrizio Gotti, Philippe Langlais, Guy Lapalme, Simon Charest, et Éric Brunelle. Atténuation des surdétections d'un correcteur grammatical de qualité commerciale. *TAL*, 53(3) : 33–66, 2012.

Stephan Gouws, Dirk Hovy, et Donald Metzler. Unsupervised mining of lexical variants from noisy text. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 82–90, Édimbourg, Royaume-Uni, 2011.

Gregory Grefenstette. Comparing two language identification schemes. In *Proceedings of the 3rd International Conference on the Statistical Analysis of Textual Data (JADT'95)*, Rome, Italie, 1995.

Émilie Guimier de Neef et Sébastien Fessard. Évaluation d'un système de transcription de SMS. In *Proceedings of the 26th International Conference on Lexis and Grammar*, Bonifacio, France, 2007.

Bo Han et Timothy Baldwin. Lexical normalisation of short text messages : Makn sens a# twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies (HLT'11)*, pages 368–378, Portland, États-Unis, 2011.

Martin Haspelmath. The indeterminacy of word segmentation and the nature of morphology and syntax. *Folia Linguistica*, 45(1) : 31–80, 2011.

Hany Hassan et Arul Menezes. Social text normalization using contextual graph random Walks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 1577–1586, Sofia, Bulgarie, 2013.

Nabil Hathout. From Wordnet to CELEX : acquiring morphological links from dictionaries of synonyms. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, îles Canaries, Espagne, 2002.

Nabil Hathout. Acquistion of the morphological structure of the lexicon based on lexical similarity and formal analogy. In *Proceedings of the 3rd Textgraphs Workshop on Graph-Based Algorithms for Natural Language Processing (CoLing'08)*, pages 1–8, Manchester, Royaume-Uni, 2008.

Nabil Hathout. Morphonette : a morphological network of French. *CoRR*, 2010.

Nabil Hathout et Fiammetta Namer. Règles et paradigmes en morphologie informatique lexématique. In *Actes de la 18ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2011)*, pages 215–220, Montpellier, France, 2011.

Nabil Hathout, Franck Sajous, et Basilio Calderone. GLÀFF, a Large Versatile French Lexicon. In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Islande, 2014.

Kenneth Heafield. KenLM : faster and smaller language model queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197, Édimbourg, Royaume-Uni, 2011.

Norman C. Ingle. A language identification table. *The Incorporated Linguist*, 15 (4) : 98–101, 1976.

Aminul Islam et Diana Inkpen. Real-word spelling correction using Google Web IT 3-grams. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP'09)*, pages 1241–1249, Singapour, 2009.

Fabrice Issac. Cybernéologisme : Quelques outils informatiques pour l'identification et le traitement des néologismes sur le web. *Langages*, 183(3), 2011.

Christine Jacquet-Pfau. Correcteurs orthographiques et grammaticaux - quel(s) outil(s) pour quel rédacteur ? *Revue française de linguistique appliquée*, 2 : 81–94, 2001.

Stephen Johnson. Solving the problem of language recognition. In *Technical report*, School of Computer Studies, University of Leeds, Angleterre, 1993.

Marcin Junczys-Dowmunt et Roman Grundkiewicz. The AMU system in the CoNLL-2014 shared task : Grammatical error correction by data-intensive and feature-rich statistical machine translation. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning : Shared Task*, pages 25–33, Baltimore, États-Unis, 2014.

Jun'ichi Kazama et Kentaro Torisawa. Exploiting Wikipedia as external knowledge for named entity recognition. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*, pages 698–707, Prague, République tchèque, 2007.

Mark D. Kernighan, Kenneth Ward Church, et William A. Gale. A Spelling Correction Program Based on a Noisy Channel Model. In *Proceedings of the 13th International Conference on Computational Linguistics (CoLing 1990)*, pages 205–210, Helsinki, Finlande, 1990.

Catherine Kobus, François Yvon, et Géraldine Damnati. Transcrire les SMS comme on reconnaît la parole. In *Actes de la 15ème conférence sur le Trai-*

tement Automatique des Langues Naturelles (TALN 2008), pages 128–138, Avignon, France, 2008.

Karen Kukich. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4) : 377–439, 1992.

Djamel Eddine Lachachi. Problématique du mot comme unité linguistique. In *L'unité en sciences du langage, Actes des 9èmes journée scientifique du réseau thématique*, pages 11–26, Paris, France, 2011.

John D. Lafferty, Andrew McCallum, et Fernando C. N. Pereira. Conditional random fields : Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML'01)*, pages 282–289, San Francisco, États-Unis, 2001.

Philippe Langlais. Étude quantitative de liens entre l'analogie formelle et la morphologie constructionnelle. In *Actes de la 16è Conférence sur le Traitement Automatique des Langues Naturelles (TALN'09)*, Senlis, France, 2009.

Philippe Langlais. Mapping source to target strings without alignment by analogical learning : A case study with transliteration. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 684–689, Sofia, Bulgarie, 2013.

Philippe Langlais et Alexandre Patry. Enrichissement d'un lexique bilingue par analogie. In *Actes de la 14e conférence sur le Traitement Automatique des Langues Naturelles (TALN 2007)*, pages 101–110, Toulouse, France, 2007.

Philippe Langlais et Alexandre Patry. Enrichissement d'un lexique bilingue par apprentissage analogique. *Traitement automatique des langues*, 49 : 13–40, 2008.

Philippe Langlais et François Yvon. Scaling up analogical learning. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, pages 51–54, Manchester, Royaume-Uni, 2008.

Philippe Langlais, François Yvon, et Pierre Zweigenbaum. Improvements in analogical learning : Application to translating multi-terms of the medical domain. In *Proceedings of the Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 487–495, Athènes, Grèce, 2009.

Dominique Laurent. *Les vraies difficultés du français au XXI^e siècle*. Synapse Développement, 2012.

Dominique Laurent, Sophie Nègre, et Patrick Séguéla. Apport des cooccurrences à la correction et à l'analyse syntaxique. In *Actes de la 16ème conférence sur le Traitement Automatique des Langues Naturelles*, Senlis, France, 2009a.

Dominique Laurent, Sophie Nègre, et Patrick Séguéla. L'analyseur syntaxique cordial dans passage. In *Actes de la 16ème conférence sur le Traitement Automatique des Langues Naturelles*, Senlis, France, 2009b.

Jean-François Lavallée et Philippe Langlais. Moranapho : un système multilingue d'analyse morphologique fondé sur l'analogie formelle. *Traitement Automatique des Langues*, 52(2) : 17–44, 2011.

John Lee et Stephanie Seneff. Correcting misuse of verb forms. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (AC'08)*, pages 174–182, Columbus, États-Unis, 2008.

Alise Lehmann et François Martin-Berthet. *Lexicologie, Sémantique, Morphologie, Lexicographie, 4ème éd.* Armand Colin, 2013.

Yves Lepage. Solving analogies on Words : An algorithm. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, pages 728–735, Québec, Canada, 1998.

Yves Lepage. Proportional analogy in Written language data. *Language Production, Cognition, and the Lexicon, Springer International Publishing*, pages 151–173, 2015.

Yves Lepage et Etienne Denoual. Purest ever example-based machine translation : Detailed presentation and assessment. *Machine Translation*, 19(3-4) : 251–282, 2005.

Yves Lepage, Adrien Lardilleux, Julien Gosme, et Jean-Luc Manguin. The greyc machine translation system for the iwslt 2008 evaluation campaign. In *Proceedings of the 5th International Workshop on Spoken Language Translation (IWSLT 2008)*, Waikiki, États-Unis, 2008.

Yves Lepage, Adrien Lardilleux, et Julien Gosme. The greyc translation memory for the iwslt 2009 evaluation campaign : one step beyond translation memory. In *The 6th International Workshop on Spoken Language Translation (IWSLT'09)*, pages 45–49, Tokyo, Japon, 2009.

Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10(8) : 707, 1966.

Mu Li, Yang Zhang, Muhua Zhu, et Ming Zhou. Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (ACL-CoLing'06)*, pages 1025–1032, Syndey, Australie, 2006.

Fei Liu, Fuliang Weng, Bingqing Wang, et Yang Liu. Insertion, deletion, or substitution ? : Normalizing text messages Without pre-categorization nor supervision. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies (HLT'11)*, pages 71–76, Portland, États-Unis, 2011.

Fei Liu, Fuliang Weng, et Xiao Jiang. A broad-coverage normalization system for social media language. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL'12)*, pages 1035–1044, Jeju, Corée du sud, 2012.

Julie Beth Lovins. *Development of a stemming algorithm.* MIT Information Processing Group, Electronic Systems Laboratory, 1968.

Marco Lui et Timothy Baldwin. Accurate language identification of twitter messages. In *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM EACL'2014)*, pages 17–25, Göteborg, Suède, 2014.

Pierre Magistry. *Unsupervised Word Segmentation and Wordhood Assessment : The case for Mandarin Chinese.* Thèse de doctorat, Université Paris Diderot (Paris 7), 2013.

Lidia Mangu et Eric Brill. Automatic Rule Acquisition for Spelling Correction. In *Proceedings of ICML'97*, pages 187–194, Nashville, États-Unis, 1997.

André Martinet. *Syntaxe générale.* Collection U. Armand Colin, 1985.

Bruno Martins et Mário J. Silva. Language identification in web pages. In *Proceedings of the 2005 ACM symposium on Applied computing (SAC'05)*, pages 764–768, New York, États-Unis, 2005. ACM Press.

Michel Mathieu-Colas. *Flexion des noms et des adjectifs composés : principes de codage.* Lexiques, Dictionnaires, Informatique (LDI), 2010.

Denis Maurel. Les mots inconnus sont-ils des noms propres. In *Actes des Journées internationales d'Analyse statistique des Données Textuelles (JADT'04)*, pages 776–784, Louvain La Neuve, Belgique, 2004.

Denis Maurel. Prolexbase : a multilingual relational lexical database of proper names. In *Proceedings of the 6th edition of the Language Resources and Evaluation Conference (LREC'08)*, pages 334–338, Marrakech, Maroc, 2008.

Aurélien Max et Guillaume Wisniewski. Mining naturally-occurring corrections and paraphrases from Wikipedia's revision history. In *Proceedings of the 7th edition of the Language Resources and Evaluation Conference (LREC'10)*, pages 3143–3148, La Valette, Malte, 2010.

Andrew McCallum et Wei Li. Early results for named entity recognition with conditional random fields, feature induction and Web-enhanced lexicons. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 188–191, Edmonton, Canada, 2003.

Andrew McCallum, Dayne Freitag, et Fernando C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*, pages 591–598, Stanford, États-Unis, 2000.

David D. McDonald. Internal and external evidence in the identification and semantic categorization of proper names. In Branimir Boguraev et James Pustejovsky, editors, *Corpus Processing for Lexical Acquisition*, pages 21–39. MIT Press, 1996.

Antoine Meillet. *Linguistique historique et linguistique générale*. Slatkine, 1921.

Andrei Mikheev. Automatic rule induction for unknown-word guessing. *Computational Linguistics*, 23(3) : 405–423, 1997.

Andrei Mikheev. Document centered approach to text normalization. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval SIGIR'00*, pages 136–143, Athènes, Grèce, 2000.

Andrei Mikheev, Marc Moens, et Claire Grover. Named entity recognition without gazetteers. In *Proceedings of the Ninth Conference on European Chapter of the Association for Computational Linguistics (EACL'99)*, pages 1–8, Bergen, Norvège, 1999.

Scott Miller, Jethran Guinness, et Alex Zamanian. Name tagging with word clusters and discriminative training. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'2004)*, pages 337–342, Boston, États-Unis, 2004.

Roger Mitton. *English Spelling and the computer*. London :Longman, 1996.

Roger Mitton. Ordering the suggestions of a spellchecker without using context. *Natural Language Engineering*, 15(2) : 173–192, 2009.

Roger Mitton. Fifty years of spellchecking. *Writting Systems Research*, 2(1) : 1–7, 2010.

Miguel A. Molinero, Benoît Sagot, et Lionel Nicolas. A morphological and syntactic wide-coverage lexicon for spanish : The leffe. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2009)*, pages 264–269, Borovets, Bulgarie, 2009.

Fabienne Moreau, Vincent Claveau, et Pascale Sébillot. Automatic morphological query expansion using analogy-based machine learning. In *Proceedings of 29th European Conference on IR Research (ECIR'07)*, pages 222–233, Rome, Italie, 2007.

Preslav Nakov, Yury Bonev, Galia Angelova, Evelyn Cius, et Walther von Hahn. Guessing morphological classes of unknown German nouns. In *Recent Advances in Natural Language Processing (RANLP'2003)*, pages 319–326, Borovets, Bulgarie, 2003.

Fiammetta Namer. FLEMM : un analyseur flexionnel du français à base de règles. *Traitement Automatique des Langues*, 41(2) : 523–547, 2000.

Rogelio Nazar et Irene Renau. Google books n-gram corpus used as a grammar checker. In *Proceedings of the Second Workshop on Computational Linguistics and Writing (CLW 2012) : Linguistic and Cognitive Aspects of Document Creation and Document Engineering*, EACL 2012, pages 27–34, Avignon, France, 2012.

Mar Ndiaye et Anne Vandeventer Faltin. Correcteur orthographique adapté à l'apprentissage du français. *BULAG (Bulletin de linguistique général et appliquée)*, 29 : 117–134, 2004.

Sylvain Neuvel et Sean A. Fulop. Unsupervised learning of morphology without morphemes. *CoRR*, 2002.

Boris New. Lexique3 : Une nouvelle base de données lexicales. In *Actes de la 13ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2006)*, pages 892–900, Louvain, Belgique, 2006.

Boris New, Christophe Pallier, Ludovic Ferrand, et Matos Rafaël. Une base de données lexicales du français contemporain sur internet : Lexique$^{\text{TM}}$. *L'année psychologique*, 101(3) : 447–462, 2001.

Boris New, Christophe Pallier, Marc Brysbaert, et Ludovic Ferrand. Lexique 2 : A new french lexical database. *Behavior Research Methods, Instruments, & Computers*, 36(3) : 516–524, 2004.

Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, et Joel R. Tetreault. The CoNLL-2013 shared task on grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning : Shared Task, (CoNLL 2013)*, pages 1–12, Sofia, Bulgarie, 2013.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, et Christopher Bryant. The CoNLL-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning : Shared Task (CoNLL 2014)*, pages 1–14, Baltimore, États-Unis, 2014.

Joel Nothman, Tara Murphy, et James R. Curran. Analysing Wikipedia and gold-standard corpora for ner training. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL'09)*, pages 612–620, Athènes, Grèce, 2009.

Damien Nouvel. *Reconnaissance des entités nommées par exploration de règles d'annotation.* Thèse de doctorat, Université François Rabelais Tours, 2012.

Kemal Oflazer. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1) : 73–89, 1996.

Y. Albert Park et Roger Levy. Automated whole sentence grammar correction using a noisy channel model. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics : Human Language Technologies (ACL-HTL'11)*, pages 934–944, Portland, États-Unis, 2011.

Jennifer Pedler et Roger Mitton. A large list of confusion sets for spellchecking assessed against a corpus of real-word errors. In *In Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC'10)*, pages 755–762, La Valette, Malte, 2010.

Deana Pennell et Yang Liu. A character-level machine translation approach for normalization of SMS abbreviations. In *Fifth International Joint Conference on Natural Language Processing, IJCNLP'11*, pages 974–982, Chiang Mai, Thaïlande, 2011.

Maud Pironneau, Éric Brunelle, et Simon Charest. Pronoun anaphora resolution for automatic correction of grammatical errors (correction automatique par résolution d'anaphores pronominales) [in french]. In *Proceedings of TALN 2014*, pages 113–124, Marseille, France, 2014. Association pour le Traitement Automatique des Langues.

Alain Polguère. *Lexicologie et sémantique lexicale. Notions fondamentales, Nouvelle édition revue et augmentée.* Les Presses de l'Université de Montréal, 2008.

Joseph J. Pollock et Antonio Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27(4) : 358–368, 1984.

Bernard Pottier. *Introduction à l'étude des structures grammaticales fondammentales.* Publications linguistiques de la Faculté des Lettres et des Sciences Humaines de Nancy I., Nancy, France, 1962.

Jean Pruvost et Jean-François Sablayrolles. *Les néologismes, 2ème ed.* Les Presses Universitaire de France, 2012.

Chen Qing, Li Mu, et Zhou Ming. Improving Query Spelling Correction Using Web Search Results. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*, pages 181–189, Prague, République tchèque, 2007.

Christian Raymond et Julien Fayolle. Reconnaissance robuste d'entités nommées sur de la parole transcrite automatiquement. In *Conférence Traitement automatique des langues naturelles, TALN'10*, Montréal, Québec, Canada, 2010.

Xiaobo Ren et Francois Perrault. The typology of unknown words : an experimental study of two corpora. In *Proceedings of the 14th Conference on Computational Linguistics (COLING'92)*, pages 408–414, Nantes, France, 1992.

Antoinette Renouf. A word in time : First findings from the investigation of dynamic text. *English Language Corpora : Design, Analysis and Exploitation*, pages 279–288, 1993.

Martin Reynaert. Multilingual Text Induced Spelling Correction. In *Proceedings of the Workshop on Multilingual Linguistic Ressources at the 20th International Conference on Computational Linguistics (CoLing'04)*, pages 117–117, Genève, Suisse, 2004.

Rafik Rhouma et Philippe Langlais. Fourteen light tasks for comparing analogical learning and phrased-based machine translation. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING-2014)*, pages 444–454, Dublin, Irlande, 2014.

Christoph Ringlstetter, Klaus U. Schulz, et Stoyan Mihov. Orthographic errors in web pages : Toward cleaner web corpora. *Computational Linguistics*, 32(3) : 295–340, 2006.

Laurent Romary, Susanne Salmon-Alt, et Gil Francopoulo. Standards going concrete : from lmf to morphalou. In *Proceedings of the Workshop on Enhancing and Using Electronic Dictionaries*, pages 22–28, Genève, Suisse, 2004.

Alla Rozovskaya. *Automated methods for text correction.* Thèse de doctorat, Université de l'Illinois, 2013.

Alla Rozovskaya et Dan Roth. Generating Confusion Sets for Context-Sensitive Error Correction. In *Proceedings of the 2010 Conference on Empirical Methods on Natural Language Processing (EMNLP'10)*, pages 961–970, MIT Stata Center, États-Unis, 2010.

Alla Rozovskaya, Dan Roth, et Vivek Srikumar. Correcting grammatical verb errors. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2014)*, pages 358–367, Göteborg, Suède, 2014.

Jean-François Sablayrolles. Néologie et dictionnaire(s) comme corpus d'exclusion. In *Néologie et terminologie dans les dictionnaires*, Lexica, pages 19–36. Champion, 2008.

Ivan A. Sag, Timothy Baldwin, Francis Bond, Ann Copestake, et Dan Flickinger. Multi-word expressions : A pain in the neck for NLP. In *Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics (CICLing'2002)*, pages 1–15, Mexico, Mexique, 2002.

Benoît Sagot. The Le*fff*, a freely available and large-coverage morphological and syntactic lexicon for French. In *Proceedings of the Seventh international conference on Language Resources and Evaluation (LREC 2010)*, pages 2744–2751, La Valette, Malte, 2010.

Benoît Sagot. DeLex, a freely-avaible, large-scale and linguistically grounded morphological lexicon for German. In *Proceedings of the 9th international conference on Language Resources and Evaluation (LREC 2014)*, pages 2778–2784, Reykjavik, Islande, 2014.

Benoît Sagot. *Informatiser le lexique : Modélisation, développement et utilisation de lexiques morphologiques, syntaxiques et sémantiques.* Habilitation à diriger des recherches en mathématiques et en informatique, Université Paris–Diderot (Paris 7), 2016. à paraître.

Benoit Sagot et Pierre Boullier. SxPipe 2 : architecture pour le traitement présyntaxique de corpus bruts. *Traitement Automatique des Langues*, 49(2) : 155–188, 2008.

Benoît Sagot et Kata Gábor. Détection et correction automatique d'entités nommées dans des corpus OCRisés. In *Traitement Automatique du Langage Naturel 2014*, pages 437–442, Marseille, France, 2014.

Benoît Sagot et Rosa Stern. Aleda, a free large-scale entity database for French. In *Proceedings of the 8th edition of the Language Resources and Evaluation Conference (LREC'12)*, pages 1273–1276, Istanbul, Turquie, 2012.

Benoît Sagot, Damien Nouvel, Virginie Mouilleron, et Marion Baranes. Extension dynamique de lexiques morphologiques pour le français à partir d'un flux textuel. In *Actes de la 20ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2013)*, pages 407–420, Les Sables d'Olonne, France, 2013.

Franck Sajous, Nabil Hathout, et Basilio Calderone. Ne jetons pas le Wiktionnaire avec l'oripeau du web ! études et réalisations fondées sur le dictionnaire collaboratif. In *Actes du 4e Congrès Mondial de Linguistique Française (CMLF 2014)*, pages 663–680, Berlin, Allemagne, 2014.

Ferdinand de Saussure. *Cours de Linguistique Générale.* Armand Colin, 1916.

Yves Scherrer et Benoît Sagot. A language-independent and fully unsupervised approach to lexicon induction and part-of-speech tagging for closely related languages. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 502–508, Reykjavik, Islande, 2014.

Djamé Seddah, Benoît Sagot, Marie Candito, Virginie Mouilleron, et Vanessa Combet. The french social media bank : a treebank of noisy user generated content. In *Proceedings of the 24th International Conference on Computational Linguistics (CoLing 2012)*, pages 2441–2457, Bombay, Inde, 2012.

Hongsuck Seo, Jonghoon Lee, Seokhwan Kim, Kyusong Lee, Sechun Kang, et Gary Geunbae Lee. A meta learning approach to grammatical error correction. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, ACL'12, pages 328–332, Jeju, Corée du Sud, 2012.

Penelope Sibun et Jeffrey C. Reynar. Language identification : Examining the issues. In *Proceedings of the 5th Symposium on Document Analysis an Information Retrieval (SDAIR-96)*, pages 125–135, Las Vegas, États-Unis, 1996.

Max Silberztein. Le dictionnaire électronique des mots composés. *Langue Française*, 87 : 71–83, 1990.

Max Silberztein. *Dictionnaires électroniques et analyse automatique de textes – Le système Intex*. Elsevier Masson, 1993.

Richard Sproat, Alan W. Black, Stanley F. Chen, Shankar Kumar, Mari Ostendorf, et Christopher Richards. Normalization of non-standard words. *Computer Speech & Language*, 15(3) : 287–333, 2001.

Herman Stehouwer et Menno van Zaanen. Language models for contextual error detection and correction. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL-09) - Workshop CLAGI*, pages 41–48, Athènes, Grèce, 2009.

Rosa Stern et Benoît Sagot. Détection et résolution d'entités nommées dans des dépêches d'agence. In *Actes de la 17e conférence sur le Traitement Automatique des Langues Naturelles (TALN 2010)*, Montréal, Canada, 2010.

Nicolas Stroppa et François Yvon. An analogical learner for morphological analysis. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL'2005)*, pages 120–127, Ann Arbor, États-Unis, 2005.

Nicolas Stroppa et François Yvon. Du quatrième de proportion comme principe inductif : une proposition et son application à l'apprentissage de la morphologie. *Traitement Automatique des Langues*, 47(1) : 33–59, 2006.

Philippe Suignard et Sophiane Kerroua. Utilisation de contextes pour la correction automatique ou semi-automatique de réclamations clients. In *Actes de la 20ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2013)*, pages 699–706, Les Sables d'Olonne, France, 2013.

Raymond Hendy Susanto, Peter Phandi, et Hwee Tou Ng. System combination for grammatical error correction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014) A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 951–962, Doha, Qatar, 2014.

Ludovic Tanguy et Nabil Hathout. Webaffix : un outil d'acquisition morphologique dérivationnelle à partir du Web. In *Actes de la 9ème conférence sur le*

Traitement Automatique des Langues Naturelles (TALN 2002), pages 245–254, Nancy, France, 2002.

Anaïs Tatossian. Clavardage et orthographe. *Correspondance, Centre collégial de développement de matériel didactique (CCDMD)*, 16(2) : 26–29, 2011.

Jean Tournier. *Introduction descriptive à la lexicogénétique de l'anglais contemporain.* Slatkine, Genève, 1985.

Jean Tournier. *Précis de lexicologie anglaise, réédition.* Ellipses, 2004.

Kristina Toutanova et Robert C. Moore. Pronunciation Modeling for Improved Spelling Correction. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, pages 144–151, Philadelphie, États-Unis, 2002.

Séverine Vienney et Cirpian Melian. La correction automatique du langage des nouvelles formes de communication écrite. *Bulag*, 29 : 179–192, 2004.

Jean Véronis. Computerized correction of phonographic errors. *Computers and the Humanities*, 22(1) : 43–56, 1988.

Jean Véronis et Émilie Guimier de Neef. *Compréhension automatique des langues et interaction*, Chapitre : Le traitement des nouvelles formes de communication écrite, pages 227–248. Hermès Science, Paris, 2006.

Géraldine Walther et Benoît Sagot. Problèmes d'intégration morphologique d'emprunts d'origine anglaise en français. In *30th International Conference on Lexis and Grammar*, Nicosie, Chypre, 2011.

Casey Whitelaw, Ben Hutchinson, Grace Y. Chung, et Gerard Ellis. Using the web for language independent spellchecking and autocorrection. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP'09)*, pages 890–899, Singapour, 2009.

Guillaume Wisniewski, Aurélien Max, et François Yvon. Recueil et analyse d'un corpus écologique de corrections orthographiques extrait des révisions de Wikipédia. In *Actes de la 17ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2010)*, Montréal, Canada, 2010.

Wei Xu, Joel Tetreault, Martin Chodorow, Ralph Grishman, et Le Zhao. Exploiting Syntactic and Distributional Information for Spelling Correction with Web-Scale N-gram Models. In *Proceedings of the 2011 Conference on Empirical Methods on Natural Language Processing (EMNLP'11)*, pages 1291–1300, Édimbourg, Royaume-Uni, 2011.

Yi Yang et Jacob Eisenstein. A log-linear model for unsupervised text normalization. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP'13)*, pages 61–72, Seattle, États-Unis, 2013.

François Yvon. Pronouncing unknown words using multi-dimensional analogies. In *Proceedings of the Conference of the European Speech Communication Association (EuroSpeech)*, volume 1, pages 199–202, Budapest, Hongrie, 1999.

François Yvon. Paradigmatic cascades : A linguistically sound model of pronunciation by analogy. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 428–435, Madrid, Espagne, 1997.

François Yvon. `spellChecker` : un système de correction automatique fondé sur des automates probabilistes. Livrable du Projet TRACE (ANR-09-CORD-023), 2011. Accessible à l'URL http ://anrtrace.limsi.fr/dev/Anr_trace_-_lot3.pdf.

François Yvon, Nicolas Stroppa, Arnaud Delhay, et Laurent Miclet. Solving analogical equations on words, 2004.

Marcos Zampieri. Bag-of-words to distinguish similar languages : How efficient are they ? In *Proceedings of the 14th IEEE International Symposium on Computational Intelligence and Informatics (CINTI2013)*, pages 37–41, Budapest, Hongrie, 2013.

Torsten Zesch. Detecting malapropisms using measures of contextual fitness. *TAL*, 53(3) : 11–31, 2012.

Paul Zumthor. Introduction aux problèmes de l'archaïsme. *Cahiers de l'Association internationale des études francaises*, 19 : 11–26, 1967.