Segmentation automatique d'un corpus de français oral en unités macrosyntaxiques

Ilaine Wang

Mémoire dirigé par Sylvain Kahane et Isabelle Tellier

Université Paris III - Sorbonne Nouvelle
Master de Sciences du Langage, Spécialité Ingénierie Linguistique

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

L'étude de l'interface entre la syntaxe et la prosodie en traitement automatique des langues intéresse en premier lieu les spécialistes de la synthèse de la parole, qui s'appuient sur la structure syntaxique d'un énoncé pour en délimiter la structure prosodique. Est alors établi en particulier un lien entre la frontière droite des unités syntaxiques et la frontière droite des groupes accentuels [Selkirk 2000]. Une analyse intonosyntaxique poussée sur un échantillon du CFPP2000 [1] permet également d'établir une corrélation forte entre le découpage prosodique et le découpage syntaxique en mettant en évidence la concomitance entre certaines frontières prosodiques (*paquets intonatifs*) et les frontières d'unités illocutoires [Lacheret-Dujour *et al.* 2011].

La problématique qui émerge de ce contexte porte sur la possibilité de dégager les unités majeures de ces données orales mises par écrit, et il se trouve que sur ce type de données, certaines informations objectives peuvent être directement exploitées : les propriétés acoustiques des sons, et plus précisément la durée, les variations de la fréquence fondamentale. Avec les moyens technologiques dont on dispose aujourd'hui, d'autres informations (morpho-syntaxiques et prosodiques) peuvent également être calculées directement à partir d'une simple transcription ou des propriétés acoustiques, mais avec, bien évidemment, des réserves concernant l'exactitude de ces prévisions.

Une seconde question se pose alors : avec quel(s) type(s) d'indices segmente-t-on automatiquement un texte oral en unités majeures ? L'hypothèse suggérée par les travaux précédents suppose que les indices prosodiques devraient être favorisées mais d'autres types d'information tels que les indices lexicaux, morphologiques et morpho-syntaxiques sont également fréquemment exploités en traitement automatique des langues, et se sont notamment avérés être une ressource pertinente pour la définition d'une structure syntaxique superficielle du français écrit [Constant *et al.* 2011], mais aussi, parmi les travaux réalisés sur l'anglais oral, en reconnaissance automatique de la parole [Liu *et al.* 2006] et en segmentation automatique [Stolcke et Shriberg 1996]. Enfin, dans une perspective de réapplication des outils sur de nouvelles données, on est également amené à se demander dans quelle mesure il est possible de segmenter avec uniquement des indices récupérées automatiquement.

Nous nous efforcerons de répondre à ces questions par la mise en place de séries d'expériences utilisant des classifieurs supervisés, plus précisément de la famille des arbres de décision qui ont déjà été exploités pour segmenter des données orales par Shriberg *et al.* [2000] pour leurs qualités : ils ont l'avantage d'effectuer une sélection des indices basée

1. Corpus de Français Parlé Parisien des années 2000, que l'on peut retrouver sur http://cfpp2000.univ-paris3.fr/ et présenté dans Branca-Rosoff *et al.* [2000]. Par ailleurs, certains échantillons du CFPP2000 font partie des sources externes utilisées pour l'ANR Rhapsodie. Il est donc possible de trouver des extraits d'échantillons issus de ce corpus dans ce mémoire.

sur l'entropie, mais aussi celui de rendre lisibles les résultats et les modèles utilisés. La segmentation d'un corpus oral en unités macrosyntaxiques est une étape préliminaire à une analyse microsyntaxique, mais aussi à d'autres tâches de traitement automatique plus complexes comme la reconnaissance automatique de la parole ou encore le résumé et la traduction automatiques [Ostendorf *et al.* 2008].

Après une brève présentation des données sur lesquelles nous avons travaillé et en particulier des annotations disponibles, nous aborderons la tâche de la segmentation automatique sous l'angle de la classification supervisée d'espaces intermots en frontières d'unités macrosyntaxiques (chapitre 2). Cette reformulation a donné lieu au développement d'une chaîne de traitements incluant des expérimentations dont les résultats sont détaillés (chapitre 3), avant d'être évalués et analysés sous différents angles (chapitre 4). Enfin, la dernière partie recensera les questions et les possibilités soulevées par ce travail préliminaire, puis portera nos regards sur les perspectives qu'il reste encore à explorer.

1.1 Projets ANR Orfeo et Rhapsodie

Le travail présenté dans ce mémoire s'inscrit dans le cadre de l'ANR ORFEO (Outils et Recherches sur le Français Ecrit et Oral), un projet qui s'étale sur trois années à partir de 2013 sous la direction de Jeanne-Marie Debaisieux. L'objectif principal de ce projet est de constituer un Corpus d'Etude pour le Français Contemporain en collectant des données, mais aussi en rassemblant un certain nombre de corpus existants issus à la fois de travaux sur l'écrit et sur l'oral, et couvrant au maximum la variété de genres et d'usages du français contemporain. Différentes couches d'annotation (morphologique, syntaxique(s), discursive, ou encore prosodique pour les données de l'oral) seront, à terme, disponibles sur l'ensemble du corpus qui constituera une véritable base de données authentiques permettant de développer des études et des outils en linguistique de corpus.

Un projet d'une telle envergure appelle, dès la phase de constitution du corpus, la conception d'outils d'annotation automatique qui faciliteront l'analyse de corpus nouvellement recueillis par des experts. L'objet de ce mémoire, la segmentation automatique d'un corpus oral en unités syntaxiques majeures, contribue précisément à cette tâche d'outillage des données puisqu'il présente une première expérience d'annotation par apprentissage automatique sur le treebank Rhapsodie, issu de l'ANR du même nom dirigé par Anne Lacheret entre 2008 et 2012 [Lacheret-Dujour *et al.* 2013], et participe ainsi au passage à l'échelle. Le treebank est présenté plus en détail dans la section 1.3.

1.2 Objectif(s)

Ce mémoire de recherche poursuit donc un objectif double qui va guider les choix faits tout au long de la chaîne de traitement mise en place.

1.2.1 Analyse de la corrélation syntaxe/prosodie

Le premier objectif est linguistique. La segmentation en unités syntaxiques majeures à partir d'indices prosodiques repose en effet sur l'hypothèse que les deux niveaux d'analyse sont corrélés (cf. les travaux pilotes sur le corpus Rhapsodie de Lacheret-Dujour *et al.* [2011] et Belião *et al.* [2013]) et que l'on peut déduire le premier du second. L'analyse que nous pourrons tirer des nos expériences permettra de vérifier cette hypothèse.

Cette directive déterminera par ailleurs certains paramètres de nos expériences, notamment la nécessité de travailler avec des modèles de classification lisibles afin de savoir, tout d'abord, si les indices prosodiques peuvent servir à segmenter efficacement en unités

syntaxiques, et si c'est effectivement le cas, de comprendre lesquels parmi ces indices se trouvent être objectivement les plus utiles.
Il sera également intéressant, avec la mise en place de deux séries d'expériences parallèles, dont l'une disposera d'annotations manuelles (ou en tout cas corrigées) du corpus Rhapsodie, de pouvoir mesurer l'utilité et l'apport de ces informations en comparaison avec un corpus moins doté en annotations.

1.2.2 Création d'un outil de segmentation automatique

Si le fondement est bien linguistique, le cœur de ce projet se situe davantage dans une optique de Traitement Automatique des Langues. La segmentation (ou annotation) d'un corpus est une tâche qui, effectuée manuellement par un expert, implique un coût très important à la fois en temps et en argent, mais pose aussi de nombreux problèmes d'homogénéité des données, d'une part à cause de la question de la régularité des analyses et d'autre part des accords entre experts dans le cas où la tâche est répartie entre plusieurs personnes. Ce coût n'est pas négligeable quand on cherche, comme dans le projet ANR Orfeo, à rendre disponible des corpus de plusieurs millions de mots.

Pour répondre à la nécessité de l'automatisation de la segmentation, nous tâcherons :
- d'explorer les possibilités offertes par les méthodes d'apprentissage automatique ;
- d'évaluer l'utilité de logiciels d'analyse automatique, de soulever les problèmes posés par leur utilisation (notamment en terme de formats d'entrée et de sortie) et de proposer des solutions ;
- de mettre en place une chaîne de traitement qui nécessite le minimum d'intervention manuelle possible.

On attend donc à l'issue de ce travail un premier outil qui puisse segmenter un corpus de français oral dont les annotations sont moins riches que celles du corpus de référence, avec des résultats corrects pour ne pas solliciter une correction manuelle trop importante.

1.3 Corpus de travail : le treebank Rhapsodie

Au cours de ces vingt dernières années, la linguistique de corpus s'est développée grâce à la constitution de corpus annotés dits de référence, avec en position pionnière le Penn Treebank [2] [Marcus *et al.* 1993], un corpus de langue anglaise de plus de 4 millions de mots entièrement annoté en part-of-speech (POS) et également doté d'un parenthésage en unités syntaxiques sur plus de la moitié du corpus. Un treebank se présente, comme son nom l'indique, sous la forme d'une structure arborescente, représentation qui a l'avantage de permettre d'ajouter à la dimension linéaire une dimension hiérarchique.
Un tel corpus de grande échelle peut servir à étudier certains phénomènes linguistiques en se basant donc sur l'usage, mais aussi confronter les théories élaborées à l'analyse de véritables échantillons de langue. Du côté du traitement automatique des langues, les treebanks sont utilisés pour entraîner ou pour tester des parseurs, analyseurs syntaxiques basés sur des modèles probabilistes, possibilité que nous allons d'ailleurs explorer dans la partie 3.

Bien que de taille modeste (environ 33000 mots pour 3h d'enregistrements), le treebank Rhapsodie est doté d'une annotation intonosyntaxique offrant la possibilité d'étudier en parallèle la prosodie et la syntaxe. C'est cette spécificité qui permet une première approche de la segmentation automatique en unités macrosyntaxiques sur le français parlé.

2. http://www.cis.upenn.edu/~treebank/

1.3.1 Composition du corpus

Le corpus sur lequel nous travaillons dans ce projet est un treebank qui a la particularité de ne comporter que des données issues de l'oral et de présenter plusieurs couches d'annotation : morpho-syntaxique, syntaxique mais aussi prosodique. Contrairement au Penn Treebank qui n'a étudié des spécificités de l'oral que les phénomènes de disfluences, Rhapsodie propose une étude approfondie de la prosodie grâce à la mise en parallèle d'informations acoustiques et perceptifs.

On recense parmi les échantillons qui composent le corpus 32 échantillons de source externe, c'est-à-dire constitués par des projets antérieurs, et 25 échantillons de source interne, constitués donc spécialement pour le projet Rhapsodie [3]. La sélection de ces échantillons assure la variété des enregistrements, qui peuvent être des monologues ou des dialogues, des discours (sermonts, discours politiques...) et des situations de parole spontanée.

1.3.2 Un corpus richement annoté

Un des objectifs du projet ANR Rhapsodie est le développement de schémas d'annotation en syntaxe et en prosodie.

Les annotations syntaxiques portent à la fois sur le niveau microsyntaxique (parties du discours et analyse en dépendance) et sur le niveau macrosyntaxique qui nous intéresse pour ce travail. Ce dernier identifie deux types d'unités principales : les unités rectionnelles, construites autour d'une tête et qui n'est syntaxiquement dépendante d'aucun élément de rang supérieur, et les unités illocutoires réalisant un acte illocutoire pouvant comporter plusieurs unités rectionnelles dont une seule, appelée noyau, porte la force illocutoire [Benzitoun *et al.* 2010]. Ces informations sont disponibles sous forme d'annotation textuelle (d'où sont tirés la plupart de nos exemples) et dans un fichier XML.

En ce qui concerne la prosodie, l'annotation a été effectuée dans des TextGrids, un des types d'objets de l'outil Praat [4] [Boersma 2002], particulièrement utilisé pour la transcription, la segmentation et l'annotation de données orales en raison de sa composition en tires de données. Chacune des tires représente une information sous la forme d'annotations divisibles en intervalles alignés sur l'axe temporel.

Les annotations, notamment en proéminences, disfluences, contours mélodiques, et périodes intonatives, s'étendent donc sur plusieurs tires comme on peut le voir en 1.1 et peuvent être alignées sur différentes unités : la période, le paquet intonatif, le groupe rythmique, le pied métrique, le mot, la syllabe, le phonème et la pause. Seules les unités qui ont servi à nos expériences, la période et la syllabe, seront détaillées dans le chapitre suivant.

3. Le détail des sources peut être retrouvé sur le site du projet : http://projet-rhapsodie.fr/propriete-intellectuelle.html

4. http://www.fon.hum.uva.nl/praat/

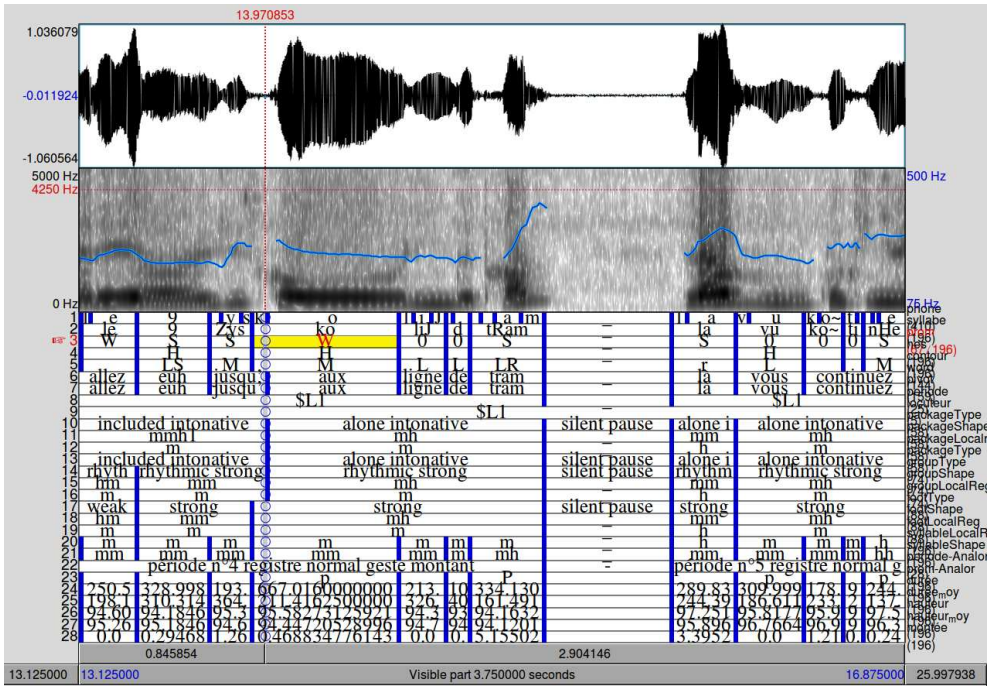


FIGURE 1.1: Exemple de TextGrid dont les annotations dans les tires sont alignées sur le signal (fréquence fondamentale en haut et spectrogramme au milieu) sous Praat [Rhap-D0007, Corpus Avanzi]

Chapitre 2

La segmentation comme tâche de classification supervisée

2.1 Redéfinition de la tâche de segmentation

Segmenter un texte, c'est le découper en plus petites unités. On a besoin pour cela de définir ces unités, mais aussi de les délimiter en en établissant les frontières. Segmenter un texte en mots revient donc à considérer les espaces et certains signes de ponctuation (l'apostrophe, et le tiret dans le cas de l'inversion sujet-verbe) comme des frontières dont on se sert pour indiquer les endroits où découper les unités, les séparer.

Pour ce projet, on cherche à segmenter un corpus en unités macrosyntaxiques, unités qui peuvent autant être constituées de plusieurs mots que d'un seul. Chaque espace intermot est donc susceptible d'être une frontière, mais tous les espaces ne sont pas des frontières. Si on fait l'hypothèse que les frontières des unités macrosyntaxiques qui nous intéressent sont prosodiquement marquées, il est possible de s'appuyer sur des indices prosodiques afin de déterminer, pour chacun des espaces, si oui ou non il s'agit d'une frontière, d'une part, et d'autre part, de quel type de frontière il s'agit. Cette tâche peut être résolue en utilisant un classifieur.
En effet, la classification est une tâche issue de la fouille de données qui consiste à répartir un ensemble d'éléments en plusieurs classes à l'aide d'informations matérialisées sous la forme d'une matrice d'attributs rattachés à chaque élément. Ces classes peuvent être inconnues (cas de la classification non-supervisée, ou clustering) comme dans l'expérience consistant à segmenter les tours de parole sans connaître le nombre de locuteurs à l'avance menée par Chen et Gopalakrishnan [1998] ou bien prédéfinies (cas de la classification supervisée) comme dans ce projet.

Les indices que nous voulons utiliser sont rattachés soit à des mots soit à des syllabes et non aux espaces intermots que l'on voudrait classifier. Dans nos expériences, on considère que l'objet à classifier est la frontière droite du mot courant, et que toutes les informations sur la position courante (0) sont celles du mot qui précède l'espace.

2.2 Classes : les frontières des unités macrosyntaxiques

Les unités qui nous intéressent pour ce mémoire sont donc les unités macrosyntaxiques qu'il est essentiel de délimiter pour pouvoir aller plus loin dans l'annotation syntaxique (notamment micro) : ce sont donc les classes dans lesquelles on cherchera à répartir tous les espaces intermots.

On distingue deux types d'unités à retrouver : les unités syntaxiques majeures (ou unités illocutoires), et les unités syntaxiques mineures (ou composantes illocutoires),

définies ci-après. Une troisième classe, inhérente à notre problématique, regroupera tous les espaces qui ne sont pas des frontières syntaxiques (de fait, la majorité des espaces).

Ces deux types d'unités sont encodées dans un premier temps sous le format BILU [1].

2.2.1 Les Unités Illocutoires (UI)

Comme nous l'avions défini précédemment, une unité illocutoire est une unité réalisant un acte illocutoire, qui peut être une assertion, une interrogation, une approbation, une promesse etc. Dans la sortie textuelle de Rhapsodie, seule la frontière droite d'une UI est annotée par la balise `//`.

On distingue différents types d'UI :
- l'UI parenthétique, dont le symbole est combiné avec celui des parenthèses : `( //)` ;
- l'UI enchâssée, dont le symbole est combiné avec celui des enchâssements, les crochets : `[ //]` ;
- l'UI en position de complément différé, dont le symbole est combiné avec celui des epéxégèses, le plus : `//+` ;
- et l'UI parallèle avec une autre UI contiguë : `//=`.

Ces deux derniers types ne sont toutefois pas étudié ici en raison du caractère lexical du parallélisme en plus de son caractère syntaxique, et du fait que l'epéxégèse d'une UI n'est pas annotée explicitement dans le format mis à disposition (voir 2.4).

Les UI enchâssées, quant à elles, correspondent à des discours rapportés, ou de façon générale, à des greffes, c'est-à-dire des séquences qui contribuent à remplir la position syntaxique d'un élément recteur [2].

[Rhap-D0005],PFC [a]
(1) UI simple (interrogation, affirmation, interrogation)
"enfin" c' est pas trop stressant comme métier "euh" // $L1 "ben" "euh" "euh" "euh" { s~ | si | si } [b] > quand même // $- $L2 "oui" //

(2) Insertion (cas d'une interruption)
ce qui fait que **($L2 "ah" ouais //)** $L1 { c' est | c' est } pas { facile | facile } //

(3) Enchâssement (cas d'une greffe)
{ j' ai | j' ai } (d' abord) "euh" travaillé dans { des "euh" **[comment & //**] | des entreprises { de | d' } automatisme } //

a. Corpus Phonologie du Français Contemporain, de Laks *et al.* [2009]
b. Les balises { | } marquent le phénomène d'entassement, que nous n'étudions pas dans ce mémoire.

Sont donc considérées comme des frontières d'UI les balises `//`, `(`, `//)`,`[` et `//]`. Ajoutons que les UI simples n'ont pas de frontières gauches explicites, mais étant donné que tous les énoncés du corpus sont, en situation, des actes illocutoires, une frontière `//` est à la fois la fin de l'unité qui précède et le début de la suivante.

Les UI sont les unités maximales de la macrosyntaxe, ce qui fait des frontières d'UI des frontières syntaxiques majeures.

1. BILU pour **B**egin, **I**n, **L**ast, **U**nique, **O**ut où B sert à annoter le début d'une unité, I sa continuité, L sa fin tandis que U cumule les fonctions de B et L pour une unité formée d'un seul bloc, et O désigne une unité externe.
2. Toutes ces informations sont extraites du protocole de codage macrosyntaxique.

2.2.2 Les Composantes Illocutoires (CI)

Dans une unité illocutoire, on distingue différentes composantes appelées composantes illocutoires dont certaines sont essentielles et d'autres facultatives.

Le noyau La composante qui porte la force illocutoire est le noyau. D'après le protocole de codage syntaxique de Rhapsodie, cinq tests permettent de reconnaître un noyau :

1. la restitution de sa valeur performative implicite sans en changer le sens ;
2. la possibilité de faire entrer le noyau sous la portée d'un adverbe d'énonciation (*franchement*, *à vrai dire*) sans en changer le sens ;
3. la commutation de sa modalité avec d'autres modalités ;
4. l'isolation d'un noyau dans un tour de parole sans changer de sens ;
5. et enfin la possibilité de cibler un noyau avec une interrogation ou une négation.

Le pré-noyau, le post-noyau et l'in-noyau Les autres composantes illocutoires sont déterminées suivant leur position par rapport au noyau. S'il se situe à gauche, il s'agit d'un pré-noyau annoté `<`, à droite d'un post-noyau annoté `>` et s'il s'insère à l'intérieur d'un noyau, il s'agit d'un in-noyau annoté par des parenthèses `()` [3].

Chacun des symboles représentant ces trois unités a une contrepartie que l'on qualifiera dans ce mémoire de "différée" (cas de l'epéxégèse, et donc associée alors au symbole `+` pour donner respectivement `<+`, `>+` et `(+ )`) qui précise que l'unité rectionnelle en question se poursuit malgré la présence d'une frontière. Tous ces symboles représentent autant de types différents de frontières de CI dans nos données du fait qu'elles puissent être marquées prosodiquement aussi, mais de manière différente.

[Rhap-M2004]
ˆmais ce soir <+ ce qui importe < c'est { l'avenir | >+ notre avenir | celui de nos enfants } //

Alors qu'une UI ne peut être composée que d'un seul noyau, les autres composantes sont des unités récursives comme on peut le voir dans l'exemple précédent.

2.3 Attributs : informations linguistiques

Dans la classification supervisée, un attribut est un élément qui apporte une information sur l'objet à classifier et qui sera donc potentiellement utilisé dans ce but. Dans cette section, nous allons présenter tous les attributs utilisés pour nos expérimentations, même si ceux-ci feront par la suite l'objet d'une sélection, classés en trois domaines : la morpho-syntaxe, et la prosodie et d'autres informations structurelles.

2.3.1 Informations morpho-syntaxiques

La première information que l'on pourrait utiliser en attribut est le mot orthographique lui-même, récupéré directement à partir de la transcription orthographique. Même si l'objet de la classification reste l'espace intermot, cet espace peut être caractérisé par son contexte (le(s) mot(s) précédent(s) ou suivants(s)). Nous n'avons néanmoins pas pu utiliser cet attribut dans nos expériences en raison du format spécifique d'entrée de Weka

3. Ce qui différencie l'insertion d'une UI avec l'insertion d'un noyau (donc, l'in-noyau) est la présence ou l'absence du symbole des UI `//`. Par ailleurs, alors que l'insertion d'une UI peut être le résultat de l'interruption de l'interlocuteur ou bien une véritable insertion parenthétique dans un monologue, l'in-noyau est forcément réalisé par le même locuteur.

et des algorithmes de classification qui imposent des contraintes sur le format des attributs : chacun des mots de notre corpus doit être déclaré comme un attribut, et pour chacun des espaces à classifier on doit alors indiquer si un mot fait partie de son contexte (3.1.1). Le fichier d'entrée ainsi constitué devient alors trop lourd pour être utilisé par Weka.

En revanche, le nombre de parties du discours et d'étiquettes de chunks est suffisamment petit pour que ces deux attributs soient représentés par une liste chacun, allégeant le poids du fichier d'entrée.

2.3.1.1 Partie du discours (POS)

Que ce soit pour les UI ou les CI, la catégorie syntaxique du mot initial n'est pas aléatoire. De même, toutes les catégories ne peuvent pas se trouver en fin d'UI ou en fin de CI : typiquement une préposition ou un déterminant, qui nécessitent respectivement un complément et une tête nominale, ont très peu de chance de se retrouver à la fin d'une unité macrosyntaxique. Les seuls cas où ces propriétés ne seraient pas vérifiées peuvent être dus à un énoncé syntaxiquement inachevé.
En attachant à chaque mot la partie du discours correspondante, on devrait augmenter les chances de ne pas détecter de frontières aberrantes.

Les étiquettes POS utilisées pour ce projet sont calculées automatiquement par SEM [4] (Segmenteur-Etiqueteur Markovien), un outil basé sur les CRF [5] [Tellier *et al.* 2012]. Le jeu d'étiquettes morpho-syntaxiques est, à peu de choses près, celui proposé par Crabbé *et al.* [2008] et n'est, par conséquent, pas adapté aux données issues de l'oral puisqu'il a été conçu pour l'écrit. Malgré quelques incohérences, notamment au niveau des marqueurs de discours ("euh", "mh" etc.) totalement absents du corpus d'apprentissage, SEM est capable d'obtenir un score de 81,6% sur des données orales [6].

dans	P	pour	P
le	DET	lesquelles	PROREL
Gâtinet	NC	euh	V
mh	ADJ	l'attachement	CLS
et	CC	est	V
puis	CC	peut-être	ADV
c'est	V	moins	_ADV [7]
un	DET	en	_ADV
pays	NC	profondeur	_ADV

FIGURE 2.1: Exemples d'erreurs d'annotation POS sur les disfluences par SEM [Rhap-D1001, ESLO]

Dans la figure 2.1, les marqueurs de discours "mh" et "euh" ont été respectivement annotées `ADJ` et `V`, les étiquettes les plus probables et cohérentes selon le modèle qui

4. www.lattice.cnrs.fr/sites/itellier/SEM.htm
5. Conditional Random Fields (champs markoviens conditionnels), modèle probabiliste permettant de calculer la probabilité d'obtenir une certaine étiquettes sachant certaines propriétés issues de l'observable (les features) que l'on aura définies au préalable.
6. Cette étude menée par Tellier *et al.* [2013] porte sur un échantillon du corpus ESLO1 (les Enquêtes Socio-Linguistiques à Orléans) de 8305 mots. Certains extraits d'ESLO font d'ailleurs partie des sources externes de Rhapsodie. Plus d'informations sur la constitution de ce corpus dans Eshkol-Taravella *et al.* [2012] et sur le site : http://www.univ-orleans.fr/eslo/

tient à la fois compte des mots qui précèdent et qui suivent, mais aussi des étiquettes correspondantes. SEM annote en fait des séquences d'étiquettes.

D'ailleurs, à cause de cette caractéristique, SEM risque de faire des erreurs supplémentaires : il a été entraîné sur des phrases, mais est appliqué ici sur des séquences qui font la taille d'un échantillon entier, puisque nous cherchons précisément à segmenter en unités syntaxiques maximales.

Les annotations POS de notre corpus ne sont donc pas parfaites, mais une correction manuelle nécessiterait un coût en temps beaucoup trop élevé pour une marge d'erreur qui n'est pas si importante. Nous nous sommes donc contentés de remplacer les erreurs systématiques sur les marqueurs discursifs (notamment les "ah" ou "mh") que l'on annote `I` pour interjection, catégorie la plus proche sur le plan morpho-syntaxique et correspondant au choix de Rhapsodie, à l'aide d'expressions régulières.

2.3.1.2 Chunk

Le chunk est la plus petite séquence d'unités linguistiques possible formant un groupe construit autour d'une tête, et qui n'est ni discontinue, ni récursive [Abney 1991]. L'analyse syntaxique qui en résulte reste superficielle comparée à une analyse en véritables constituants syntaxiques, mais elle est suffisante pour un premier repérage des potentielles frontières macrosyntaxiques.

le	DET	B-NP
l'	_DET	I-NP
immobilier	NC	I-NP
ici	ADV	B-AdP
dans	P	B-PP
ce	DET	I-PP
coin	NC	I-PP
est	V	B-VN
très	ADV	B-AP
très	ADV	I-AP
cher	ADJ	I-AP

FIGURE 2.2: Exemple d'annotation (corrigée) en chunks [Rhap-D0004, CFPP2000]

À partir des formes graphiques accompagnées d'une couche d'étiquettes POS, SEM produit une segmentation en chunks encodée en BIO [8] suivi du chunk. L'exemple en 2.2 met en évidence le découpage en chunks proposé par SEM, qui peut potentiellement aider le classifieur à ne pas segmenter au milieu d'un groupe nominal ou prépositionnel (respectivement dans l'exemple, NP et PP). Toutefois, les deux types de frontières sont loin de coïncider : les frontières macrosyntaxiques sont moins fréquentes que les frontières de chunks, dont les unités sont souvent très petites (voir le noyau verbal NV dans l'exemple).

Notons enfin que la segmentation en chunks et non en constituants syntaxiques impose certaines limites et nécessite de faire certains choix linguistiques discutables [9] mais nous ne revenons pas sur ces choix étant donné qu'il s'agit dans ce projet de ne l'utiliser

8. BIO pour **B**egin (mot en début de chunk), **I**n (mot à l'intérieur d'un chunk), **O**ut (mot qui ne fait pas partie d'un des chunks définis).

9. Ces choix sont exposés dans le guide d'annotation à l'adresse suivante : http://www.lattice.cnrs.fr/sites/itellier/guide.html

qu'en tant qu'indice pour un classifieur probabiliste et de n'apporter qu'un minimum de correction manuelle aux données.

2.3.1.3 Marqueurs discursifs

Par marqueurs discursifs, on entend les "petits mots qui balisent l'oral" servant tantôt à lier et tantôt à ponctuer et que Morel et Boileau [1998] répartissent en quatre groupes suivant leur fonction majeure :

1. la "régulation de la coénonciation" ;
2. la "modulation de la qualification du référent" ;
3. la "restriction du champ référentiel" ;
4. et "la scansion du discours".

Dans Rhapsodie, parmi ces possibilités, seules les fonctions (1), (2) et (4) ont été considérées comme définissant des marqueurs du discours. On a également ajouté dans cette catégorie les régulateurs de parole comme "mh" ou encore la particule de disfluence "euh", ainsi que des unités complexes comme "tu vois" ou "tu crois pas" possédant des propriétés analogues à ceux des marqueurs discursifs simples. Tous ces éléments sont repérés par des guillemets " " dans le corpus, comme dans les exemples suivants.

Cas (1) :

```
[Rhap-M0023]
"ouais" il y a un accident "quoi" //
```

Le premier régulateur permet de lier la phrase à l'énoncé qui précède tandis que le second est un régulateur qui ponctue le présent énoncé.

Cas (2) :

```
[Rhap-D0009]
il n'y a plus "disons" ce qu'on appelait autrefois { d~ | de } grands écrivains //
```

Les marqueurs discursifs ne se trouvent pas forcément en début ou en fin d'énoncé.

Cas (4) :

```
[Rhap-D0009]
François Mauriac était-il le dernier "alors" //=
```

2.3.1.4 Introducteurs d'UI

Certains mots, notamment les conjonctions et les adverbes comme "et" ou "mais", sont considérés commes des introducteurs d'UI s'ils permettent de préciser la nature de la relation entre l'UI qu'ils introduisent et l'UI qui précède, et s'ils n'ont pas de relation syntaxique avec d'autres éléments de l'UI [10].

```
[Rhap-D0009]
^parce ^que prélude < ça fait penser { à ( évidemment ) | à } Frédéric Chopin
//
```

10. http://rhapsodie.ilpga.fr/wiki/Unit%C3%A9s_illocutoires

Dans le corpus, chaque mot constituant un introducteur est précédé de la balise ^.

Cette information fait partie du codage macrosyntaxique et devrait, compte tenu de sa fonction, être un attribut essentiel dans la classification des frontières. C'est ce que nous allons essayer de vérifier à travers une première série d'expériences. Dans la seconde, elle ne sera évidemment pas utilisée en tant qu'attribut puisqu'elle résulte d'une annotation manuelle.

2.3.2 Indices prosodiques

La prosodie est un domaine d'étude entremêlant plusieurs niveaux d'analyse : selon Lacheret-Dujour et Beaugendre [1999], elle est "actualisée dans la substance principalement par des modulations de fréquence fondamentale, de durée et d'intensité, qui se combinent aux caractéristiques intrinsèques des unités phonétiques dans le signal de parole et qui ont pour fonction de réaliser le système dans le discours". Ce lien entre données physiques concrètes et fonctionalité discursive est traduit par la distinction, parmi nos données prosodiques, de deux types d'indices : d'un côté les indices acoustiques (hauteur, durée, montée, slowing) et de l'autre les indices perceptifs (proéminence, période, hésitation, contour syllabique, registre local).

Les premiers ont été directement récupérés à partir de sorties de l'outil de segmentation semi-automatique en constructions prosodiques Analor [11] [Avanzi *et al.* 2008], tandis que les seconds ont été annotés soit automatiquement (avec Analor), soit manuellement dans des tires du format TextGrid. La récupération des données est détaillée dans la section 2.4.

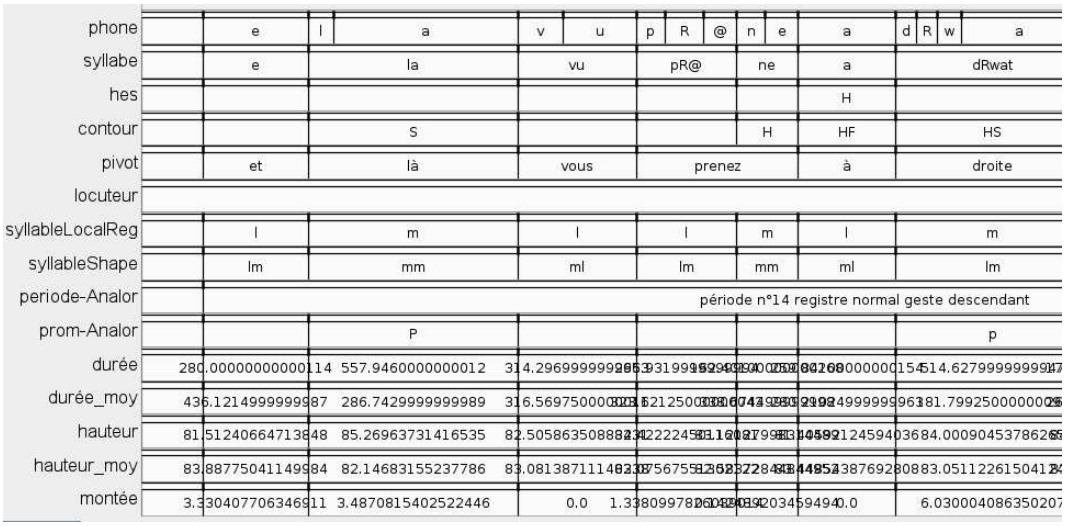


FIGURE 2.3: Exemple de TextGrid avec les tires correspondant à peu près à nos attributs, ouvert sous Analor [Rhap-D0017]

2.3.2.1 Indices acoustiques

Tous les indices acoustiques sont calculés et alignés sur la syllabe, ce qui implique que l'utilisation de ces indices pour la segmentation automatique nécessite un alignement préalable du texte au son et une identification des frontières de syllabes.

11. http://www.lattice.cnrs.fr/analor

Hauteur, durée et montée Sur Analor, la détection des proéminences est réalisé grâce à la combinaison de différents paramètres auxquels sont attribuées des notes qui évaluent la force d'une syllabe par rapport aux autres. À partir du fichier résultats, il est possible de récupérer les paramètres (notamment acoustiques) ayant servi à cette tâche. Parmi eux, nous avons retenu la hauteur, la hauteur moyenne et la montée (exprimées en demi-tons) ainsi que la durée et la durée pondérée (en ms).

La durée pondérée et la hauteur moyenne sont calculées sur un empan, c'est-à-dire une fenêtre de quelques syllabes précédant et suivant la syllabe courante. Dans notre cas, l'empan s'étend de deux syllabes avant à deux syllabes après la syllabe courante. La montée caractérise, quant à elle, l'amplitude du contour mélodique montant.

Slowing D'autre part, une autre mesure de durée a été utilisée : le coefficient de ralentissement (ou *slowing*), calculé par Cédric Gendrot en comparant les durées des syllabes du corpus Rhapsodie aux durées de références de diphones tirés du NCCF [12]. Contrairement à la valeur brute de durée des syllabes, cette mesure normalisée permet de rendre compte de la variation de durée tout en prenant en considération la variation co-intrinsèque d'un phonème (notamment l'allongement d'un phonème pour compenser la durée d'un phonème adjacent) [Gendrot *et al.* 2012].
Un fort coefficient indique donc un allongement significatif de la syllabe par rapport à la moyenne, un indice qui pourrait permettre de repérer les hésitations, les emphases, mais surtout les fins d'unités prosodiques et d'unités syntaxiques souvent caractérisées par un accent entraînant un allongement.

2.3.2.2 Indices perceptifs

Les indices présentés ci-dessous ont été encodés manuellement sur des bases perceptives uniquement, sauf pour les périodes et les proéminences, puisqu'on a utilisé celles qui ont été détectées automatiquement par le logiciel Analor. L'idée est que l'auditeur sélectionne, parmi les variations acoustiques qu'il perçoit, celles qui sont pertinentes pour la communication langagière.

Périodes La période intonative est définie par Lacheret *et al.* [2002] comme une unité d'intégration prosodique maximale. Dans le logiciel Analor, la détection des frontières de la période se fait sur le principe de l'activation de seuils pré-établis sur les quatre critères suivants [Avanzi *et al.* 2008] :

1. la durée de la pause ;
2. l'amplitude du geste (différence de hauteur entre le dernier extremum de F0 et la moyenne de F0 sur toute la portion qui précède la pause) ;
3. l'amplitude du saut (différence de hauteur entre la dernière valeur de la fréquence fondamentale F0 précédent la pause et la première valeur de F0 suivant la pause) ;
4. l'absence d'hésitation à proximité immédiate de la pause.

Une fois les frontières établies, la période est caractérisée par son registre (`bas`, `normal`, `haut`) et son geste (`descendant`, `plat`, `montant`). Dans nos données, ces deux caractéristiques sont concaténées pour donner, par exemple, la valeur `normalmontant`.

Proéminences (faibles, fortes) Une syllabe proéminente est une syllabe saillante qui se détache perceptivement des autres. On distingue la proéminence forte annotée `P` de la proéminence faible annotée `p`. De la même manière que pour les périodes, les proéminences potentielles sont repérées par des critères [Avanzi *et al.* 2011] :

12. Le Nijmegen Corpus of Casual French [Torreira *et al.* 2010], un corpus de plus de 30h de parole spontanée

1. la durée pondérée de la syllabe courante ;
2. sa hauteur moyenne ;
3. l'amplitude de sa montée (si le contour est montant) ;
4. la présence d'une pause silencieuse (sans "euh" d'hésitation) après la syllabe courante.

La moyenne pondérée des scores de ces quatre critères déterminera la force de la proéminence.

Hésitations Les hésitations peuvent être soit la particule caractéristique pour le français parlé "euh", soit des syllabes hésitantes. Dans les deux cas, le symbole `H` est utilisé dans la tire correspondante s'il y a une hésitation, sinon l'intervalle reste vide.

Contours globaux et registres locaux sur la syllabe Les contours globaux [Obin 2012] sont attribués à plusieurs unités dans le corpus Rhapsodie, mais nous ne retenons que celui de la syllabe, par souci d'homogénéité des attributs. Ils permettent de décrire le niveau de hauteur de la F0 par rapport à la moyenne du locuteur selon un encodage sur cinq niveaux : `L` pour un niveau très bas, `l` pour un niveau bas, `m` pour un niveau moyen, `h` pour un niveau haut, et `H` pour un niveau très haut. Cette description est donnée pour le point initial de l'unité, et son point final.
À cette information peut s'ajouter le niveau et la position du point le plus saillant (extremum) dans l'unité donnée, qui peut être dans le premier tiers `1`, dans le deuxième `2` ou dans le dernier `3`.

La valeur `lhH3` sur une syllabe signifie donc que le point initial du contour de F0 de la syllabe est bas, que son point final est haut, et que le point le plus saillant est très haut et se trouve dans le dernier tiers de la syllabe.

Sur la même échelle de niveaux, on trouve le registre local du locuteur.

2.3.3 Changement de locuteur

Outre les indices morpho-syntaxiques et prosodiques, nous avons inclus dans nos expériences un attribut sur le changement de locuteur pouvant s'avérer utile pour la segmentation des dialogues [Stolcke et Shriberg 1996, Shriberg *et al.* 2000, Liu *et al.* 2006]. En effet, la plupart du temps, un changement de locuteur implique une frontière macrosyntaxique puisque nous travaillons sur des données alignées sur la seule dimension temporelle, où tous les locuteurs se succèdent sur un même signal et une même transcription.

Cette remarque ne vaut pas pour les chevauchements de parole, puisque dans ces cas précis les tours de parole ne sont pas en réalité interrompus mais superposés. Encodées sur une seule position linéaire, les données ne permettent pas l'annotation des énoncés de plusieurs locuteurs en même temps, ce qui entraîne (1) pour l'annotation syntaxique, la délinéarisation des données initiales et le montage artificiel de tours de parole successifs dans la transcription, et (2) pour l'annotation prosodique, la sélection forcée d'un locuteur dit dominant et la perte d'informations concernant le(s) autre(s) locuteur(s).

[Rhap-D2012]
(1)
$L2 [...] il y a Fitelberg qui ensuite "euh" va abandonner la composition { pour **$- "euh" la direction de l'orchestre |} //+ $L1 {| pour la direction de l'orchestre } // -$**
(2)
$L2 ça pourrait être **$- { Lohengrin | ($L1 "oui" //) -$** ^et Parsifal } > effectivement //

Dans le premier extrait, les paroles qui se chevauchent ont été mises l'une à la suite de l'autre, donnant l'impression d'un simple changement de tour de parole. Dans ce cas, les frontières syntaxiques sont bien celles de chacun des locuteurs. En revanche, dans le deuxième extrait, le locuteur $L1 est artificiellement inséré dans les paroles du locuteur $L2 alors que les énoncés se superposent. Ainsi, si le POS du mot qui suit *Lohrengrin* n'est plus `CC` pour *et* mais `I` pour *oui*.

De plus, les parenthèses correspondent donc bien à des frontières macrosyntaxiques pour le locuteur $L1 mais l'énoncé de locuteur $L2, qui formait une unité, a été artificiellement découpé en trois segments.

Du côté de l'annotation prosodique, on aurait dans les deux cas uniquement les informations du locuteur dominant. Or, nos données sont alignées sur la transcription orthographique : si le locuteur dominant est $L2, les indices prosodiques rattachés aux segments de $L1 seront alors ceux de $L2.

Dans nos données récupérées de la tire locuteur, comme dans l'annotation textuelle, les locuteurs sont représentés par le symbole $L[numero_de_locuteur]. En cas de chevauchement, on a simplement le symbole des deux locuteurs séparés par un tiret : $L1-$L3 signifie que le mot courant se trouve dans un chevauchement entre le locuteur $L1 et le locuteur $L3. Cette information fait par ailleurs partie des critères pour détecter une période dans Analor.

2.4 Sélection et format des données

Le corpus Rhapsodie dispose d'annotations riches à la fois en syntaxe et en prosodie, fruit du travail cumulatif de plusieurs annotateurs et de révisions expertes manuelles. Pour se conformer aux objectifs définis par notre problématique, il ne s'agit toutefois pas seulement d'utiliser le maximum d'informations linguistiques parmi ceux qui sont susceptibles d'aider à la segmentation en unités macrosyntaxiques pour classifier, mais aussi dans une seconde série d'expériences, d'utiliser uniquement celles qui sont indispensables (pour lancer les logiciels d'analyse automatique comme Analor 2.3.2 notamment) et qui ne demandent qu'un minimum d'intervention humaine.
De cette manière, il sera possible de savoir, d'une part, quel(s) type(s) d'information permet(tent) de classifier efficacement les espaces intermots, et d'autre part, dans quelle mesure ces expériences sont réalisables sur des corpus moins dotés.

Les expériences que l'on appelera par la suite *sur données complètes* comprennent tous les attributs présentés dans le tableau 2.1, tandis que les expériences *sur données partielles* ne prendront en compte que les attributs dits automatiques. Il est important de noter que même si ces attributs ont effectivement été récupérés de manière automatique, leur calcul par des outils de traitement automatique suppose un certain nombre de prérequis : une transcription orthographique segmentée en mots pour l'application de SEM, un alignement du texte au son et une identification des frontières de syllabes pour tous les indices prosodiques, et enfin l'annotation des pauses et des "euh" pour la détection de périodes par Analor [13].

L'ensemble de ces informations est en fait issu des TextGrids (format de prédilection adopté pour annoter et corriger) puis récupéré par un parseur développé par Julie Belião (voir Belião [2012]) en un format tabulaire, tel qu'on le voit dans la figure 2.4, avec un fichier par échantillon du corpus. En ligne, on trouve chacun des mots du corpus suivant la segmentation utilisée pour l'annotation syntaxique manuelle, et, en colonne, les

13. Du moins jusqu'à la version 0.2.5 d'Analor utilisée pour ce mémoire.

	Unité	Codage
Classes	UI	BILUO
	Noyau	BILUO
	Pré-noyau (différé)	BILUO
	In-noyau (différé)	BILUO
	Post-noyau (différé)	BILUO
Attributs automatiques	POS	ADJ, _ADJ, ADJWH, ADV, _ADV...
	Chunk	B-AdP, B-AP, B-CONJ, B-NP, B-PP...
	Slowing	Nombre réel
	Montée	Nombre réel
	Hauteur (moyenne)	Nombre réel
	Durée (pondérée)	Nombre réel
	Proéminence	Nombre réel
	Période	basdescendant, basmontant, basplat...
	Locuteur	$L1, $L1-$L2, $L1-$L3, $L1-$L4...
Attributs manuels	Marqueur discursif	BILUO
	Introducteur	BILUO
	Hésitation	H, %
	Contour	hh, h1, hlH1, hlH2, hlH3, hlh2, hm,...
	Registre	m, h, H, l, L

TABLE 2.1: Récapitulatif des différentes unités utilisées pour les expériences de ce mémoire, et leur codage à ce stade

différentes données des TextGrids, incluant nos classes et nos attributs.

Toutes les données perceptives nous ont été directement fournies dans ce tableau et ce, sur un empan de [-3 :3], c'est-à-dire sur trois unités précédant l'unité courante et trois unités suivant le mot courant. Ceci signifie que pour un attribut tel que le registre local, on a l'information du registre de la dernière syllabe qui précède immédiatement l'espace intermot (position 0) ainsi que les valeurs de registre local des trois syllabes qui précèdent et des trois syllabes qui suivent.

Quant aux données acoustiques et morpho-syntaxiques, elles ont été ajoutées artificiellement dans les TextGrid à l'aide d'un script afin d'être parsées mais ne donnent l'information que sur le mot ou la syllabe courante. On a élargi à un empan similaire à celui des données perceptives à l'étape suivante : la vectorisation des données pour Weka 3.1.1.

	A	B	C	D	E	F	G	H	I	J
1	label	iu	kernel	prekernel	integratedprekernel	inkernel	integratedinkernel	postkernel	integratedpostkernel	parenthesis
2	est-ce	B	B	0	0	0	0	0	0	0
3	que	I	I	0	0	0	0	0	0	0
4	vous	I	I	0	0	0	0	0	0	0
5	pourriez	I	I	0	0	0	0	0	0	0
6	décrire	I	I	0	0	0	0	0	0	0
7	euh	I	I	0	0	0	0	0	0	0
8	les	I	I	0	0	0	0	0	0	0
9	déplacements	I	I	0	0	0	0	0	0	0
10	avec	I	I	0	0	0	0	0	0	0
11	précision	L	L	0	0	0	0	0	0	0
12	une	B	B	0	0	0	0	0	0	0
13	journée	I	I	0	0	0	0	0	0	0
14	eh	B	B	0	0	0	0	0	0	B
15	ben	I	I	0	0	0	0	0	0	I
16	XXX	I	I	0	0	0	0	0	0	I
17	soit	I	I	0	0	0	0	0	0	I
18	on	I	I	0	0	0	0	0	0	I
19	travaille	L	L	0	0	0	0	0	0	L
20	par	I	I	0	0	0	0	0	0	0
21	exemple	L	L	0	0	0	0	0	0	0
22	soit	B	0	0	0	0	0	0	0	0
23	on	I	B	0	0	0	0	0	0	0
24	travaille	L	L	0	0	0	0	0	0	0
25	donc	B	0	U	0	0	0	0	0	0
26	moi	I	0	U	0	0	0	0	0	0
27	ben	I	B	0	0	0	0	0	0	0
28	je	I	I	0	0	0	0	0	0	0
29	vais	I	I	0	0	0	0	0	0	0
30	je	I	I	0	0	0	0	0	0	0
31	je	I	I	0	0	0	0	0	0	0
32	prends	I	I	0	0	0	0	0	0	0
33	le	I	I	0	0	0	0	0	0	0
34	mét~	I	I	0	0	0	0	0	0	0
35	je	I	I	0	0	0	0	0	0	0
36	prends	I	I	0	0	0	0	0	0	0
37	le	I	I	0	0	0	0	0	0	0
38	métro	I	I	0	0	0	0	0	0	0
39	le	I	I	0	0	0	0	0	0	0
40	matin	I	I	0	0	0	0	0	0	0
41	bon	I	I	0	0	0	0	0	0	0
42	jusqu'	I	I	0	0	0	0	0	0	0
43	au	I	I	0	0	0	0	0	0	0
44	Palais	I	I	0	0	0	0	0	0	0
45	Royal	L	L	0	0	0	0	0	0	0

FIGURE 2.4: Extrait du tableau de données une fois parsées par Julie Belião

Chapitre 3

Classification des frontières par apprentissage automatique

La classification supervisée est une tâche qui consiste à apprendre un modèle (classifieur) à partir d'exemples dont on connaît la classe. Le classifieur retiendra alors un certain nombre de critères qu'il pourra réappliquer pour classifier de nouvelles données. Afin de tirer le maximum des apprentissages sur notre corpus, on a recours à la validation croisée. Le corpus donné en entrée est alors divisé en deux ensembles distincts : la première partie, représentant dans notre cas 9/10 du corpus entier, sert de corpus d'entraînement (*training set*) pour le classifieur qui va ensuite tester le modèle construit sur le 1/10 restant, soit la seconde partie (*test set*). Cette phase d'entraînement est répétée autant de fois qu'il y a de divisions, jusqu'à ce que chacune des divisions ait servi tantôt de *training set* et tantôt de *test set*. A l'issue de ces tests, une évaluation globale des modèles construits est donnée sous la forme d'une matrice comparant des résultats avec les classes déjà connues, appelée matrice de confusion, et dont on se servira dans la section 3.2 comme d'une premier indice de la qualité de la classification.
Si ce processus est entièrement assuré par Weka, d'autres prétraitements ont été nécessaires pour préparer nos données à la classification.

3.1 Préparation du corpus

3.1.1 Vectorisation des données

Pour rappel, nous disposons à ce stade de données tabulaires, où chaque colonne représente une information syntaxique ou acoustique sans distinction entre celles qui définissent les classes que l'on cherche à retrouver et celles des attributs que l'on voudrait utiliser dans ce but.

3.1.1.1 Mise en forme

La première chose à faire avant de pouvoir lancer les expériences est de transformer ces données afin qu'ils correspondent au format d'entrée très spécifique de Weka : ARFF (Attribute-Relation File Format). Un fichier ARFF est constitué d'un en-tête et d'un corps.

En-tête. Chaque fichier d'entrée commence par un en-tête dans lequel on déclare le nom de la présente base de données, mais surtout chacun des attributs avec leur nom et leur type ainsi que les classes, qui sont en fait traitées comme un attribut spécial. On aura donc au début de chacun de nos fichiers la ligne `@relation rhapsodie`, avant la déclaration des attributs, faite suivant le modèle `@attribute <nom> <type>`, où le

```
% <--- déclaration de la relation (base de données) --->
@relation rhapsodie
% <--- déclaration des attributs --->
@attribute discusivemarker { B, I, L, U, 0 }
@attribute intro { B, I, L, U, 0 }
@attribute jonct { B, I, L, U, 0 }
        [...]
@attribute context_durée(-3) real
@attribute context_durée(-2) real
@attribute context_durée(-1) real
@attribute context_durée(0) real
@attribute context_durée(1) real
@attribute context_durée(2) real
@attribute context_durée(3) real
        [...]
@attribute class { true, false }

% <--- instances (détails des données) --->
@data
L,0,0,S,_,0,W,0,0,0,0,_,0,0,0,0,0,mh,_,mm,mm,mm,mm,lm,h,_,m,m,m,m,m,3.32,2.128,0.796,
1.285,0.754,1.025,0.885,3.9821227650293487,0,0.0,0.30642110528566135,0.0,0.0,0.0,0.0,80.5
7386752455572,0,80.46180677534059,80.09978782680969,80.17011234510599,79.681211929126
25,79.11693649233516,84.59288786938436,0,81.68825906469125,81.28318000621775,79.64043
354446343,78.97067087127435,78.73833943824063,589.9999999999999,0,135.00000000000156,
280.00000000000114,179.99999999999972,129.999999999999,219.99999999999886,215.0000000
0000165,0,230.00000000000043,148.33333333333343,191.25000000000014,221.279,204.680749
99999965,P,0,0,p,0,0,0,basmontant,$,$,$,$,$,$,$L2-$L1,$L2-$L1,$L2-$L1,$L2-$L1,$L2-$L1,
$L2-$L1,$L2-$L1,NC,ADJ,I,I,CLS,V,DET,I-PP,I-PP,O,B-AdP,B-NP,B-VN,B-NP,false
        [...]
```

FIGURE 3.1: Extrait d'un fichier ARFF

type de l'attribut peut être sous forme numérique, de liste, d'une chaîne de caractères ou même de date, selon sa nature.

Corps. Après la déclaration `@data`, chaque ligne qui suit est une instance, composée d'une série de valeurs (une pour chacun des attributs, y compris la classe qui se trouve à la fin) séparées par des virgules. L'ordre des valeurs d'attributs doit rigoureusement suivre celui des attributs dans l'en-tête.
Dans le cas où une valeur est inexistante (notamment pour toutes les données qui portent sur les positions [-3 :3], nécessairement incomplètes pour chaque début et fin de fichiers), on a choisi de mettre un 0.

3.1.1.2 Transformation des données

Pour la plupart des informations (colonnes de 2.4), cette transformation consiste simplement à récupérer la valeur correspondant à l'instance (ligne), ou bien nécessite un léger traitement, comme c'est le cas pour les POS et les chunks, qui, n'ayant qu'une colonne pour la position courante (0), ont demandé un petit module pour que soient également récupérées les trois positions précédentes et les suivantes.
Le traitement le plus important reste celui des classes, annotées, pour rappel, suivant le codage BILU(O) qui est utilisé pour décrire la position de chaque mot à l'intérieur d'une unité, alors que l'on voudrait savoir uniquement si la frontière droite d'un mot est bien une frontière macrosyntaxique ou non. Pour obtenir cette information, on s'intéresse essentiellement aux extrémités des unités, à partir desquelles on peut repérer les frontières.

Frontières d'UI. On considère qu'il y a bien une frontière d'UI à droite d'un mot :

1. s'il est à la fin du fichier ;
2. s'il est annoté L ou U ;
3. s'il est suivi d'un mot annoté B ou U.

Le dernier critère peut paraître redondant avec le deuxième mais il permet, d'une part, de reconnaître les frontières gauches des UI enchâssées ou parenthétiques, et d'autre part de délimiter les UI incomplètes. En effet, comme on peut le voir dans l'exemple suivant, dans l'annotation macrosyntaxique de référence, les syntagmes inachevés ont été marqués par le signe &, signe absent de nos données.

```
[Rhap-D0003],PFC
^et les pompiers { "euh" | "euh" } & //
c'était à Nantes //
```

L'absence du signe &, auquel aurait été rattachée l'annotation L, a pour effet que dans nos données, cette phrase est étiquetée comme suit :

token	et	les	pompiers	euh	euh	c'était	à	Nantes
UI	B	I	I	I	I	B	I	I

On ne peut donc pas reposer uniquement sur les fins des UI (étiquettes L ou U) pour déterminer leurs frontières.

Frontières de CI. Pour les CI, les critères sont un peu différents puisque l'on voudrait tantôt récupérer des frontières droites (pré-noyaux), tantôt des frontières gauches (post-noyaux), tantôt les deux (in-noyaux). On considère donc qu'il y a bien une frontière de CI à droite d'un mot :

1. s'il n'est pas suivi d'un introducteur d'UI [1] ;
2. s'il est annoté L ou U dans les colonnes pré-noyaux(différés) (fin de CI) ;
3. s'il est suivi d'un mot annoté B ou U dans les colonnes in-noyaux(différés) (précède une CI) ou s'il y est annoté L ou U (fin de CI) ;
4. s'il est suivi d'un mot annoté B ou U dans les colonnes post-noyaux(différés) (précède une CI).

Le détail de ces manipulations se trouve dans le script vectorisationWeka.pl (annexe A.1).

3.1.2 Adaptation des données aux objectifs

L'application des expériences de classification sur l'ensemble de nos données complètes rendrait la tâche de classification et l'interprétation des résultats plus difficiles en raison du très grand nombre d'éléments à prendre en compte. Les données ont alors été triées pour répondre à nos besoins et à nos objectifs.

Données complètes vs. partielles La question de la falsifiabilité des données et de la possibilité de réappliquer les traitements mis en place sur de nouvelles données moins dotées implique que deux séries d'expériences doivent se faire en parallèle : dans la première série, l'ensemble de données comportera donc toutes les informations disponibles dans le corpus Rhapsodie et susceptibles d'être utiles à notre tâche de segmentation, tandis que dans la deuxième série, on a uniquement recours aux informations que l'on a récupérées de manière automatique.

1. On a considéré que les introducteurs d'UI ne faisaient pas partie des noyaux mais bien des UI. Pour distinguer ce cas particulier, on a choisi d'annoter cette frontière `O` comme on l'expliquera dans la section 3.1.2

UI vs. CI Chacune des expériences présentées dans ce chapitre se fait sur plusieurs niveaux. En effet, afin de répondre aux différents objectifs définis précédemment, on a vectorisé les données en plusieurs fichiers d'entrée correspondant à différents ensembles de classes à retrouver. On commencera donc par tenter de savoir dans un premier temps si les espaces intermots sont des frontières d'unités macrosyntaxiques ou non, avant d'aller plus loin en cherchant à typer la frontière. Par ailleurs, le nombre de types de frontières qui nous intéressent s'élève à 13 (5 types de frontières d'UI, 7 types de frontières de CI et 1 non-frontière) et représente autant de classes à retrouver. Pour faciliter la tâche des classifieurs, on traitera séparément d'un côté les frontières d'UI, et de l'autre les frontières de CI.

La déclaration des classes dans les fichiers ARFF se fera alors avec les valeurs suivantes :

- `@attribute class { true, false ; }` où `true` représente toutes les frontières d'UI ;
- `@attribute class { //, (, //), [, //], false ; }` pour le typage des frontières d'UI ;
- `@attribute class { true, false, O ; }` où `true` représente toutes les frontières de CI ;
- `@attribute class { <, <+, >, >+, (, (+, ), ///, O, false ; }` pour le typage des frontières de CI.

Les symboles du codage macrosyntaxique de Rhapsodie ont donc été conservées, mais on remarquera toutefois que dans le dernier cas deux symboles ont dû être ajoutés à cause de l'absence des ceux qui servaient à segmenter en UI. Si les frontières de noyaux n'ont pas besoin d'être explicitement délimitées dans le corpus d'origine, c'est en effet parce qu'elles le sont implicitement grâce aux symboles des autres unités syntaxiques, dont l'UI. Dans la mesure où nous avons choisi de ne pas typer à la fois les UI et les CI, on a préféré utiliser un symbole unique représentant toute frontière de noyau/d'UI. Néanmoins, les UI et les noyaux ne se confondent pas parfaitement : les introducteurs sont intégrés aux UI mais pas aux noyaux, d'où la nécessité d'adopter un autre symbole, `O` (pour OUT), pour signaler les introducteurs comme ne faisant partie d'aucune composante illlocutoire.

3.2 Expérimentations

Les expériences de classification supervisée ont été réalisées grâce au logiciel open source Weka, où sont implémentés divers types d'algorithmes de classification supervisée. Aucune de ces différentes familles d'algorithmes n'est intrinsèquement meilleure qu'une autre : elles répondent à des besoins et des objectifs différents. Certaines ont ainsi un meilleur score objectif ou ne nécessitent pas un long temps de calcul durant la phase d'apprentissage, mais reposent sur des modèles probabilistes illisibles pour un être humain.
Pour ce projet, on a choisi de travailler avec des algorithmes des familles *rules* et *trees* pour la lisibilité de leurs modèles qui permettent de visualiser respectivement les séries de règles ou les arbres de décision produits pour classifier. Parmi les classifieurs que nous avons pu tester [2], c'est le constructeur d'arbre de décision J48 qui a obtenu les meilleurs résultats.

2. Il s'agit de JRip, DecisionStump et DecisionTable en *rules*, et J48, REPTree, RandomForest ainsi que RandomTree en *trees*

3.2.1 Premiers résultats sur le corpus de base

Dans cette partie, les résultats sont présentés sous la forme de matrices de confusion, qui croisent les classes attribuées automatiquement par le modèle (colonnes) avec les classes de référence indiquées dans le corpus d'entraînement (lignes). Les croisements entre les colonnes et les lignes correspondant à la même classe ont été colorés en bleu et mettent en évidence les éléments correctement classifiés. Ce type de représentation permet donc de voir concrètement où ont été classées chacune des instances. Les arbres de décision construits seront, quant à eux, analysés dans le chapitre 4 suivant.

```
=== Confusion Matrix ===

    a    b    c   <-- classified as
    5    0    2  |     a = classe 1
    11   4    1  |     b = classe 2
    3    0    7  |     c = classe 3
```

Nombre d'items correctement classifiés (en bleu) : 5 dans la classe 1, 4 dans la classe 2 et 7 dans la classe 3.

Les valeurs qui restent en noir se lisent comme du **bruit** par colonne, et comme du **silence** par ligne.
La deuxième classe est celle qui aurait la meilleure précision puisque pas de bruit (0 item classifié en 1 et en 3) mais c'est aussi celle qui a le plus grand silence et donc qui aurait le plus faible score de rappel : sur les 16 instances appartenant à 2, seuls 4 sont correctements classifiés.

FIGURE 3.2: Exemple inventé de matrice de confusion et sa lecture

On s'attache également au pourcentage de classifications correctes (en gras sur la première ligne), mais aussi à un tableau de scores fourni par Weka, et dont on a retenu les critères objectifs classiques en fouille de données :

- la précision, soit le rapport entre le nombre d'instances correctement classifiées dans une classe donnée et le nombre total d'instances classifiées dans cette classe ;
- le rappel, soit le rapport entre le nombre d'instances correctement classifiées dans une classe donnée et le nombre total d'instances appartenant à cette classe ;
- et la F-mesure qui est la moyenne harmonique entre la précision et le rappel.

Par souci de concision, nous avons choisi de ne présenter que les résultats du classifieur qui a obtenu les meilleurs scores pour notre tâche.

3.2.1.1 Repérage des frontières

La première série d'expériences porte sur le repérage des frontières, qui est redéfinie en classification en deux classes : `true` s'il s'agit bien d'une frontière macrosyntaxique, et `false` si ce n'est pas le cas.

Résultats sur les données complètes

Frontières d'UI :

```
Correctly Classified Instances       30152               91.3 %
  Precision   Recall  F-Measure  Class
    0.858      0.453    0.593    true
    0.936      0.991    0.963    false

=== Confusion Matrix ===

     a     b   <-- classified as
  1371  2252 |     a = true
   630 28781 |     b = false
```

Commentaires : La deuxième ligne de la matrice de confusion montre clairement ici que presque la totalité des instances est classée en `false` (29411) et que pour cette classe, le classifieur se trompe peu (630 instances `true` classées en `false`), ce qui explique pourquoi la précision et le rappel de la classe `false` sont élevés comparés à la classe `true` dont le rappel est inférieur à 0.5 : plus de la moitié des frontières d'UI est classifiée `false`.

Frontières de CI :

```
Correctly Classified Instances       28886               87.4 %
  Precision  Recall  F-Measure   Class
    0.687     0.423    0.524     true
    0.896     0.963    0.928     false
    0.798     0.688    0.739       O

=== Confusion Matrix ===

     a     b     c   <-- classified as
  2074  2781    44 |     a = true
   875 26116   132 |     b = false
    69   247   696 |     c = O
```

Commentaires : Mêmes remarques que pour les frontières d'UI. On peut toutefois noter que même si la majorité des introducteurs d'UI (classe `O`) est correctement classifiée (696 contre 247+69 erreurs et un rappel supérieur à 0.5), le classifieur semble avoir plus de mal à les départager de la classe `false` que de la classe `true`.

Résultats sur les données partielles

Frontières d'UI :

```
Correctly Classified Instances       30095               91.1 %
  Precision  Recall  F-Measure    Class
    0.837     0.474    0.605      true
    0.938     0.989    0.963      false

=== Confusion Matrix ===

     a     b   <-- classified as
  1348  2275 |     a = true
   664 28747 |     b = false
```

Frontières de CI :

```
Correctly Classified Instances       28699               86.9 %
  Precision  Recall  F-Measure    Class
    0.652     0.408    0.502      true
    0.896     0.955    0.925      false
    0.753     0.783    0.767       O

=== Confusion Matrix ===

     a     b     c   <-- classified as
  2001  2847    51 |     a = true
  1008 25906   209 |     b = false
    62   158   792 |     c = O
```

Observations globales. Si on se fie au nombre d'espaces correctement classifiés, les résultats obtenus sur le corpus de test sont plutôt bons puisqu'on a au minimum 86,9% des espaces rattachés à la bonne classe. Cependant, un rapide parcours du tableau des critères de performances et des matrices de confusion nous laisse voir que ce score élevé est en très grande partie dû aux non-frontières qui constituent en fait près de 80% des espaces du corpus. En conséquence, dans tous les cas, les résultats des frontières sont largement moins bons que ceux des non-frontières, et ce, en particulier au niveau du score du rappel : le classifieur obtient de très bons scores de rappel sur la classe `false`

(toujours supérieur à 0.9) et un score correct sur `O`, au détriment de la classe true puisqu'il ne parvient pas à identifier la moitié des frontières (rappel toujours inférieur à 0.5).

De façon générale, on remarque également que la classification des frontières d'UI est meilleure que celle des frontières de CI, notamment en terme de précision mais aussi de rappel. La classification des frontières d'UI semble donc reposer sur des critères plus solides et un peu plus généralisables que ceux servant à classifier les frontières de CI.

Enfin, en ce qui concerne les deux ensembles de données, contrairement à ce dont on aurait pu s'attendre, le classifieur ne fait pas forcément mieux avec les données complètes qu'avec les données récupérées automatiquement, qui sont pourtant à la fois moins nombreuses et plus sujettes à des erreurs. Les scores sont en réalité très proches, voire même légèrement meilleurs dans le cas des UI. On peut également noter que le nombre d'éléments de la classe `O` bien classifiés est curieusement plus élevé avec l'apprentissage sur les données partielles (792 contre 696) alors que l'on ne dispose pas dans cet ensemble de données de l'annotation des introducteurs d'UI.

3.2.1.2 Typage des frontières

Dans cette seconde série d'expériences, on tâchera de classifier les frontières directement selon leur type, et non sous une classe commune `true` comme on l'a fait précédemment.

Résultats sur les données complètes

Frontières d'UI

```
Correctly Classified Instances       30081               91.1 %

=== Confusion Matrix ===

     a     b     c     d     e     f   <-- classified as
   993    31    24     4    14  1710 |     a = //
    53    11     2     0     0   167 |     b = (
    43     1    15     0     0   154 |     c = //)
     3     0     0     0     1    93 |     d = [
    25     0     0     1     0    62 |     e = //]
   515    17    25     5     3 29062 |     f = false
```

Frontières de CI

```
Correctly Classified Instances       28427               86.1 %

=== Confusion Matrix ===

     a     b     c     d     e     f     g     h     i     j   <-- classified as
    65    33     0     0     2     0     0   125     0   569 |     a = <
    40    19     0     0     2     2     0    80     1   225 |     b = <+
     1     0     4     0     0     0     0    12     3   151 |     c = >
     0     0     0     0     1     0     0     8     0    38 |     d = >+
     2     1     0     1     0     0     0    23     0   140 |     e = (
     0     1     1     0     0     0     0     3     0    15 |     f = (+
     4     0     0     0     0     0     0    12     5   150 |     g = )
    69    38     8     2     8     2     8  1395    43  1587 |     h = ///
     1     4     1     0     1     0     1    45   787   172 |     i = O
    52    31    13     2     8     4    13   658   185 26157 |     j = false
```

Résultats sur les données partielles

Frontières d'UI

```
 Correctly Classified Instances       30026               90.9 %

=== Confusion Matrix ===
```

```
    a     b     c     d     e     f   <-- classified as
  955    30    12     5     8  1766 |     a = //
   50    20     3     0     0   160 |     b = (
   43     2     9     0     0   159 |     c = //)
    4     0     2     0     2    89 |     d = [
   26     0     0     1     1    60 |     e = //]
  537    22    14     1    12 29041 |     f = false
```

Frontières de CI

```
Correctly Classified Instances       28124               85.1 %

=== Confusion Matrix ===

    a     b     c     d     e     f     g     h     i     j   <-- classified as
  106    38     0     0     0     1     2   114     2   531 |     a = <
   44    22     0     1     0     0     1    78     1   222 |     b = <+
    3     0     1     0     0     0     0    15     4   148 |     c = >
    1     0     0     0     0     0     0     8     0    38 |     d = >+
    2     1     0     0     0     0     0    24     0   140 |     e = (
    1     0     0     0     0     0     0     2     0    17 |     f = (+
    3     0     1     2     0     0     2    13     3   147 |     g = )
   95    46     7     5     8     2     9  1142    38  1808 |     h = ///
    4     2     1     0     0     0     1    51   797   156 |     i = O
   94    38    15     5     8     2    13   666   228 26054 |     j = false
```

Observations. Ces données sur le typage confirment les remarques faites à propos des résultats du repérage des frontières : encore une fois, la différence entres les deux ensembles de données, complètes et partielles, semble avoir très peu d'impact sur la classification, du côté des UI comme des CI, et la classification des frontières d'UI donne des résultats légèrement meilleurs, et ce, même au sein du typage des CI puisque la classe `///` (qui permet de segmenter les noyaux, unités pouvant se confondre avec les UI) obtient une F-mesure de 0.505 contre 0 à 0.126 pour les autres frontières de CI. La disproportion entre les classes qui entraîne un biais vers les classes majoritaires est accentuée à cause de la multitude de classes, et se trouve particulièrement observable dans les matrices de confusion ci-dessus où la plupart des instances sont clairement concentrées dans certaines colonnes (`false` ou `//` dans le cas des UI, et `false` ou `///` dans le cas des CI), tandis que les autres classes sont quasiment vides.

Si l'on cherche maintenant à comparer les résultats du repérage et du typage des frontières, on remarquera que la différence de score entre les deux types de classification n'est pas très importante : autour de -0.2% pour les UI, et -1.7% maximum pour les CI, avec dans les deux cas presque le même nombre d'instances classifiées `false`.
Le typage permet toutefois de mettre en évidence les frontières macrosyntaxiques les mieux classifiées, à savoir les frontières d'UI non-parenthétiques et non-enchâssées (classe `//`), ainsi que les frontières de noyaux non-délimités par des marqueurs de pré- ou de post-noyaux (classe `///`). Cette prédominance semble aller de pair avec le nombre d'occurrences de chacun des types de frontières dans le corpus, dont on peut se faire une idée en observant ligne par ligne les matrices de confusion.

On peut donc déduire quatre conclusions de ces premiers résultats :

1. le biais provoqué par la disproportion des classes est considérable et fausse les résultats de la classification ;
2. les frontières d'UI sont mieux classifiées que les frontières de CI ;
3. le repérage des frontières obtient des résultats légèrement meilleurs que leur typage ;
4. malgré des données moindres et entièrement calculées de manière automatique, les résultats des classifications avec l'ensemble de données partielles donnent des résultats proches de celles faites avec l'ensemble de données complètes.

3.2.1.3 Annotation du corpus

Nous avons redéfini la tâche de segmentation en une tâche de classification de chaque espace intermot. L'ensemble de ces classifications successives peut lui-même être redéfini en une tâche d'annotation puisqu'il nous permet d'obtenir, en sortie, un corpus annoté.

Prédictions à partir des modèles appris Une fois appris, les modèles construits peuvent être appliqués sur d'autres ensembles de données, que la classe à laquelle appartient chacune de leurs instances soit connue ou non, à condition qu'ils aient les mêmes attributs que ceux du corpus d'entraînement. Dans le cas de Weka, il faut également que ces données soient au format ARFF pour que le modèle puisse les traiter comme un nouveau corpus de test, et produire en sortie des prédictions de classes, sous forme d'un fichier tabulaire contenant : (1) l'identifiant de l'instance, (2) la classe à laquelle elle devrait appartenir (dans le cas où on ne disposerait pas de cette information, on aurait simplement `1:?` partout), (3) la classe prédite par le classifieur, (4) un champ indiquant si la classification est juste (champ vide) ou erronée (signe `+`) et enfin (5) une estimation de la probabilité que la classe prédite soit juste.

```
=== Predictions on test data ===

   inst#   actual  predicted error prediction
     1    6:false    6:false       0.945
     2    6:false    6:false       0.945
     3    6:false    6:false       0.945
     4    6:false    6:false       0.89
     5    6:false    6:false       0.772
     6    6:false    6:false       0.945
     7    6:false    6:false       0.945
     8    6:false    6:false       0.945
     9    6:false    6:false       0.945
    10       1://    6:false   +   0.713
    11    6:false    6:false       0.945
    12        2:(        2:(       0.75
```

FIGURE 3.3: Extrait de la sortie de l'application du modèle de classification des UI (types)

Évaluation de l'annotation En remplaçant l'identifiant de l'instance par les mots correspondants et en récupérant les colonnes 2 et 3, il est possible d'étiqueter le corpus en associant à chacun des mots du corpus la classe réelle et la classe prédite. On est ensuite en mesure d'évaluer cette annotation à l'aide d'un programme d'évaluation développé par Denys Duchier dans le cadre du projet ANR CRoTAL. Ce dernier prend en entrée un fichier au format tabulaire contenant sur chaque ligne le mot, une étiquette de référence, et l'étiquette à évaluer, comme on peut le voir en 3.4.

```
est-ce       false    false
que          false    false
vous         false    false
pourriez     false    false
décrire      false    false
euh          false    false
les          false    false
déplacements false    false
avec         false    false
précision       //    false
une          false    false
journée          (    (
```

FIGURE 3.4: Extrait du corpus transformé au format d'entrée pour le programme d'évaluation de CRoTAL

Le programme CRoTAL fournit ensuite en sortie, de la même manière que Weka, une évaluation en terme de précision, de rappel et de F-mesure [3] pour chacune des étiquettes, et un score d'exactitude (*accuracy*) qui correspond à la proportion d'étiquettes correctes sur l'ensemble du corpus étiqueté.

Sur données complètes

Frontières d'UI (type) :

```
accuracy: 0.937
    //     P: 84.29%  R: 49.86%  F: 62.65%
    (      P: 78.69%  R: 20.60%  F: 32.65%
    //)    P: 79.25%  R: 19.72%  F: 31.58%
    [      P: 60.00%  R:  6.19%  F: 11.21%
    //]    P: 73.53%  R: 28.41%  F: 40.98%
    false  P: 94.24%  R: 99.35%  F: 96.73%
```

Frontières de CI (type) :

```
accuracy: 0.9
    (      P: 63.16%  R:  7.19%  F: 12.90%
    (+     P: 57.14%  R: 20.00%  F: 29.63%
    )      P: 72.22%  R:  7.60%  F: 13.76%
    <      P: 73.43%  R: 25.06%  F: 37.37%
    <+     P: 76.64%  R: 28.46%  F: 41.50%
    >      P: 67.86%  R: 11.11%  F: 19.10%
    >+     P: 63.64%  R: 14.89%  F: 24.14%
    ///    P: 81.96%  R: 59.08%  F: 68.66%
    O      P: 81.05%  R: 85.38%  F: 83.16%
    false  P: 91.29%  R: 98.28%  F: 94.66%
```

Sur données partielles

Frontières d'UI (type) :

```
accuracy: 0.936
    //     P: 83.99%  R: 48.38%  F: 61.39%
    (      P: 78.57%  R: 23.61%  F: 36.30%
    //)    P: 83.93%  R: 22.07%  F: 34.94%
    [      P: 71.43%  R: 10.31%  F: 18.02%
    //]    P: 87.50%  R: 15.91%  F: 26.92%
    false  P: 94.18%  R: 99.43%  F: 96.73%
```

Frontières de CI (type) :

```
accuracy: 0.899
    (      P: 62.07%  R: 10.78%  F: 18.37%
    (+     P: 80.00%  R: 20.00%  F: 32.00%
    )      P: 65.38%  R:  9.94%  F: 17.26%
    <      P: 79.41%  R: 30.60%  F: 44.18%
    <+     P: 68.21%  R: 27.91%  F: 39.62%
    >      P: 75.76%  R: 14.62%  F: 24.51%
    >+     P: 63.64%  R: 14.89%  F: 24.14%
    ///    P: 80.26%  R: 54.94%  F: 65.23%
    O      P: 82.55%  R: 85.08%  F: 83.80%
    false  P: 91.15%  R: 98.36%  F: 94.62%
```

L'exactitude est finalement assez proche du pourcentage d'instances correctement classifiées lors de la phase d'apprentissage, et les proportions d'instances bien classifiées par classe sont globalement conservées.
Ces résultats serviront de baseline pour évaluer la qualité des expériences suivantes.

3.2.2 Expériences avec remaniement des données

Dans ces deux premières séries d'expériences, nous avons essayé de jouer à la fois sur la nature et sur le nombre des classes à retrouver avec des résultats plutôt infructueux puisque regrouper toutes les frontières en une classe n'augmente pas de manière sensible les scores. Dans les expériences qui suivent, on tentera donc un remaniement des données sur d'autres paramètres, notamment au niveau des instances ou même des attributs, afin d'optimiser les choix des classifieurs.

Chacune des idées entraînera une série de manipulations similaires à celle de l'-expérience précédente, que l'on évaluera de manière individuelle en comparant systématiquement les résultats de l'annotation à ceux obtenus en 3.2.1. Cette évaluation permettra de nous faire une idée de la pertinence du paramètre pour notre tâche. Par ailleurs, étant donné le peu de différences constaté entre les résultats sur données complètes et partielles, nous nous attacherons uniquement, dans un premier temps, aux résultats des données complètes, avant de faire une comparaison entre les deux versions dans la section 3.2.3.

3. Les scores sont donnés en pourcentage mais il s'agit bien des mêmes mesures. Une précision de 0.8429 sur l'ensemble du corpus revient à 84,29% pour crotal.

3.2.2.1 Séparation des monologues et des dialogues

Le corpus sur lequel nous travaillons comporte à la fois des monologues et des dialogues, une hétérogénéité qui pourrait bruiter les données et donc interférer dans la classification des frontières.
Dans un dialogue, le nombre de frontières d'unités macrosyntaxiques, en particulier d'UI, peut être plus important que dans un monologue par le fait qu'un critère de segmentation supplémentaire intervient, à savoir le changement de locuteur qui entraîne nécessairement la fin de l'UI qui le précède.

[Rhap-D0006, CFPP2000]
$L1 "mh" //
$L2 donc < "euh" on dit [oui // on veut bien parler avec vous //+ ^mais { a~ | après } le déménagement //]
$- $L1 d'accord //
$L2 vous n'avez -$ $L2 pas autour de vous cette impression que & //
$L1 non //

[Rhap-M0018]
^et au moment { où la fille passe | ^et qu'elle voit le manque d'attention du boulanger } <+ elle prend un bout de pain //
^et elle se sauve en courant //
^ mais à ce moment-là <+ elle est vue par une dame { { plutôt âgée | ^ et bonne bourgeoise } //

FIGURE 3.5: Extraits d'un dialogue (haut) et d'un monologue (bas)

Outre l'alternance de la parole et les interruptions, la présence d'un ou de plusieurs interlocuteurs implique également des chevauchements qui peuvent fausser nos données. En effet, en cas de chevauchement, il est impossible de récupérer les indices acoustiques de tous les locuteurs : il y a nécessairement une sélection, et les données (voir tableau 2.4) dont on dispose sont alors celles qui sont liées au signal le plus marqué. Ceci explique pourquoi on peut trouver dans nos données des valeurs d'attributs acoustiques qui se répètent sur plusieurs lignes, même si le locuteur est différent. Typiquement, dans le dialogue de l'exemple en 3.5, les informations acoustiques du mot "d'accord" ne sont pas alignées sur sa dernière syllabe, mais sur la dernière syllabe du groupe de mots avec lesquel il se chevauche, soit la syllabe [ve] de "vous n'avez".

Pour éviter que les caractéristiques des dialogues ne soient généralisées aux monologues (et vice-versa), on a choisi de tester les classifieurs sur les monologues et sur les dialogues séparés en deux sous-ensembles de données. Dans un premier temps, on s'est servi de ces sous-ensembles à la fois comme corpus d'entraînement du classifieur et comme corpus de test à annoter, puis nous avons limité leur utilisation à la phase de test.
Les résultats sont ceux de l'annotation, auxquels on a ajouté une colonne qui évalue la différence de F-mesure entre ces résultats et ceux obtenus lors de l'expérience initiale où les monologues et les dialogues étaient regroupés.

En tant que corpus d'entraînement et de test

Dialogues

Frontières d'UI (type) :

```
accuracy: 0.926 (-0.01)
    //      P: 80.00%  R: 47.57%  F: 59.66% -2.99%
    (       P: 76.56%  R: 22.37%  F: 34.63% +1.98%
    //)     P: 85.71%  R: 18.00%  F: 29.75% -1.83%
    [       P: 57.14%  R:  4.88%  F:  8.99% -2.22%
    //]     P: 83.33%  R: 20.00%  F: 32.26% -8.72%
    false   P: 93.35%  R: 99.15%  F: 96.16% -0.57%
```

Frontières de CI (type) :

```
accuracy: 0.89 (-0.01)
    (       P: 58.82%  R:  9.52%  F: 16.39% +3.49%
    (+      P: 100.00% R:  7.69%  F: 14.29% -15.34%
    )       P: 62.50%  R:  4.67%  F:  8.70% -5.06%
    <       P: 79.26%  R: 17.48%  F: 28.65% -8.72%
    <+      P: 76.60%  R: 29.63%  F: 42.73% +1.23%
    >       P: 80.00%  R: 11.11%  F: 19.51% +0.41%
    >+      P: 50.00%  R: 12.50%  F: 20.00% -4.14%
    ///     P: 80.63%  R: 58.67%  F: 67.92% -0.74%
    O       P: 85.76%  R: 75.17%  F: 80.12% -3.04%
    false   P: 89.99%  R: 98.23%  F: 93.93% -0.73%
```

Monologues

Frontières d'UI (type) :

```
accuracy: 0.956 (+0.02)
    //      P: 85.96%  R: 52.21%  F: 64.96% +2.31%
    (       P:    N/A  R:    N/A  F:    N/A
    //)     P: 100.00% R:  7.69%  F: 14.29% -17.29%
    [       P: 50.00%  R:  6.67%  F: 11.76% +0.55%
    //]     P: 100.00% R: 15.38%  F: 26.67% -14.31%
    false   P: 96.04%  R: 99.46%  F: 97.72% +0.99%
```

Frontières de CI (type) :

```
accuracy: 0.907 (+0.007)
    (       P: 57.14%  R:  6.45%  F: 11.59% -1.31%
    (+      P: 66.67%  R: 28.57%  F: 40.00% +10.37%
    )       P: 61.11%  R: 17.19%  F: 26.83% +13.07%
    <       P: 66.67%  R:  6.59%  F: 12.00% -25.37%
    <+      P: 84.62%  R:  8.73%  F: 15.83% -25,67%
    >       P: 100.00% R:  3.70%  F:  7.14% -11.96%
    >+      P: 100.00% R: 40.00%  F: 57.14% +33%
    ///     P: 82.57%  R: 57.42%  F: 67.73% -0.93%
    O       P: 89.07%  R: 59.27%  F: 71.18% -11.98%
    false   P: 91.31%  R: 99.32%  F: 95.15% +0.49%
```

Observations. Le taux d'espaces correctement classifiés sur l'ensemble du corpus ne change pas beaucoup. Dans le détail, les résultats sont un peu moins bons dans le cas des dialogues, avec une dégradation des scores répartie sur toutes les classes sauf une pour les frontières majeures, et sauf trois pour les frontières mineures. En revanche, pour les monologues, la comparaison est plus mitigée : si on perd sur des classes peu représentées (notamment les frontières droites d'UI parenthétiques `//)` et d'enchâssement `//]` ainsi que les pré-noyaux `<` et `<+`), on gagne sur les classes les plus représentées. À noter enfin : dans l'ensemble, là où les dialogues perdent, les monologues gagnent, et vice-versa. Cette remarque implique que les proportions des classes ne sont pas les mêmes dans les deux types de corpus.

En tant que corpus de test uniquement

Dialogues

Frontières d'UI (type) :

```
accuracy: 0.927 (-0.01)
    //      P: 83.53%  R: 46.91%  F: 60.08% -2.57%
    (       P: 78.69%  R: 21.92%  F: 34.29% +1.64%
    //)     P: 82.98%  R: 19.50%  F: 31.58% =
    [       P: 66.67%  R:  7.32%  F: 13.19% +1.98%
    //]     P: 84.00%  R: 28.00%  F: 42.00% +1.02%
    false   P: 93.32%  R: 99.38%  F: 96.25% -0.48%
```

Frontières de CI (type) :

```
accuracy: 0.896 (-0.004)
    (       P: 58.82%  R:  9.52%  F: 16.39% +3.95%
    (+      P: 66.67%  R: 15.38%  F: 25.00% -4.63%
    )       P: 55.56%  R:  4.67%  F:  8.62% -5.14%
    <       P: 75.12%  R: 26.14%  F: 38.79% +1.42%
    <+      P: 77.32%  R: 30.86%  F: 44.12% +2.62%
    >       P: 73.91%  R: 11.81%  F: 20.36% +1.26%
    >+      P: 40.00%  R:  6.25%  F: 10.81% -13,33%
    ///     P: 83.07%  R: 59.26%  F: 69.17% +0.51%
    O       P: 83.44%  R: 84.80%  F: 84.12% +0.96%
    false   P: 90.67%  R: 98.26%  F: 94.31% -0.35%
```

Monologues

Frontières d'UI (type) :

```
accuracy: 0.96 (+0.023)
    //      P: 86.28%  R: 59.36%  F: 70.33% +7.68%
    (       P:    N/A  R:    N/A  F:    N/A
    //)     P: 50.00%  R: 23.08%  F: 31.58% =
    [       P:    N/A  R:    N/A  F:    N/A
    //]     P: 44.44%  R: 30.77%  F: 36.36% -4.62%
    false   P: 96.55%  R: 99.27%  F: 97.89% +1.16%
```

Frontières de CI (type) :

```
accuracy: 0.912 (+0.012)
    (       P: 100.00% R:  3.23%  F:  6.25% -6.65%
    (+      P: 50.00%  R: 28.57%  F: 36.36% +6,73%
    )       P: 88.89%  R: 12.50%  F: 21.92% +8.16%
    <       P: 67.24%  R: 21.43%  F: 32.50% -4.87%
    <+      P: 75.00%  R: 23.81%  F: 36.14% -5.36%
    >       P: 40.00%  R:  7.41%  F: 12.50% -6.6%
    >+      P: 83.33%  R: 33.33%  F: 47.62% +23.48%
    ///     P: 77.71%  R: 58.37%  F: 66.67% -1.99%
    O       P: 75.39%  R: 86.91%  F: 80.74% -2.42%
    false   P: 92.83%  R: 98.33%  F: 95.50% +0.84%
```

Observations. Dans l'ensemble, cette expérience a des résultats un peu plus positifs que la précédente où deux modèles avaient été entraînés. En effet, les tendances sont similaires mais pour les monologues elles sont aussi moins marquée que celles observées lors de l'expérience précédente en terme de proportion (moins de gain et surtout moins de perte). Quant aux dialogues, le modèle appris sur le corpus entier permet de mieux classifier dans les classes peu représentées pour les dialogues, et pour cause : la majorité des UI parenthétiques et des enchâssements se trouvent dans les dialogues.

3.2.2.2 Pré-sélection des instances avec SEM

Une des raisons qui pourraient expliquer le fort taux d'erreurs de classification dans l'expérience initiale était l'importante disproportion entre le nombre d'espaces intermots représentant une frontière syntaxique et le nombre d'espace intermots qui n'en sont pas. Pour avoir une idée précise de ce déséquilibre, les frontières de CI tous types confondus ne représentent que 4,8% des frontières du corpus entier (1573 occurrences sur 33034). Un modèle construit avec des classes plus équilibrées pourrait avoir des résultats moins

biaisés.

Avec l'attribut `chunk` de nos données, on dispose déjà d'une analyse syntaxique superficielle permettant d'identifier les potentielles frontières non-pertinentes pour la classification. Pour cela, on choisit de considérer que si le mot qui suit le mot courant n'est ni un début de chunk (étiquette `B`) ni hors chunk (étiquette `O`), mais aussi que sa classe de référence est `false`, on ne l'inclut pas dans le corpus d'entraînement. La seconde condition est nécessaire pour limiter la perte d'information dans les autres classes déjà sous-représentées. De cette manière, on réduit de 14301 le nombre d'instances `false` sans faire de suppression arbitraire.
Précisons enfin que cette pré-sélection des instances n'a lieu d'être que dans le corpus d'entraînement et non dans le corpus de test puisque l'objectif est d'annoter le corpus entier.

Frontières d'UI (type) :

```
accuracy: 0.557 (-0.38)
    //     P: 11.92%  R: 64.34%  F: 20.11% -42.54%
    (      P: 32.81%  R: 18.03%  F: 23.27% -9.38%
    //)    P: 61.39%  R: 29.11%  F: 39.49% +7.91%
    [      P:    N/A  R:    N/A  F:    N/A
    //]    P: 70.83%  R: 19.32%  F: 30.36% -10.62%
    false  P: 92.58%  R: 55.61%  F: 69.49% -27.24%
```

Frontières de CI (type) :

```
accuracy: 0.536 (-0.364)
    (      P: 33.33%  R:  9.58%  F: 14.88% +1.98%
    (+     P: 50.00%  R: 15.00%  F: 23.08% -6.55%
    )      P: 18.00%  R: 10.53%  F: 13.28% -0.48%
    <      P: 16.00%  R: 40.05%  F: 22.86% -14.51%
    <+     P: 12.94%  R: 40.92%  F: 19.66% -21.84%
    >      P: 14.66%  R: 16.37%  F: 15.47% -3.63%
    >+     P: 11.59%  R: 17.02%  F: 13.79% -10.35%
    ///    P: 18.10%  R: 72.06%  F: 28.93% -39.73%
    O      P: 75.00%  R: 83.89%  F: 79.20% -3.96%
    false  P: 89.07%  R: 51.72%  F: 65.44% -29.22%
```

Observations. Par rapport à l'expérience initiale, que ce soit pour le typage des frontières d'UI ou de CI, l'exactitude perd pratiquement 0.4 points, et on observe une chute globale de tous les scores (sauf pour deux classes), en particulier des classes les plus représentées statistiquement : les non-frontières (classe `false`), dont le rappel est divisé par deux, et les frontières de noyaux (classe `///`) et d'UI non-parenthétiques et non-enchâssées (classe `//`), qui perdent, eux, en précision (respectivement -63,86% et -72,37%). Les critères de l'arbre de décision pour classifier en `false` semblent alors plus sélectifs tandis que ceux des classes `//` et `///` engendrent beaucoup de bruit.
Cette dégradation considérable des scores n'était pourtant pas prévisible à partir des résultats de la validation croisée : si on regarde les scores de F-mesure des classes citées (figure 3.6 ci-dessous), ils sont corrects pour les noyaux, très bons pour les frontières non-syntaxiques mais très mauvais pour la frontière gauche des in-noyaux (classe `(`). Elle peut être due soit au fait que ce modèle utilise des règles trop spécifiques à l'échantillon de corpus ayant servi à l'apprentissage (cas de sur-apprentissage), soit au fait que la pré-sélection, qui repose sur une annotation automatique non-corrigée, ait éliminé des exemples pertinents.

C'est surtout le rappel que l'on cherchait à améliorer avec cette pré-sélection, et on observe effectivement une hausse du score du rappel pour toutes les frontières (sauf pour la frontière droite des greffes `//]`) mais cette hausse ne parvient pas à compenser la très large perte sur la précision.

Precision	Recall	F-Measure	Class
0.026	**0.006**	**0.01**	**(**
0	0	0	(+
0.036	0.006	0.01	)
0.363	0.179	0.24	<
0.12	0.054	0.075	<+
0.063	0.018	0.027	>
0	0	0	>+
0.632	**0.571**	**0.6**	///
0.783	0.795	0.789	O
0.863	**0.943**	**0.901**	**false**

FIGURE 3.6: Évaluation de la classification des frontières de CI (typage) par validation croisée avec pré-sélection SEM

3.2.3 Comparaison avec les résultats sur données partielles

Comme dans l'expérience sur le corpus de base, le fait que l'on ne dispose pas d'attributs tirés d'annotations manuelles ni d'informations corrigées mis à part le part-of-speech des marqueurs discursifs, n'entraîne qu'une petite marge d'erreur supplémentaire.

	Données complètes		Données partielles	
	monologues	dialogues	monologues	dialogues
UI	0.96	0.93	0.96	0.93
CI	0.91	0.89	0.91	0.89

TABLE 3.1: Tableau comparatif des résultats de la séparation des monologues et des dialogues dans le corpus de test

On observe donc la même fluctuation des scores que sur les données complètes, et arrondie au centième les scores sont identiques (figure 3.1).

3.2.4 Conclusions

Aucune de nos manipulations n'a eu de résultats vraiment concluants pour améliorer la classification des espaces en frontières. Le meilleur classifieur parmi ceux que nous avons testé est basé sur le constructeur d'arbre de décision J48, et permet de classifier correctement 89% des espaces intermots, si on se fie aux résultats de la figure 3.8 qui combinent les annotations des frontières d'UI et de CI[4].
N'apparaissent pas de cette évaluation les classes `///` et `O`, puisqu'elles servaient essentiellement à pallier l'absence des marqueurs de frontières d'UI. Par ailleurs, la classe nommée simplement `(` regroupe ici à la fois la frontière gauche d'une UI parenthétique, et d'un in-noyau.

Ce score est toutefois à pondérer. Il inclut en effet la reconnaissance des non-frontières. Si on ne voudrait évaluer que les frontières et que l'on retire les espaces intermots qui ne sont pas des frontières syntaxiques, et qui ont bien été classifiés comme `false`, l'exactitude tombe à moins de 0.45 points (figure 3.8). Sur le corpus entier, le meilleur des modèles que nous avons appris utilise donc des critères qui permettent d'identifier relativement efficacement les frontières macrosyntaxiques (en moyenne 0.65 de précision, et jusqu'à 0.8 pour les frontières d'UI simples) mais ne parvient pas à couvrir la moitié des

4. Nous avons obtenu ce résultat en combinant les résultats de l'application des modèles de classification des frontières d'UI et de CI avec le script formatEvalCombi.pl, en annexe. En cas de conflit de classe sur une instance, on a considéré que la classe UI était prioritaire sur la classe CI.

```
accuracy: 0.897
        //     P: 80.53%  R: 59.97%  F: 68.75%
        (      P: 77.33%  R: 14.91%  F: 25.00%
        //)    P: 79.25%  R: 19.72%  F: 31.58%
        [      P: 60.00%  R:  6.19%  F: 11.21%
        //]    P: 73.53%  R: 28.41%  F: 40.98%
        (+     P: 66.67%  R: 13.33%  F: 22.22%
        )      P: 73.33%  R:  6.67%  F: 12.22%
        <      P: 75.39%  R: 24.81%  F: 37.33%
        <+     P: 77.52%  R: 28.17%  F: 41.32%
        >      P: 70.83%  R: 10.18%  F: 17.80%
        >+     P: 66.67%  R: 14.63%  F: 24.00%
        false  P: 90.65%  R: 98.79%  F: 94.54%
```

FIGURE 3.7: Évaluation des UI et des CI combinés sur le corpus de base

occurrences (rappel très bas, allant de 0.06 à 0.6).

```
accuracy: 0.428
        //   P: 91.73%  R: 59.97%  F: 72.52%
        (    P: 85.29%  R: 14.91%  F: 25.38%
        //)  P: 85.71%  R: 19.72%  F: 32.06%
        [    P: 85.71%  R:  6.19%  F: 11.54%
        //]  P: 83.33%  R: 28.41%  F: 42.37%
        (+   P: 100.00% R: 13.33%  F: 23.53%
        )    P: 100.00% R:  6.67%  F: 12.50%
        <    P: 81.43%  R: 24.81%  F: 38.03%
        <+   P: 84.75%  R: 28.17%  F: 42.28%
        >    P: 77.27%  R: 10.18%  F: 17.99%
        >+   P: 75.00%  R: 14.63%  F: 24.49%
```

FIGURE 3.8: Évaluation des UI et CI combinés sur le corpus de base, sans la classe `false`

On a néanmoins des résultats un peu plus convaincants si on ne cherche pas à classifier tous les types de frontières sur l'ensemble du corpus. Parmi les expériences faites, le meilleur score que l'on obtient se fait effectivement en appliquant, aux monologues seuls, le classifieur appris sur l'ensemble du corpus :

```
accuracy: 0.512
    //     P: 86.28%  R: 59.36%  F: 70.33%
    (      P:    N/A  R:    N/A  F:    N/A
    //)    P: 50.00%  R: 23.08%  F: 31.58%
    [      P:    N/A  R:    N/A  F:    N/A
    //]    P: 44.44%  R: 30.77%  F: 36.36%
    false  P:    N/A  R:    N/A  F:    N/A
```

FIGURE 3.9: Évaluation des UI (types) sur les monologues uniquement, et sans la classe `false`

Chapitre 4

Analyse des résultats

Nous avons jusqu'à présent non seulement pu observer directement les résultats des séries d'expériences réalisées grâce aux matrices de confusion, mais nous avons également pu les évaluer selon différents critères (précision, rappel, F-mesure) et sous différents angles (classification, puis annotation).
Dans ce nouveau chapitre, nous tenterons de comprendre les résultats obtenus, en analysant notamment les critères retenus par les différents classifieurs dans les arbres de décision, avant de les appréhender non plus en terme de classification ou d'annotation mais sous l'angle initial de ce mémoire, celui de la segmentation.

4.1 Analyse de la classification

Pour chacune de nos expériences, nous avons observé un biais vers la classe majoritaire dû à la disproportion entre la classe `false` et les autres. Il s'agit certes du déséquilibre le plus important entre les classes, mais ce n'est pas le seul : on peut également remarquer un certain nombre de disparités dans les résultats des autres classes. Si nous classons les résultats de l'évaluation des UI et des CI combinés de la figure 3.8, nous obtenons le tableau suivant :

Classes	Rappel	F-mesure
`//`	> 0.5	> 0.5

`//]`
`<`
`<+`
	< 0.5	< 0.5

`//)` `(`*parenthèse*	< 0.2	< 0.35

`>`
`>+`
`(`*in-noyau*
`(+`
	< 0.15	< 0.25

`[` `)`	< 0.1	< 0.15

TABLE 4.1: Tableau comparatif des résultats de l'évaluation de toutes les classes de l'expérience initiale sur données complètes selon le taux de rappel et de F-mesure

La précision étant plutôt bonne et sensiblement la même pour toutes les classes (au-

tour de 0.55 pour [, < et > et 0.7 pour les autres), nous avons choisi de ne pas la représenter dans ce tableau et de ne considérer que le rappel et la F-mesure pour regrouper nos classes par tranche allant de moins de 0.1 à plus de 0.5 de rappel et de F-mesure.

Pour expliquer ce contraste, on peut se reporter à la figure 4.1 suivante où sont comparés le nombre d'occurrences de chaque classe [1] et le rappel [2]. L'ordre des classes est sensiblement le même dans le tableau précédent. La classe qui obtient les meilleurs résultats, celle des frontières d'UI //, se trouve en effet être la plus représentée de nos données d'entraînement, tandis que la moitié inférieure du tableau contient à peu près les mêmes éléments mais dans un ordre légèrement différent. Appuyant cette idée, les post-noyaux > se trouvent loin dans le tableau comparés aux pré-noyaux <, et semblent donc plus difficiles à classifier en raison d'une fréquence quatre fois inférieure.

	rhapsodie	**monologues**	**dialogues**	Rappel
//	2776	657	2119	0.499
<	794	182	612	0.251
<+	369	127	243	0.285
(*parenthèse*	233	14	219	0.206
//)	213	13	200	0.197
>	171	27	144	0.111
)	171	64	107	0.076
(*in-noyau*	167	62	105	0.072
[	96	15	82	0.062
//]	88	13	75	0.284
>+	47	15	32	0.149
(+	20	7	13	0.2

FIGURE 4.1: Nombre d'occurrences de chaque classe (UI puis CI) calculées à partir des matrices de confusion et rangées par ordre décroissant

Si on se réfère à ces observations, on peut en déduire qu'il y a un lien très fort entre le taux de rappel des résultats, et la fréquence d'une classe dans les données d'entraînement. Un grand nombre d'occurrences d'une classe semble effectivement permettre au classifieur de retenir des critères couvrant un plus grand nombre de cas et ainsi d'obtenir un meilleur taux de rappel, même si cela signifie aussi qu'il y a un plus grand nombre d'items à retrouver. Cette remarque est importante dans la mesure où si l'on cherche surtout à avoir un bon rappel et pas forcément une bonne précision pour faciliter le travail ultérieur d'un expert qui n'aurait plus qu'à trier, on pourrait se permettre de sur-échantillonner artificiellement les classes les moins représentées et profiter de l'effet de sur-apprentissage qui en résulterait.

Néanmoins, les résultats de certaines classes ne sont pas prévisibles en fonction de leur fréquence. À partir de ces deux mêmes tableaux, on note que la répartition dans les différentes tranches ne suit pas tout à fait proportionnellement l'ordre établi par le

1. Calculé suivant les matrices de confusion après classification
Remarque : les parenthèses des UI parenthétiques et des in-noyaux et les crochets des greffes sont dépariées dans ces données à cause du fait que l'on ne puisse pas cumuler les classes sur une seule position : en cas de conflit, seule la première est assignée selon l'ordre établi dans le script A.1.
2. Récupéré de l'expérience initiale en 3.2.1

nombre d'occurrences des classes.
La colonne *Rappel* en 4.1 confirme cette intuition :

1. les UI parenthétiques ont presque quatre fois moins d'occurrences que les pré-noyaux `<` mais leur taux de rappel est à peine 0.06 inférieur ;
2. à l'inverse, l'in-noyau compte exactement le même nombre d'occurrences que le post-noyau `>` (et seulement 4 occurrences de moins pour la frontière droite) mais a un rappel inférieur de 0.04 ;
3. tous les items différés (pré-noyaux `<+`, post-noyaux `>+` et in-noyaux `(+`) ont un taux de rappel plus élevé que leur contrepartie non-différée (respectivement `<`, `>` et `(`) ;
4. les frontières droites des greffes `//]` se retrouvent avec le troisième meilleur rappel juste derrière les pré-noyaux différés `<+`, quatre fois supérieurs en nombre et devant les pré-noyaux `<`, neuf fois supérieurs.

Si on examine de plus près ces quatre cas particuliers, on constate tout d'abord que, sur l'ensemble du corpus (dialogues et monologues confondus), 45% des parenthèses formant une UI (1) coïncident avec une alternance de tour de parole ou avec un chevauchement, information que nous avons par le biais de l'attribut `locuteur` et qui peut donc être utilisée par le classifieur. Dans le modèle de classification des UI, il est d'ailleurs l'attribut le plus utilisé pour classifier les `//)` juste devant le contour global syllabique et le ralentissement [3].
Les choses sont un peu différentes pour les greffes (4) dont seule la frontière droite a un score correct et très largement supérieur à celui de la frontière gauche. On peut déduire d'une telle différence que la frontière gauche des greffes est prosodiquement moins marquée que celle de droite, et il en est de même pour le cas de l'in-noyau (2), probablement moins marqué que d'autres unités comme l'UI parenthétique ou le post-noyau.
Enfin, les items différés (3) vont à l'encontre de notre théorie et semblent être plutôt avantagés par le fait qu'ils aient moins d'occurrences que leur contrepartie non-différée. En effet, ils forment des groupes plus petits mais ont a peu près la même répartition des occurrences : une grande majorité en `false`, quasiment aucune confusion avec la contrepartie non-différée, moins d'un tiers correctement classifié et le reste dans une autre classe (tableau 4.2).

	`<+`	`<`	`(+`	`(`	`>+`	`>`
correct	28%	25%	20%	7%	15%	11%
`false`	55%	65%	75%	82%	74%	85,5%
contrepartie	5%	1%	N/A	N/A	N/A	N/A
autre	12%	9%	5%	11%	11%	3,5%

TABLE 4.2: Répartition des occurrences des classes différées et de leur contrepartie non-différée

4.2 Analyse des arbres de décision

Le classifieur qui a obtenu les meilleurs scores sur nos données, et dont on présente les résultats depuis le chapitre 3 est J48, un algorithme de classification supervisé qui procède en cherchant à chaque étape l'attribut qui permet de mieux discriminer les

3. Informations observables directement dans les arbres de décision, présentés dans la section suivante.

données. À partir de là, un seuil est fixé (dans le cas de données numériques) pour diviser les données et ce nœud devient un critère de classification qui va en générer d'autres de manière récursive, jusqu'à pouvoir répartir chaque instance dans une des classes. Il s'agit d'une implémentation de la méthode *Divide and Conquer*, diviser pour régner.

L'ensemble de ces critères hiérarchisés peuvent être organisés en une arborescence d'attributs, représentation fournie par Weka sous la forme d'un arbre textuel (voir B avec possibilité de le visualiser en un arbre graphique. Dans ce projet, la plupart des attributs ont de longues listes de valeurs : les nœuds sont alors divisés en autant de branches que de valeurs possibles, ce qui explique la taille considérable de certains de nos arbres (jusqu'à 5010 feuilles, et une profondeur de 12 pour l'arbre de classification des CI sur données complètes) et, par conséquent, le fait qu'on ne puisse pas utiliser directement les graphes de Weka, alors illisibles car exhaustifs.

4.2.1 Lecture des arbres

Nous avons donc décidé de ne présenter dans cette partie qu'un échantillon de chaque arbre, en descendant jusqu'au troisième niveau de profondeur afin de n'isoler que les attributs les plus discriminants pour chaque classification. Les arbres textuels équivalents se trouvent quant à eux en annexe (B).

Ces arbres de décision permettent de répondre à certains points de notre problématique. Trois éléments nous intéressent particulièrement :

1. les attributs, et en particulier ceux qui sont proches de la racine de l'arbre, pour savoir quels paramètres (morpho-syntaxiques ou acoustiques) permettent de discriminer le mieux les classes ;
2. les seuils retenus et les valeurs de ces attributs ;
3. et enfin le taux de classification effective et d'erreur sur ces valeurs.

Dans les arbres de décision textuels dont on voit un extrait en 4.2, les attributs (en rouge dans notre exemple) se trouvent en début de ligne, indentés selon le niveau de profondeur et sont suivis des différentes valeurs qu'ils peuvent avoir.
On dénombre trois situations différentes :

- Si ces valeurs déterminent immédiatement une classe, celle-ci est indiquée selon le format `x: classe (y/z)` où `x` est une valeur d'attribut, `y` le nombre d'instances correctement classifiées et `z` le nombre d'instances mal classifiées (s'il y a lieu).
- Si elles doivent être combinées à d'autres critères pour déterminer une classe, elles engendrent un nouveau niveau de classification [4].
- Enfin, si une valeur ne permet pas de classifier (cas en gris dans notre exemple), l'algorithme lui attribue la classe majoritaire de la valeur suivie de `(0,0)`. Cette distinction permet que soit attribuée une classe à chacune des instances même en l'absence d'exemple pertinent dans le corpus d'entraînement.

Nos graphes ne représentent que le début des arbres, avec comme point de départ la racine, que l'on appelle par la suite le "premier niveau", jusqu'à arriver au troisième niveau de profondeur. Les attributs y sont entourés de cercles bleus d'où partent des

4. Puisque nous avons choisi de ne représenter que les trois premiers niveaux de profondeur des arbres, les niveaux au-delà de cette limite sont tronqués. Il s'agit dans nos échantillons des valeurs sans informations entre parenthèses derrière.

```
|   pos(1) = CLS
|   |   pos(0) = ADJ, _ADJ: // (3.0/1.0), ADJWH: // (0.0), ADV, _ADV, ADVWH:
 // (0.0), CC: false (7.0), _CC: // (0.0), CLO: false (1.0), CLR: false (1.
0), CLS: // (0.0), _CLS: // (0.0), CS: false (12.0/1.0), _CS: false (6.0/1.0),
DET: // (0.0),_DET: // (2.0/1.0), DETWH: // (0.0), ET: false (1.0), I, _I: //
 (0.0), NC, _NC: // (2.0/1.0), NPP: // (6.0/1.0), _NPP: // (1.0), P: [ (2.0
/1.0), _P: // (0.0), P+D: false (1.0), P+PRO: false (1.0), PREF: // (0.0),
PRO, _PRO: // (0.0), PROREL: false (5.0/1.0), PROWH: // (0.0), V, _V: // (2.0
/1.0), VIMP: // (0.0), VINF, _VINF: // (2.0), VPP: // (9.0/1.0), _VPP: // (0.
0), VPR: // (0.0), _VPR: // (0.0), VS: // (1.0), 0: // (0.0)
```

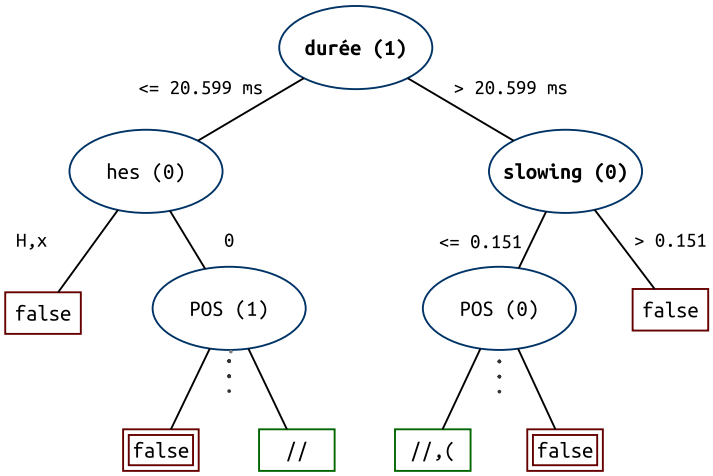
FIGURE 4.2: Extrait de l'arbre de décision de la classification en UI (annexe B.1)

branches étiquetées de leurs valeurs, tandis que les classes se trouvent dans des rectangles (rouges pour la classe `false` et verts pour les autres). Les bords des classes sont doublés s'il s'agit de la classe majoritaire pour la valeur à laquelle elles sont rattachées. Pour des raisons de clarté, le taux de réussite et d'erreur a été omis et les valeurs ne sont indiquées que jusqu'au deuxième niveau de profondeur. Les valeurs du troisième niveau peuvent cependant être consultées en détail dans les arbres textuels de l'annexe B. Par ailleurs, les attributs prosodiques ont été mis en gras afin de mettre en évidence leur rôle dans la classification.

Dans les sections suivantes, nous allons comparer les arbres construits lors de l'expérience initiale 3.2.1 par type de classes : dans un premier temps, nous confronterons les attributs utilisés pour classifier les frontières UI sur les données complètes et partielles, puis nous ferons de même pour la classification des frontières de CI.

4.2.2 Classification des UI

Sur données complètes



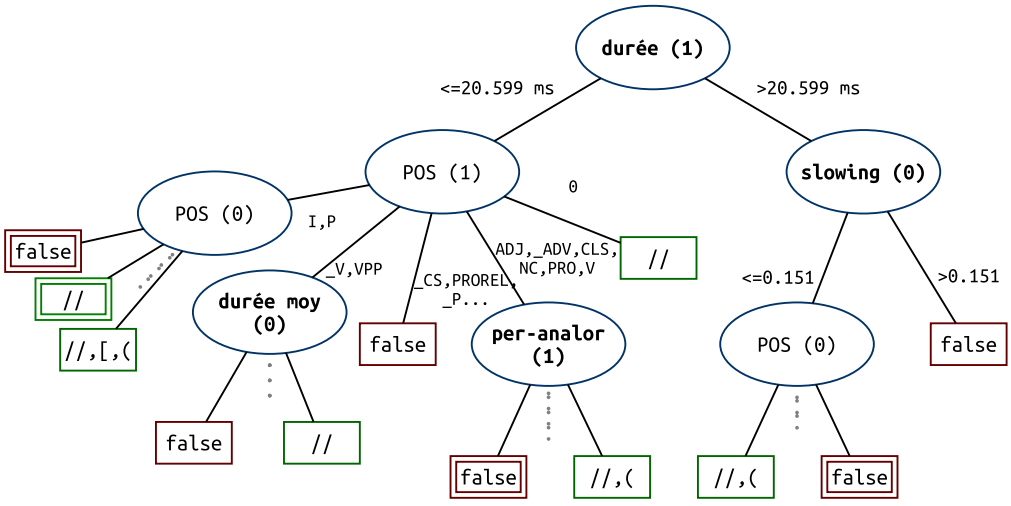
Premier niveau. Le classifieur distingue en premier lieu deux groupes d'instances : celles qui sont suivies d'une syllabe dont la durée est inférieure ou égale à 20.599 ms, et celles qui sont suivies d'une syllabe dont la durée est supérieure à cette même valeur. Il

s'agit bien de la durée de la syllabe suivante et non courante comme on aurait pu l'anticiper d'après l'observation selon laquelle en français la syllabe qui précède une frontière syntaxique ou prosodique majeure subit un allongement corrélé à l'accent final. Dans un second temps, sont examinées les syllabes courantes.

Branche gauche. Si la syllabe courante est une disfluence (marqueur d'hésitation ou indéterminé), alors la frontière droite du mot courant n'est pas une frontière syntaxique majeure, sinon il faut regarder la catégorie syntaxique du mot qui suit pour pouvoir classifier en `false` ou en `//`, ou bien s'appuyer sur d'autres critères (pointillés).

Branche droite. Si la syllabe courante a un coefficient de ralentissement supérieur à 0.151, le mot courant n'est pas suivi d'une frontière syntaxique majeure. Dans le cas inverse, on s'en remet une nouvelle fois à la catégorie syntaxique mais cette fois à celle du mot courant.

Sur données partielles



Premier niveau. L'attribut racine de l'arbre construit sur les données partielles est exactement le même que celui des données complètes : il s'agit de la durée du mot suivant.

Branche gauche. Pour remplacer l'attribut permettant de typer les disfluences (attribut perceptif), le classifieur choisit la catégorie syntaxique du mot suivant qui permet déjà de classifier tantôt en `false` tantôt en `//`, mais complexifie considérablement ce deuxième niveau de classification étant donné le nombre de possibilités de valeurs. Sur le graphe, on n'a alors représenté que les classes immédiates et les trois attributs les plus fréquents parmi ceux du troisième niveau. On retrouve alors une nouvelle fois le POS mais du mot courant, et deux attributs prosodiques : la durée moyenne de la syllabe courante et la présence ou non d'une frontière de période selon Analor.

Branche droite. Cette branche est identique à la branche droite de l'arbre précédent – jusqu'à ce niveau du moins. On peut donc se reporter à l'analyse prédédente.

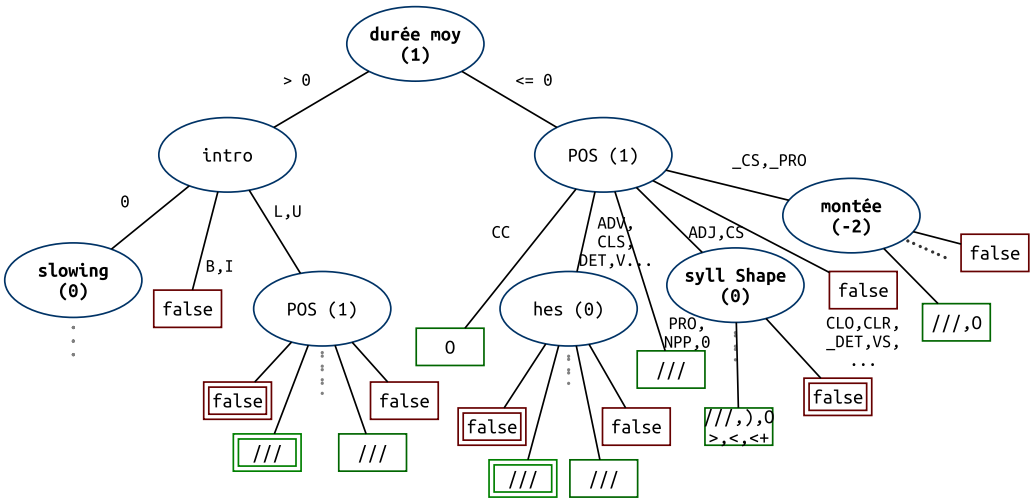
Analyse globale

Les premiers paramètres utilisés pour classifier les frontières d'UI sont sensiblement les mêmes dans l'expérience avec tous les attributs, et celle avec uniquement les attributs dont les valeurs ont été calculées automatiquement. On remarque notamment que le premier critère est le même, et qu'une des deux branches qui en découle est également identique. À noter également que le nombre d'attributs acoustiques (en gras) est à peu près égal à celui des attributs perceptifs et morpho-syntaxiques.

À ce niveau de profondeur, on dispose d'ores et déjà de critères pour classifier en `false`, mais aussi discriminer les frontières d'UI simples (`//`), les frontières gauches d'UI parenthétique (`(`) ainsi que celles des greffes (`[`) pour l'arbre construit à partir des données partielles.

4.2.3 Classification des CI

Sur données complètes



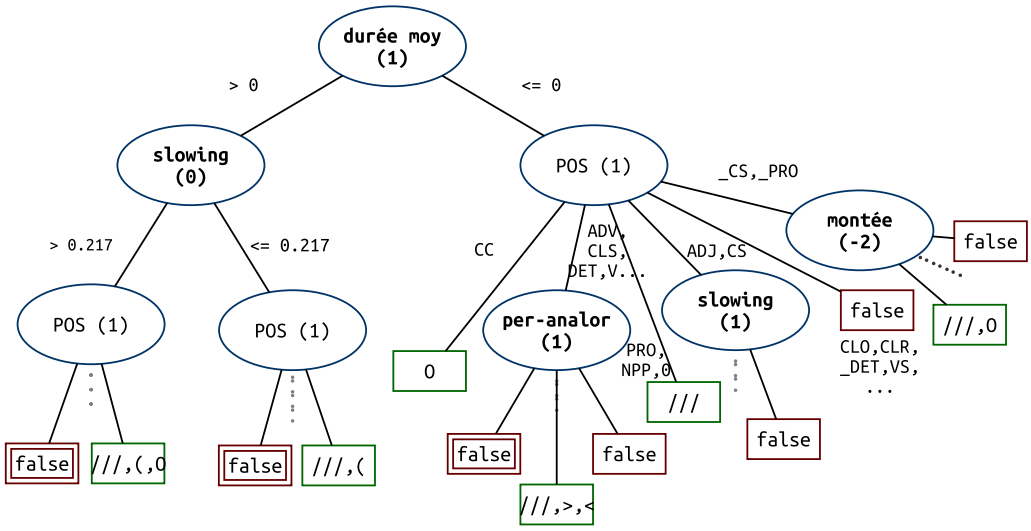
Premier niveau. Comme pour la classification des frontières d'UI, le premier attribut utilisé est un attribut acoustique, celui de la durée (ici, moyenne). Ce critère distingue littéralement les instances dont la syllabe suivante a une durée inférieure à 0 des autres, ce qui n'a pas vraiment de sens compte tenu du fait que ce soit une mesure de durée. Il s'agit en fait de repérer les syllabes suivies d'une pause (champ vide dans nos données, remplacé par un `0` pour éviter d'avoir une valeur vide).

Branche gauche. Si la syllabe suivante est une pause, on regarde en second lieu s'il s'agit d'un introducteur d'UI : si oui (valeur `U`), on se tourne ensuite vers le POS du mot suivant, en sachant que dans la plupart des cas l'espace en question sera classifié frontière majeure (`///`) ; sinon (valeur `0`), le prochain critère est le coefficient de ralentissement

(`slowing`). Dans le cas où l'on aurait affaire à un groupe d'introducteurs, seule la frontière en fin du groupe peut potentiellement être une frontière syntaxique majeure. Les mots en début (`B`) ou à l'intérieur (`I`) du groupe mènent donc directement à la classe `false` pour ne pas couper au milieu du groupe.

Branche droite. Si la syllabe suivante n'est pas une pause, alors on s'en remet au POS du mot suivant. Une partie des POS ne permet pas de classifier immédiatement et mène à d'autres critères : la présence d'une disfluence ou le contour syllabique en position 0 (mot (ou syllabe) courant(e)), ou bien la valeur de la montée en position -2 (deux syllabes avant la syllabe courante). Quant à l'autre partie, elle mène directement à trois classes : si le mot suivant est une conjonction de coordination (`CC`), alors l'espace courant est considérée comme une frontière mineure précédant un introducteur d'UI (classe `O`), si c'est une pronom (`PRO`), un nom propre (`NPP`) ou si on ne dispose pas de POS, alors c'est une frontière majeure (classe `///`). Enfin, si on se trouve dans un autre cas, notamment si le mot qui suit est un clitique autre qu'un clitique sujet (`CLO`,`CLR`), ou bien un verbe (`VINF`, `VS`), alors l'espace courant n'est pas une frontière syntaxique.

Sur données complètes



Premier niveau. Encore une fois, le premier attribut servant à classifier est le même, que ce soit pour les données complètes ou les données partielles. Il s'agit en effet d'un attribut acoustique récupéré automatiquement à partir du logiciel Analor : la durée moyenne.

Branche gauche. En remplacement de l'attribut `intro`, on trouve cette fois le `slowing` courant, avec un seuil légèrement supérieur à celui qui était utilisé dans l'arbre de décision de la classification des frontières d'UI. Si on s'arrête au troisième niveau de profondeur, on ne distingue pas vraiment de différence entre les deux groupes divisés par ce critère. Dans les deux cas, l'attribut suivant est le POS du mot qui suit, avec comme débouchés directs les classes `false`, `///`, et `(`, avec simplement la classe `O` en plus dans le cas où le slowing est supérieur à 0.217.

Branche droite. Cette branche est similaire à celle de l'arbre précédent à un détail près. Puisqu'on ne dispose pas dans ces données de l'attribut `hes`, il a été remplacé par un attribut acoustique : la présence d'une frontière de période selon Analor dans la position suivante. Alors que la présence d'une disfluence ne menait directement qu'à une non-frontière ou à une frontière majeure, ce nouvel attribut permet de classifier également en frontières mineures (frontière droite de pré-noyau `<` ou frontière gauche de post-noyau `>` dans quelques cas.

Analyse globale

Les remarques faites à propos des arbres de décision de la classification en frontières majeures sont également valables pour celle des frontières mineures : les arbres sur données partielles sont similaires aux arbres sur données complètes, si on ne prend bien évidemment pas en compte des remplacements nécessaires d'attributs disponibles dans le premier cas mais pas le second, et, encore une fois, le classifieur semble utiliser autant les attributs acoustiques (en particulier `durée` et le `slowing`), que morpho-syntaxiques (`POS` et `intro`) et perceptifs (`hes`).

Autre remarque : un coup d'œil aux arbres textuels annexés (plus détaillés que ceux présentés dans cette partie) permet de voir que les arbres construits sur les données partielles sont dans les deux cas plus petits que ceux construits sur les données complètes (environ 600 nœuds supplémentaires pour classifier les frontières d'UI, et jusqu'à plus de 2000 nœuds pour les frontières de CI). Cette observation peut paraître tout à fait anodine étant donné que les classifieurs disposent de plus de données dans un cas que dans l'autre, mais cela signifie également que chaque classification nécessite un maximum d'information pour arriver à un résultat satisfaisant et donc que les attributs dont on dispose pour le moment ne sont pas suffisamment efficaces.

4.3 Analyse de la segmentation

Le sujet de ce mémoire est la segmentation d'un corpus oral en unités macrosyntaxiques, mais cette tâche a été redéfinie en une série de classification des espaces intermots, ce qui revenait à annoter chacune des frontières droites des mots du corpus pour des raisons pratiques. Cette annotation a ensuite été évaluée en détail dans la section 3.2.1.3 et suffit à se faire une idée de la qualité de la segmentation.

Toutefois, il est possible, à partir de l'annotation du corpus, de revenir à la segmentation en annotant cette fois le corpus non pas en terme de frontières, mais en terme d'appartenance à un groupe syntaxique. La segmentation peut ainsi être représentée sous la forme d'une couche d'étiquettes de type BI[5]. Cette représentation linéaire ne permet néanmoins pas de traiter des constituants récursifs ou discontinus : les segments syntaxiques que l'on utilise sont alors des chunks[6].

Deux solutions sont possibles pour traiter les segments discontinus avec les chunks :
- soit on sur-découpe en segmentant à chacune des frontières, même si les segments ne sont pas des unités complètes
- soit on sous-découpe en ne retenant que les frontières les plus larges, celles de l'unité la plus haute dans l'arbre, avant de chercher à retrouver les niveaux inférieurs dans un second temps.

5. B pour **B**egin et I pour **I**n, comme dans le format BILU défini précédemment. Nous n'utilisons pas les étiquettes L et U pour nous conformer au format d'entrée du programme d'évaluation, qui ne les reconnaît pas (voir partie 4.3.2).

6. Notion définie en 2.3.1 et déjà utilisée dans nos attributs.

[Rhap-D2013]
ˆet [rien n'arrête la vague rose //] >+ écrivent Les Dernières Nouvelles d'Alsace //

[Rhap-M2002]
[oui // l'art est le reflet de la société //+ ˆmais "euh" { pas tout le temps | ˆet pas "euh" & | ˆet pas sous tous les aspects } //]

FIGURE 4.3: Exemples de segments discontinus : enchâssement d'une greffe dans une UI, de plusieurs UI dans une greffe

Celle qui a été retenue est la première, puisque la seconde suppose que l'on pourrait naviguer, d'une manière ou d'une autre, à travers les profondeurs d'arbres syntaxiques. Or, si c'est le cas pour le corpus de référence, ce n'est pas vrai pour ce même corpus que nous avons annoté automatiquement. Il serait donc difficile d'évaluer la segmentation. De plus, cette solution nous permet d'examiner toutes les frontières à la fois, sans avoir besoin de le faire par étapes.

On aurait donc, pour l'exemple tiré de D2013 de la figure 4.3, les trois segments suivants :

et	rien	n'arrête	la	vague	rose
B-UI	B-GRF	I-GRF	I-GRF	I-GRF	I-GRF

écrivent	Les	Dernières	Nouvelles	d'Alsace
B-UI	I-UI	I-UI	I-UI	I-UI

4.3.1 De l'annotation à la segmentation

Passer de l'annotation en frontières à la segmentation en chunks nécessite plus qu'un simple changement d'étiquettes. L'annotation en frontière indique effectivement où se trouvent les frontières, où segmenter en unités syntaxiques, mais l'étiquette en elle-même ne porte qu'une seule information, alors que la frontière est une interface entre deux unités, nécessairement à la fois fin et début.

Pour étiqueter en chunks, on a besoin de deux informations : la position du mot dans le groupe syntaxique (B ou I, avec O pour la classe `O` qui type la frontière entre un noyau et un introducteur d'UI dans la classification en frontières de CI), et le type de groupe (détails dans la figure 4.4). À l'aide de scripts, on tâchera de transformer la sortie de la tâche de classification réalisée au préalable (figure 3.4) en annotation en chunks comme dans la figure 4.5. Le format d'entrée pour le programme d'évaluation est le même que précédemment : en première colonne on trouve les mots, puis l'étiquetage du corpus de référence (considéré correct), et enfin l'étiquetage tel qu'il est prédit par le classifieur (à évaluer).

UI	: unité illocutoire	KERNEL	: noyau
PAR	: UI parenthétique	(INT-)INKNL	: in-noyau (intégré ou différé)
GRF	: greffe	(INT-)PREKNL	: pré-noyau (intégré ou différé)
		(INT-)POSTKNL	: post-noyau (intégré ou différé)

FIGURE 4.4: Les chunks et leur classe équivalente

Le cas de l'epéxégèse (unités en position de complément différé) est nommé *différé* dans ce mémoire mais *intégré* dans nos données, d'où le préfixe INT-.

[Rhap-D0001,CFPP2000]
"je veux dire" là < ça me laisse rêveur //
Paul Valéry en Zep //

token	etiq_ref	etiq_weka
ça	B-KERNEL	B-KERNEL
me	I-KERNEL	I-KERNEL
laisse	I-KERNEL	I-KERNEL
rêveur	I-KERNEL	I-KERNEL
Paul	B-KERNEL	B-PREKNL
Valéry	I-KERNEL	I-PREKNL
en	I-KERNEL	I-PREKNL
Zep	I-KERNEL	I-PREKNL

FIGURE 4.5: Extrait du corpus transformé au format d'entrée pour le programme d'évaluation en chunks de CRoTAL

4.3.1.1 Typage des frontières majeures

Dans le cas du typage des UI, la transformation est relativement aisée :

sub chunk{} UI type

```
# Attribution des étiquettes complètes
(1) Si la classe précédente est vide...          -- B-UI    # cas des débuts de fichiers
(2)     Sinon, si la classe précédente est (...  -- B-INS
(3)     Sinon, si la classe précédente est [...  -- B-GRF

# Attribution des positions
(4)     Sinon, si la classe n'est pas false...   -- B-[numero_de_la_classe]
(5)     Sinon, dans tous les autres cas...       -- I-    # pas une frontière gauche
```

On repose donc principalement sur les frontières gauches des unités syntaxiques pour déterminer le type de chunks. Ensuite, il suffit de faire passer un module pour remplir les informations manquantes (les cas 4 et 5), en lisant le fichier ligne par ligne. On peut ainsi récupérer le type du groupe puis l'assigner à toutes les étiquettes qui suivent tant qu'on ne tombe pas sur un autre début de groupe (étiquette B-).

sub fill{}

```
Pour chaque ligne du fichier...
    Si l'étiquette du mot est complète (cas 1, 2 et 3), alors...  -- Récupérer le type
        Sinon, si l'étiquette du mot est composée d'un chiffre (cas 4), alors...
            -- Remplacer ce chiffre par UI;
            -- Récupérer le type UI;
        Sinon, si l'étiquette commence par B, alors...            -- Réinitialiser le type
        Sinon, si l'étiquette commence par I (cas 5), alors...    -- Assigner le type
```

4.3.1.2 Typage des frontières mineures

Les choses se compliquent un peu pour les frontières de CI. En effet, si les frontières des différents types de chunks pour les UI ne sont pas ambiguës grâce au fait que seul un des chunks (UI) n'a pas de frontière gauche, ce n'est pas le cas des frontières mineures. Sur les dix types de frontières possibles, trois sont des frontières droites : les pré-noyaux

(<), les pré-noyaux différés (<+) et les frontières majeures (///).

On a donc un algorithme un peu plus complexe mais basé sur le même principe que pour la transformation de l'annotation des frontières d'UI :

sub chunk{} CI type

```
# Attribution des étiquettes complètes
(1) Si la classe courante n'est pas un pré-noyau(différé)    -- B-KERNEL
(2)     Sinon, si la classe précédente est >...               -- B-POSTKNL
(3)     Sinon, si la classe précédente est >+...              -- B-INT-POSTKNL
(4)     Sinon, si la classe précédente est (...               -- B-INKNL
(5)     Sinon, si la classe précédente est (+...              -- B-INT-INKNL
(6)     Sinon, si la classe précédente est O...               -- B-OUT

# Attribution des positions
(7)     Sinon, si la classe précédente est vide...            -- B-   # débuts de fichiers
(8)     Sinon, si la classe n'est pas false...                -- B-
(9)     Sinon, dans tous les autres cas...                    -- I-   # pas une frontière gauche

# Attribution des types de chunks
(10) Si la classe courante est <...                       -- Ajouter PREKERNEL   # X < Y
(11)    Sinon, si la classe courante est <+...            -- Ajouter INT-PREKERNEL   # X X < Y
(12)    Sinon, si la classe courante est > ou >+...       -- Ajouter KERNEL
```

Remarques :
(1) Un pré-noyau ne peut précéder qu'un autre pré-noyau ou un noyau.
(6) On utilise un chunk OUT dans un premier temps pour délimiter les introducteurs. Dans un second temps, on retransforme ces étiquettes en O pour rester dans le codage BIO.
(12) Un post-noyau ne peut être précédé que d'un autre post-noyau ou d'un noyau.

Après cette première ébauche d'étiquettes, on passe, comme pour les UI, le module qui permet de compléter les données manquantes en lisant le fichier de haut en bas. Seulement cette fois, à cause des cas ambigus cités ci-dessus, on ne peut pas attribuer systématiquement à chaque frontière gauche non-typée jusque là (B-) le type "KERNEL" (comme on l'a fait avec "UI" précédemment) puisqu'il peut aussi s'agir de pré-noyaux (différés ou non). Cette information doit alors être récupérée à partir du typage des chunks (cas 10, 11 et 12), et répercutée de bas en haut.

sub reverseFill{}

```
Pour chaque ligne du fichier en partant du bas...
    Si l'étiquette du mot est complète et commence par I-
    (cas 9 + 10/11/12),  alors...                          -- Récupérer le type
        Sinon, si l'étiquette du mot est I-, alors...       -- Assigner le type
        Sinon, si l'étiquette du mot est B-, alors...
            -- Assigner,
            -- Réinitialiser le type
        Sinon, si l'étiquette est complète et commence par B-, alors...-- Réinitialiser le type
```

Le script complet qui permet la transformation de l'annotation en frontières en annotation en chunk se trouve en annexe (A.3).

4.3.2 Évaluation de la segmentation

Pour évaluer cette segmentation sous forme de couches d'étiquettes, on utilise une nouvelle fois le programme d'évaluation du projet CRoTAL de Denys Duchier, qui permet d'évaluer les chunks dans leur intégralité, en prenant en compte les deux frontières. On a également la possibilité d'évaluer chacune des étiquettes individuellement pour mieux comprendre lesquelles des étiquettes B- ou des étiquettes I- influe le plus sur le score du chunk.

Segmentation en unités majeures (UI)

Sur les données complètes

```
 GRF: P:  20.00% R:   2.06% F:   3.74%
 INS: P:  44.26% R:  11.59% F:  18.37%
  UI: P:  41.24% R:  23.17% F:  29.67%

B-GRF: P:  60.00% R:   6.19% F:  11.21%
I-GRF: P:  27.27% R:   8.64% F:  13.13%
B-INS: P:  78.69% R:  20.60% F:  32.65%
I-INS: P:  20.17% R:  12.79% F:  15.65%
 B-UI: P:  86.81% R:  48.78% F:  62.46%
 I-UI: P:  91.03% R:  98.12% F:  94.44%
```

Sur les données partielles

```
 GRF: P:  28.57% R:   4.12% F:   7.21%
 INS: P:  44.29% R:  13.30% F:  20.46%
  UI: P:  41.05% R:  22.29% F:  28.90%

B-GRF: P:  71.43% R:  10.31% F:  18.02%
B-INS: P:  78.57% R:  23.61% F:  36.30%
 B-UI: P:  86.95% R:  47.22% F:  61.20%
I-GRF: P:  33.33% R:  12.71% F:  18.40%
I-INS: P:  14.04% R:  15.99% F:  14.95%
 I-UI: P:  91.06% R:  97.26% F:  94.06%
```

Les scores sont beaucoup plus bas que pour l'évaluation de l'annotation des frontières (3.2.1.3) mais le classement reste le même : le meilleur score (précision, rappel et F-mesure confondus) revient aux UI, puis aux UI parenthétiques, et enfin aux greffes.
Si on regarde le détail, on constate que le score des étiquettes prises séparément est meilleur que celui du chunk GRF, ce qui signifie que le problème est surtout la cohésion entre les deux types d'étiquettes qui ne forment pas les chunks entiers attendus. Pour les UI parenthétiques, ce sont les étiquettes I- qui semblent tirer le score du chunk vers le bas, alors que pour les greffes, c'est exactement l'inverse.

Comme on l'a observé auparavant dans l'évaluation de l'annotation, les greffes et les UI parenthétiques (classes sous-représentées) ont un taux de précision et de rappel légèrement supérieur avec les données partielles, contrairement aux UI, mais les différences sont moindres.
On peut néanmoins souligner que le score global de la segmentation en chunks sur données partielles est meilleur qu'avec les données complètes, alors que ce n'était pas tout à fait le cas pour l'annotation à cause du biais entraîné par la classe `false`. Les pertes dans la classe majoritaire contrebalançaient alors complètement les gains des autres classes. Ici, les annotations `false` sont toutes différées dans chacun des chunks (étiquette I-).

Segmentation en unités mineures (CI)

Sur les données complètes

```
      KERNEL: P: 47.98% R: 30.58% F: 37.35%
       INKNL: P: 21.05% R: 2.40% F: 4.30%
   INT-INKNL: P: 28.57% R: 10.00% F: 14.81%
      PREKNL: P: 42.44% R: 14.48% F: 21.60%
  INT-PREKNL: P: 42.34% R: 15.76% F: 22.97%
     POSTKNL: P: 39.29% R:  6.43% F: 11.06%
 INT-POSTKNL: P: 63.64% R: 14.89% F: 24.14%

    B-KERNEL: P: 79.12% R: 50.70% F: 61.80%
    I-KERNEL: P: 81.51% R: 78.05% F: 79.74%
     B-INKNL: P: 63.16% R:  7.19% F: 12.90%
     I-INKNL: P: 15.00% R:  9.23% F: 11.43%
 B-INT-INKNL: P: 57.14% R: 20.00% F: 29.63%
 I-INT-INKNL: P: 48.15% R: 34.67% F: 40.31%
    B-PREKNL: P: 47.80% R: 16.00% F: 23.97%
    I-PREKNL: P: 31.95% R: 23.94% F: 27.37%
B-INT-PREKNL: P: 32.05% R: 14.79% F: 20.24%
I-INT-PREKNL: P: 33.13% R: 29.82% F: 31.39%
   B-POSTKNL: P: 67.86% R: 11.11% F: 19.10%
   I-POSTKNL: P: 33.76% R: 12.56% F: 18.31%
B-INT-POSTKNL: P: 63.64% R: 14.89% F: 24.14%
I-INT-POSTKNL: P: 65.22% R: 18.52% F: 28.85%
            O: P: 41.58% R: 81.26% F: 55.01%
```

Sur les données partielles

```
      KERNEL: P: 47.85% R: 29.52% F: 36.51%
       INKNL: P: 24.14% R: 4.19%  F: 7.14%
   INT-INKNL: P: N/A    R: N/A    F: N/A
      PREKNL: P: 51.64% R: 19.77% F: 28.60%
  INT-PREKNL: P: 41.72% R: 17.12% F: 24.28%
     POSTKNL: P: 48.48% R:  9.36% F: 15.69%
 INT-POSTKNL: P: 54.55% R: 12.77% F: 20.69%

    B-KERNEL: P: 78.94% R: 48.88% F: 60.38%
    I-KERNEL: P: 82.30% R: 76.68% F: 79.39%
     B-INKNL: P: 62.07% R: 10.78% F: 18.37%
     I-INKNL: P: 14.20% R: 11.79% F: 12.89%
 B-INT-INKNL: P: 80.00% R: 20.00% F: 32.00%
 I-INT-INKNL: P: 18.18% R: 16.00% F: 17.02%
    B-PREKNL: P: 60.00% R: 22.74% F: 32.98%
    I-PREKNL: P: 31.23% R: 22.63% F: 26.24%
B-INT-PREKNL: P: 34.15% R: 16.57% F: 22.31%
I-INT-PREKNL: P: 28.85% R: 24.35% F: 26.41%
   B-POSTKNL: P: 75.76% R: 14.62% F: 24.51%
   I-POSTKNL: P: 41.18% R: 18.25% F: 25.29%
B-INT-POSTKNL: P: 63.64% R: 14.89% F: 24.14%
I-INT-POSTKNL: P: 32.95% R: 11.93% F: 17.52%
            O: P: 39.76% R: 83.10% F: 53.78%
```

On observe une baisse des scores pour tous les chunks, à l'exception des post-noyaux différés (INT-POSTKNL), dont les scores sont exactement identiques à ceux de l'annotation. On en déduit qu'à partir de l'annotation des frontières gauches uniquement `>+`, il est possible de retrouver le chunk complet.

L'autre point remarquable est l'impossibilité d'évaluer les chunks in-noyaux différés. Malgré une F-mesure correcte comparée aux autres unités pour chacune des étiquettes qui les compose (0.32 pour B-INT-INKNL, et 0.17 pour I-INT-INKNL, contre une moyenne de 0.3 pour les étiquettes B- et 0.29 pour les I-), aucun des chunks INT-INKNL ne correspond parfaitement à ceux du corpus de référence.

La figure suivante permet d'illustrer le problème. Le début du chunk est correctement identifié mais alors que l'in-noyau différé s'arrête à Champagne dans le corpus de référence, le nôtre s'étend jusqu'à la frontière suivante. Il s'agit donc bien d'un problème de segmentation : l'absence d'une frontière droite d'in-noyau `)` après le mot Champagne invalide à la fois le chunk INT-INKNL et le chunk PREKNL qu'il écrase.

[Rhap-D2013]
[il faut s'appeler Rachida Dati] **(+ écrit aussi Libération Champagne) pour Le Parisien Aujourd'hui en France** <+ à la une < ce n'est plus une vague //=

token	etiq_ref	etiq_weka
écrit	B-INT-INKNL	B-INT-INKNL
aussi	I-INT-INKNL	I-INT-INKNL
Libération	I-INT-INKNL	I-INT-INKNL
Champagne	I-INT-INKNL	I-INT-INKNL
pour	B-INT-PREKNL	I-INT-INKNL
Le	I-INT-PREKNL	I-INT-INKNL
Parisien	I-INT-PREKNL	I-INT-INKNL
Aujourd'hui	I-INT-PREKNL	I-INT-INKNL
en	I-INT-PREKNL	I-INT-INKNL
France	I-INT-PREKNL	I-INT-INKNL

FIGURE 4.6: Illustration d'un chunk in-noyau différé mal segmenté

L'évaluation de la segmentation est en fait deux fois plus exigeante que celle de l'annotation simple parce qu'elle évalue non seulement si les (deux) frontières correspondent mais aussi si le type de chunk est correctement identifié.

Chapitre 5

Conclusion et perspectives

A partir d'un enregistrement audio et de la transcription correspondante, il est possible de segmenter automatiquement un texte oral en unités syntaxiques majeures, et ce, même en ne s'appuyant que sur des informations calculées par des programmes, avec un relatif équilibre entre les indices prosodiques et morpho-syntaxiques. Bien que nous ne soyons parvenus qu'à un résultat relativement peu satisfaisant, ce travail préliminaire a permis de répondre en partie aux questions que nous nous posions, tout en soulevant un certain nombre de difficultés et de pistes pour améliorer nos résultats.

Performance algorithmique Pour répondre à notre problématique, nous avons choisi de favoriser les algorithmes dont les modèles sont interprétables, mais la lisibilité peut se faire au détriment de la performance. Dans un second temps, il serait intéressant de tester les algorithmes les plus performants habituellement, notamment les classifieurs SVM [1] qui ont l'avantage de mettre au point des critères en confrontant chacune des classes les unes aux autres, au lieu de ne chercher qu'à opposer une classe à toutes les autres (cadre One-Versus-All), une méthode qui permet de mieux cerner les caractéritiques des différentes classes.

Disproportion des classes Face à ce problème, nous avons voulu réduire la classe majoritaire (celle des *non-frontières* que nous avons appelé la classe `false`) mais ce sous-échantillonnage n'a pas eu le résultat espéré puisqu'il a entraîné un sur-apprentissage important sur les autres classes et une perte sur la précision qui n'est pas compensée par le gain sur le score de rappel.

Un sur-échantillonnage des classes sous-représentées pourrait être une solution, mais elle comporterait les mêmes risques que le sous-échantillonnage. La disproportion des classes dans la classification est en fait un problème reconnu qui n'a pas encore de solution efficace à ce jour, mais de nombreux travaux récents se tournent non pas du côté du corpus d'apprentissage mais plutôt du classifieur. Une des pistes à explorer serait alors d'implémenter localement dans Weka un de ces algorithmes d'optimisation [Gay 2009, Yang *et al.* 2013] modifiés expressément pour ce type de corpus.

Classification par étapes Jusqu'ici, les expériences pour repérer les frontières et pour les typer ont été faites en parallèle. Le but était de se servir des résultats du repérage des frontières pour pouvoir faciliter le typage en récupérant les prédictions faites (classes `true` et `false`) puis d'en faire directement un attribut supplémentaire que le classifieur qui va chercher à typer pourra utiliser pour l'aider à discriminer les espaces qui représentent

1. Les séparateurs à vaste marge (ou **S**upport **V**ector **M**achine en anglais), dont l'algorithme implémenté dans Weka est SMO (pour **S**equential **M**inimal **O**ptimization), qui calculent un coefficient pour chaque attribut en prenant en compte à la fois leur pouvoir discriminant et leur couverture.

potentiellement une frontière syntaxique s'il juge que c'est un bon critère de classification. De cette manière, on se contenterait d'orienter le classifieur sans le contraindre (comme c'est le cas avec la pré-sélection par SEM), un avantage plus qu'appréciable quand on manipule des données non-corrigées. Cette manipulation a ensuite été abandonnée en raison de résultats peu concluants puisque l'on a obtenu, pour rappel, moins de 1% d'instances mieux classifiées dans le repérage (résultats de l'expérience initiale).

De la même manière, il est possible d'utiliser les résultats de la classification de frontières d'UI pour ensuite classifier les frontières de CI puisque celles-ci ne sont pas sans lien : les frontières peuvent coïncider (cas d'un noyau formant une UI), ou se trouver à l'intérieur des UI avec des contraintes implicites (par exemple, aucune frontière de post-noyau ne peut précéder une frontière de pré-noyau à l'intérieur d'une même UI). Néanmoins, puisque la classification supervisée se réalise comme une suite de opérations locales indépendantes les unes des autres, la prise en compte de ces contraintes correspond mieux à une tâche d'annotation basée sur les CRF [Liu *et al.* 2006] qui, comme on l'a précisé, ont l'avantage de tenir compte des autres étiquettes à l'intérieur d'une unité donnée [2] et donc d'annoter directement des séquences entières d'étiquettes. Cette méthode ne peut, par conséquent, pas être appliquée sur un corpus non segmenté, mais il suffirait alors de le prétraiter en segmentant d'abord en UI (ou tout autre unité maximale) grâce à la classification supervisée comme on l'a fait pour ce mémoire.

Redéfinition des classes Enfin, loin des considérations statistiques, nous pourrions également envisager de repenser les classes à retrouver en prenant notamment en compte le côté pratique et l'objectif de la mise en œuvre d'une telle chaîne d'automatisation. Quelles sont les classes les plus utiles et les plus importantes pour faciliter le travail d'un annotateur expert qui devra passer par la suite pour s'assurer de l'exactitude de la segmentation ? Les plus faciles à corriger dans le cas d'erreurs systématiques ?

Par ailleurs, nous nous sommes basés uniquement sur les frontières d'UI et de CI mais il serait intéressant d'élargir les classes à d'autres unités utilisées pour décrire la structure syntaxique de données orales, comme les entassements [3] décrits spécifiquement dans Kahane et Pietrandrea [2012] et étudiés sur le plan intono-syntaxique dans Belião [2012] qui a mis en évidence la dimension prosodique (en particulier au niveau du ralentissement) de ce phénomène syntaxique.

Cette étude conforte donc l'idée que les unités de la syntaxe et de la prosodie sont fortement liées et ce qui n'est là qu'un travail préliminaire ouvre la voie à d'autres expériences plus approfondies, à des travaux explorant davantage les possibilités que nous avons tâché d'éclairer.

2. Typiquement, pour les données issues de l'écrit, l'unité de base est la phrase. C'est notamment le cas pour les modèles de SEM.

3. Constructions caractérisées par le fait que plusieurs éléments mis en parallèle occupent une même position syntaxique

Bibliographie

Abney, S. (1991). *Parsing by Chunks*. Kluwer Academic Publishers, Dordrecht, r. berwick, s. abney and c. tenny (eds.) édition.

Avanzi, M. (2012). *L'interface prosodie/syntaxe en français parlé. Dislocations, incises et asyndètes*. Peter Lang.

Avanzi, M., Lacheret, A., Victorri, B. *et al.* (2008). Analor, un outil d'aide pour la modélisation de l'interface prosodie-grammaire. *Travaux linguistiques du CERLICO*, (21):27–46.

Avanzi, M., Obin, N., Lacheret, A. et Victorri, B. (2011). Toward a continuous modeling of french prosodic structure : Using acoustic features to predict prominence location and prominence degree. *In Proceedings of Interspeech*, pages 2033–2036.

Belião, J. (2012). Formalisation, implémentation et exploitation d'une hiérarchie objet intonosyntaxique : "étude intonosyntaxique du phénomène d'entassement". Mémoire de D.E.A., Université Sorbonne-Nouvelle.

Belião, J., Lacheret, A. et Kahane, S. (2013). Disfluencies and discursive markers : when prosody and syntax plan discourse. *DiSS 2013*. (soumis).

Benzitoun, C., Dister, A., Gerdes, K., Kahane, S., Pietrandrea, P., Sabio, F. et Debaisieux, J.-M. (2010). Tu veux couper là faut dire pourquoi. propositions pour une segmentation syntaxique du francais parlé. *Actes du Congrès Mondial de Linguistique Francaise*.

Boersma, P. (2002). Praat, a system for doing phonetics by computer. *Glot international*, 5(9/10):341–345.

Branca-Rosoff, S., Fleury, S., Lefeuvre, F. et Pires, M. (2000). Discours sur la ville. *Corpus de français parlé parisien des années*.

Chen, S. et Gopalakrishnan, P. (1998). Speaker, environment and channel change detection and clustering via the Bayesian Information Criterion. *In Proc. DARPA Broadcast News Transcription and Understanding Workshop*, page 8. Virginia, USA.

Constant, M., Tellier, I., Duchier, D., Dupont, Y., Sigogne, A., Billot, S. *et al.* (2011). Intégrer des connaissances linguistiques dans un crf : application à l'apprentissage d'un segmenteur-étiqueteur du français. *TALN2011*, 1.

Crabbé, B., Candito, M. *et al.* (2008). Expériences d'analyse syntaxique statistique du français. *In Traitement automatique des langues naturelles-TALN 2008*.

Eshkol-Taravella, I., Baude, O., Maurel, D., Hriba, L., Dugua, C. et Tellier, I. (2012). Un grand corpus oral «disponible». *Traitement Automatique des Langues*, 52.

Gay, D. (2009). *Calcul de motifs sous contraintes pour la classification supervisée*. Thèse de doctorat, Université de Nouvelle Calédonie.

Gendrot, C., Adda-Decker, M. et Schmid, C. (2012). Comparaison de parole journalistique et de parole spontanée : analyses de séquences entre pauses. *In Actes de la conférence conjointe JEP-TALN-RECITAL 2012, volume 1 : JEP*, pages 649–656, Grenoble, France. ATALA/AFCP.

Kahane, S. et Pietrandrea, P. (2012). Typologie des entassements en francais. *In Actes de la conférence Linx*.

Lacheret, A., Victorri, B. *et al.* (2002). La période intonative comme unité d'analyse pour l'étude du français parlé : modélisation prosodique et enjeux linguistiques. *Verbum*, 1(24):55–72.

Lacheret-Dujour, A. et Beaugendre, F. (1999). *La prosodie du français*. CNRS éditions.

Lacheret-Dujour, A., Kahane, S. et Pietrandrea, P. (2013). *Rhapsodie : a Prosodic and Syntactic Treebank for Spoken French*. John Benjamins Amsterdam. col. Studies in Corpus Linguistics.

Lacheret-Dujour, A., Kahane, S., Pietrandrea, P., Avanzi, M. et Victorri, B. (2011). Oui mais elle est où la coupure, là ? quand syntaxe et prosodie s' entraident ou se complètent. *Langue française*, (170):61–79.

Laks, B., Durand, J., Lyche, C. *et al.* (2009). Le projet PFC (Phonologie du Français Contemporain) : une source de données primaires structurées. *Phonologie, variation et accents du français Hermès*, pages 19–6.

Liu, Y., Shriberg, E., Stolcke, A., Hillard, D., Ostendorf, M. et Harper, M. (2006). Enriching speech recognition with automatic detection of sentence boundaries and disfluencies. *Audio, Speech, and Language Processing, IEEE Transactions on*, 14(5):1526–1540.

Marcus, M. P., Marcinkiewicz, M. A. et Santorini, B. (1993). Building a large annotated corpus of english : The penn treebank. *Computational linguistics*, 19(2):313–330.

Morel, M.-A. et Boileau, L. D. (1998). *Grammaire de l'intonation : L'exemple du français*. Editions Ophrys.

Obin, N. (2012). Contours globaux. http ://www.projet-rhapsodie.fr/plus/publications/doc_download/179-contours-globaux.html.

Ostendorf, M., Favre, B., Grishman, R., Hakkani-Tur, D., Harper, M., Hillard, D., Hirschberg, J., Ji, H., Kahn, J. G., Liu, Y. *et al.* (2008). Speech segmentation and spoken document processing. *Signal Processing Magazine, IEEE*, 25(3):59–69.

Selkirk, E. (2000). The interaction of constraints on prosodic phrasing. *In Prosody : Theory and experiment*, pages 231–261. Springer.

Shriberg, E., Stolcke, A., Hakkani-Tür, D. et Tür, G. (2000). Prosody-based automatic segmentation of speech into sentences and topics. *Speech communication*, 32(1):127–154.

Stolcke, A. et Shriberg, E. (1996). Automatic linguistic segmentation of conversational speech. *In Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, volume 2, pages 1005–1008. IEEE.

Tellier, I., Dupont, Y. et Courmet, A. (2012). Un segmenteur-étiqueteur et un chunker pour le français. *Actes de TALN'12, session démo.*

Tellier, I., Dupont, Y., Eshkol-Taravella, I. et Wang, I. (2013). Adapt a text-oriented chunker for oral data : how much manual effort is necessary ? (en soumission).

Torreira, F., Adda-Decker, M. et Ernestus, M. (2010). The Nijmegen corpus of casual french. *Speech Communication*, 52(3):201–212.

Yang, H., Fong, S., Wong, R. et Sun, G. (2013). Optimizing classification decision trees by using weighted naïve bayes predictors to reduce the imbalanced class problem in wireless sensor network. *International Journal of Distributed Sensor Networks*, 2013.

Annexe A

Code source des scripts

A.1 Vectorisation des données pour Weka

vectorisationWeka.pl

```
use strict;
use utf8;
use Switch;
use open IN => ":encoding(utf8)", OUT => ":utf8";
binmode STDOUT, ":utf8";
#-------------------------------------------------------------#
my $rep = $ARGV[0];
$rep =~ s/[\/]$//;

my $selection = $ARGV[1];
my $type = $ARGV[2];
my $level = $ARGV[3];
my $word = $ARGV[4];
my @tokenList;
#-------------------------------------------------------------#
# Ouverture du dossier racine des fichiers Rhapsodie

opendir(DIR, $rep) or die "can't open $rep: $!\n";
my @files = `ls $rep`;
my $extension;

if ($selection) { $extension .= "Select."; }
if ($word) { $extension .= "Token."; }
$extension .= $type.".".$level;
open(FILEOUT,">:encoding(utf8)",$rep."/rhapsodie.".$extension.".arff");

print FILEOUT "\@relation rhapsodie\n";

my @titres =  split("\t",`head -1 $rep/$files[0]`);
chomp($titres[$#titres]);

#-------------------------------------------------------------#

print "Récupération du nom des attributs (titres des colonnes du fichier)
    ...\n";

&defineAttributes(@titres);

if ($type eq "UI" and $level eq "type") { print FILEOUT "\@attribute class
    { //, (, //), [, //], false }\n"; }
elsif ($type eq "UI" and $level eq "boolean") { print FILEOUT "\@attribute
    class { true, false }\n"; }
elsif ($type eq "CI" and $level eq "type") { print FILEOUT "\@attribute
    class { <, <+, >, >+, (, (+, ), ///, 0, false }\n"; }
else { print FILEOUT "\@attribute class { true, false, 0 }\n"; }
```

```
print FILEOUT "\n\@data\n";

print "...done !\n";

#-------------------------------------------------------------#

print "Attribution des valeurs des données...\n";

&vecteurs(@files);

print "...done !\n";

#-------------------------------------------------------------#

sub defineAttributes {
  my $titleList = @_;
  for my $x (0 .. $#titres) {

    print $x." : $titres[$x]\n";
    next if ($x >= 1 and $x <= 23); # sauter ce qu'on cherche

    switch ($titres[$x]) {
      case /label/          { $word ? &vectoToken() : last; }
      case /(pos|chunk)/        { &multiPos($titres[$x], 3); }
      case /(montée|durée|hauteur|Slowing)/ { print FILEOUT "\@attribute ".
          $titres[$x]." real\n"; }
      else { print FILEOUT "\@attribute ".$titres[$x]." ".&chooseList(
          $titres[$x])."\n"; }
    }
  }
}

sub vectoToken {
   open(FILETK,"<:encoding(utf8)","data/token");
   while (my $ligne=<FILETK>) {
     chomp($ligne);
     push(@tokenList, $ligne);
     &multiPos($ligne, 0);
  }
}

sub multiPos {
  my $colname = shift(@_);
  my $i = shift(@_);
  for my $position (-$i .. $i) {
    print FILEOUT "\@attribute \"".$colname."($position)\" ".&chooseList(
        $colname)."\n";
  }
}

sub chooseList {
  my $colname = shift(@_);
  my $list;
  switch ($colname) {
    case /prom\(/   { $list = "{ _, 0, S, W, x }" ; }
    case /hes\(/      { $list = "{ _, 0, H, x }" ; }
    case /Shape/    { $list = "{ _, --, -+, +-, ++, 0, +++1, -++2, +++2,
        -++3, +++3, -h, +h, h-, h+, +h+1, h-+1, h++1, --h2, -h+2, +h+2, h
        -+2, h++2, -h+3, h-+3, h++3, hh, hh+1, -hh2, h-h2, hh+2, -hh3, hh
        +3, hhh1, hhh2, hhh3, hl, hl+1, hl+2, hl+3, hlh2, hm, hm+1, hm+2,
        hm+3, hmh1, hmh2, hmh3, -l, +l, l-, l+, --l1, +l+1, --l2, +l+2, l
        -+2, l++2, l++3, lh, -lh1, lh+1, -lh2, l-h2, lh+2, l-h3, lh+3, lhh2
        , lhh3, ll, -ll1, ll+1, -ll2, l-l2, ll+2, -ll3, ll+3, llh1, llh2,
        llh3, lll2, lll3, llm1, llm2, llm3, lm, -lm1, l-m1, -lm2, l-m2, lm
        +2, -lm3, l-m3, lm+3, lmh1, lmh2, lmh3, lmm2, lmm3, -m, +m, m-, m+,
         +m+1, m-+1, --m2, -m+2, +m+2, m-+2, m++2, m-+3, m++3, mh, m-h1, mh
        +1, -mh2, m-h2, mh+2, -mh3, m-h3, mh+3, mhh1, mhh2, mhh3, ml, ml+1,
```

```
          ml+2, ml+3, mlh1, mlh2, mlh3, mlm1, mlm2, mlm3, mm, m-m1, mm+1, -
          mm2, m-m2, mm+2, -mm3, mm+3, mmh1, mmh2, mmh3, mmm1, mmm2, mmm3 }"
          ; }
    case /LocalReg/   { $list = "{ 0, _, m, h, H, l, L }" ; }
    case /prom-Analor/  { $list = "{ 0, p, P }" ; }
    case /periode-Analor/ { $list = "{ _, -, !, \$, 0,  basdescendant,
        basmontant,  basplat,  hautdescendant,  hautmontant,  hautplat,
        normaldescendant,  normalmontant,  normalplat }" ; }
    case /locuteur\(/   { $list = "{ _, \$L1, \$L1-\$L2, \$L1-\$L3, \$L1-\
        $L4, \$L1-\$L5, \$L2, \$L2-\$L1, L2-\$L1, \$L2-\$L3, \$L3, \$L3-\
        $L1, \$L3-\$L2, \$L4, \$L4-\$L1, \$L4-\$L3, \$L5, 0 }" ; }
    case /^pos$/      { $list = "{ ADJ, _ADJ, ADJWH, ADV, _ADV, ADVWH, CC,
        _CC, CLO, CLR, CLS, _CLS, CS, _CS, DET, _DET, DETWH, ET, I, _I, NC,
         _NC, NPP, _NPP, P, _P, P+D, P+PRO, PREF, PRO, _PRO, PROREL, PROWH,
         V, _V, VIMP, VINF, _VINF, VPP, _VPP, VPR, _VPR, VS, 0 }" ; }
    case /chunk/    { $list = "{ B-AdP, B-AP, B-CONJ, B-NP, B-PP, B-
        __UNKNOWN__, B-VN, I-AdP, I-AP, I-CONJ, I-NP, I-PP, I-__UNKNOWN__,
        I-VN, O, O }" ; }
    else { $list = "real" ; }
  }
  return $list;
}

sub vecteurs {
  print "Constitution de la matrice de données...\n";
  foreach my $file (@files) {
    chomp($file);

    my @lines;
    my @elements;

    if ($file =~ /\.sem$/) {
      print "$file...\n";
      # Récupération de chaque ligne (sauf celle des titres des colonnes)
      open(FILEIN,"<:encoding(utf8)",$rep."/".$file);
      while (my $ligne=<FILEIN>) {
        next if $ligne =~ /label/;
        chomp($ligne);

        @elements = split('\t',$ligne);
        push(@lines, [@elements]);
      }
      close(FILEIN);
      &data(@lines);
    }
  }
}

sub data {

  my @attributs = @_;
  for my $i (0 .. $#attributs-1) { # pour chaque ligne d'attributs

  #-------------------------------------------------------------#
  #   Attribution des valeurs de la classe
  #-------------------------------------------------------------#

  my $class;

  # on se trouve á une frontière d'UI si:
  # (1) on est á la fin du fichier,
  # (2) on est suivi d'un U OU d'un B
  # (3) on est au niveau d'un L ou d'un U
    if ($type eq "UI") {
      if ($level eq "boolean") {
        if ($i == $#attributs-1 or $attributs[$i+1][1] =~ /(B|U)/ or
            $attributs[$i][1] =~ /(L|U)/) { $class = "true\n"; }
```

```
        else { $class = "false\n"; }
      }
      elsif ($level eq "type") {
        if ($attributs[$i+1][9] eq "B" or $attributs[$i+1][9] eq "U") {
            $class = "(\n"; }
        elsif ($attributs[$i][9] eq "L" or $attributs[$i][9] eq "U") {
            $class = "//)\n"; }
        elsif ($attributs[$i+1][10] eq "B" or $attributs[$i+1][10] eq "U")
            { $class = "[\n";}
        elsif ($attributs[$i][10] eq "L" or $attributs[$i][10] eq "U") {
            $class = "//]\n";}
        elsif ($i == $#attributs-1 or ($attributs[$i][1] eq "I" and
            $attributs[$i+1][1] eq "U" and $attributs[$i+2][1] eq "B") or
            $attributs[$i][1] eq "L" or $attributs[$i][1] eq "U") { $class
            = "//\n"; }
        else { $class = "false\n"; }
      }
    }
    # on se trouve á une frontière de CI si:
    # (1) on n'est pas suivi d'un introducteur d'UI ou d'un joncteur
    # (2) on est au niveau d'un L ou d'un U (fin) dans les colonnes (
      integrated)prekernel
    # (3) on est suivi d'un B ou d'un U dans les colonnes (integrated)
      inkernel ou si on est au niveau d'un L ou d'un U
    # (4) on est suivi d'un B ou d'un U (début) dans les colonnes (integrated
      )postkernel
    elsif ($type eq "CI") {
      if ($level eq "boolean") {
        if ($attributs[$i+1][13] =~ /(B|U)/ or $attributs[$i+1][14] =~ /(B|
            U)/) { $class = "0\n"; }
        elsif ($attributs[$i][3] =~ /(L|U)/ or $attributs[$i][4] =~ /(L|U)/
             or $attributs[$i+1][5] =~ /(B|U)/ or $attributs[$i+1][6] =~ /(
            B|U)/ or $attributs[$i][5] =~ /(L|U)/ or $attributs[$i][6] =~
            /(L|U)/ or $attributs[$i+1][7] =~ /(B|U)/ or $attributs[$i
            +1][8] =~ /(B|U)/ or $i == $#attributs-1 or  $attributs[$i
            +1][2] =~ /(B|U)/ or $attributs[$i][2] =~ /(L|U)/) { $class = "
            true\n"; }
        else { $class = "false\n"; }
      }
      elsif ($level eq "type") {
        if ($attributs[$i+1][13] =~ /(B|U)/ or $attributs[$i+1][14] =~ /(B|
            U)/) { $class = "0\n"; }
        elsif ($attributs[$i][3] =~ /(L|U)/) { $class = "<\n"; }
        elsif ($attributs[$i][4] =~ /(L|U)/) { $class = "<+\n"; }
        elsif ($attributs[$i+1][5] =~ /(B|U)/) { $class = "(\n"; }
        elsif ($attributs[$i+1][6] =~ /(B|U)/) { $class = "(+\n"; }
        elsif ($attributs[$i][5] =~ /(L|U)/ or $attributs[$i][6] =~ /(L|U)
            /) { $class = ")\n";}
        elsif ($attributs[$i+1][7] =~ /(B|U)/) { $class = ">\n"; }
        elsif ($attributs[$i+1][8] =~ /(B|U)/) { $class = ">+\n"; }
        elsif ($i == $#attributs-1 or $attributs[$i+1][2] =~ /(B|U|0)/ or
            $attributs[$i][2] =~ /(L|U)/) { $class = "///\n";}
        else { $class = "false\n"; }
      }
    }
    else { print "invalid type (3rd argument)\n"; }

    #-------------------------------------------------------------#
    #   Attribution des valeurs des attributs       #
    #-------------------------------------------------------------#

      my $attLine;

      my $selected = 1;
      if ($selection) { if ($attributs[$i+1][116] !~ /^(B|0)/ and $class =~ /
          false/) { $selected = 0; } }
```

```
196       next if $selected == 0;

          if ($word) {
            for my $token (@tokenList) {
              for my $l (-0 .. 0) {
201             if ($attributs[$i+$l][0] eq $token) { $attLine .= "1," ;}
                else { $attLine .= "0," ; }
              }
            }
          }
206
          for my $j (2 .. $#titres) { # pour chaque colonne d'attributs
            next if ($j >= 1 and $j <= 23); # sauter les classes que l'on cherche
                !

            # traitement particulier des POS et Chunks qui doivent étendre leur
                portée á [-3:3]
211         if ($titres[$j] =~ /^(pos|chunk)$/) {
              for my $k (-3 .. 3) {
                if ($i+$k <= -1 or $i+$k >= $#attributs or $attributs[$i][$j] eq
                    "N/A") { $attLine .= "0,";} # cas de valeurs vides...
                else {$attLine .= &clean($attributs[$i+$k][$j]).",";}
              }
216         }
            elsif ((!$attributs[$i][$j]) or  $attributs[$i][$j] eq "N/A") {
                $attLine .= "0,";}
            else {
              if ($titres[$j] =~ /Shape/) { $attributs[$i][$j] =~ tr/LH/-+/; }

221           $attLine .= &clean($attributs[$i][$j]).",";
            }
            $j++;
          }
          print FILEOUT $attLine.$class;
226       $i++;
        }

      }

231   sub clean {
        my $word = shift(@_);
        $word =~ s/''//g;
        $word =~ s/période n°.+? registre //g;
        $word =~ s/ geste //g;
236     $word =~ tr/\?\%!/!x?/; # on met un ? lá pour signaler les informations
            manquantes
        return $word;
      }
```

A.2 Transformation des données pour l'évaluation de l'annotation

formatEval.pl

```
my $all;
my $cpt = 0;
my $type = $ARGV[0];
my $level = $ARGV[1];
my $filename = $ARGV[2];

open(FILEIN,"<:encoding(utf8)",$ARGV[2]) or die "Can't open $ARGV[2] !\n";
open(FILETK,"<:encoding(utf8)",$ARGV[3]) or die "Can't open token file !\n"
    ;

$filename =~ /.+\/(.+?)\.res$/;
print "Analyse de la sortie $ARGV[2]...\n";

open(FILEOUT,">:encoding(utf8)","eval/".$1.".".$type.".".$level.".annot")
    or die "Can't create output file !\n";
open(FILETXT,">:encoding(utf8)", $1.".".$type.".".$level.".txt") or die "
    Can't create output file !\n";

my $fileToken = do { local $/; <FILETK> };
my @tokens = split('\n',$fileToken);

my $fileIn = do { local $/; <FILEIN> };
my @lines = split('\n', $fileIn);

my $x;

for my $item (@lines) {
  next if ($item !~ /[[:digit:]]/); # éviter les lignes qui ne nous inté
      ressent pas

  my $newItem = &clean($item);

  $all .= $tokens[$x]."\t".$newItem."\n";
  $x++;
}

print FILEOUT $all;
print FILETXT &textFormat($all);

close(FILEOUT);
close(FILETXT);

sub clean { # nettoyage pour récupérer un format tabulaire propre
  my $line = shift(@_);
  $line =~ s/[\s\d]+:(.+?)[\s\d]+:(.+?)[\s+]+.+?$/\1\t\2/;
  return $line;
}

sub textFormat { # mise en forme pour obtenir une sortie textuelle
  my $data = shift(@_);
  $data =~ s/(.+?)\t.+?\t(.+?)/\1\t\2/g;
  $data =~ s/false\n//g;
  $data =~ s/([^\/])\n/\1 /g;
  $data =~ s/\t/ /g;
  $data =~ s/\n/\n\n/g;
  return $data;
}
```

A.3 Transformation des données pour l'évaluation du chunking

formatCeval.pl

```
my $all;
my $cpt = 0;
my $type = $ARGV[0];
my $level = $ARGV[1];
my $filename = $ARGV[2];

open(FILEIN,"<:encoding(utf8)",$ARGV[2]) or die "Can't open $ARGV[2] !\n";
open(FILETK,"<:encoding(utf8)",$ARGV[3]) or die "Can't open token file !\n";

$filename =~ /.+\/(.+?)\.res$/;
print "Analyse de la sortie $ARGV[2]...\n";

open(FILEOUT,">:encoding(utf8)","eval/".$1.".".$type.".".$level.".chunk")
    or die "Can't create output file !\n";

my $fileToken = do { local $/; <FILETK> };
my @tokens = split('\n',$fileToken);

my $fileIn = do { local $/; <FILEIN> };
my @lines = split('\n', $fileIn);

for my $x (0 .. $#lines) {
  next if ($lines[$x] !~ /[[:digit:]]/); # éviter les lignes qui ne nous
      intéressent pas

  my $actualClass = &chunk(1,$lines[$x],$lines[$x-1],$lines[$x-2]);
  my $predictedClass = &chunk(2,$lines[$x],$lines[$x-1],$lines[$x-2]);

  $all .= $actualClass."\t".$predictedClass."\n";
}

my @col1 = &fill($all,0);
my @col2 = &fill($all,1);

if ($type eq "CI") {
  my $newData;
  for (my $j=0;$j<=$#col1;$j++) { $newData .= $col1[$j]."\t".$col2[$j]."\n"
      ; }
  @col1 = &reverseFill($newData,0);
  @col2 = &reverseFill($newData,1);
  }

for (my $i=0;$i<=$#col1;$i++) {
  print FILEOUT $tokens[$i]."\t".&clean($col1[$i])."\t".&clean($col2[$i])."
      \n";
}

close(FILEIN);
close(FILETK);
close(FILEOUT);

sub getClass { # récupération de la classe (n° uniquement)
  my $column = shift(@_);
  my $line = shift(@_);

  $column == 1 ? $line =~ /\s(\d+):.+?:/ : $line =~ /\d:.+?\s+(\d+):/;
  return $1;
}

sub chunk { # transformation de l'annotation en chunk
  my $chunk;
  my $nb = shift(@_);
```

```
  my $line = shift(@_);
  my $prevLine = shift(@_);
  my $prevLine2 = shift(@_);

63  if ($type eq "UI") {
    if ($level eq "boolean") {
      if (&getClass($nb, $prevLine) eq "" or &getClass($nb,$prevLine) == 1)
          { return "B-UI"; }
      else { return "I-UI"; }
    }
68  elsif ($level eq "type") {
      if (&getClass($nb,$prevLine) eq "") { return "B-UI"; }
      elsif (&getClass($nb,$prevLine) == 2) { return "B-INS"; }
      elsif (&getClass($nb,$prevLine) == 4) { return "B-GRF"; }
      elsif (&getClass($nb,$prevLine) != 6) { return "B-".&getClass($nb,
          $prevLine); }
73    else { return "I-"; }
    }
  }
  elsif ($type eq "CI") {

78    if ($level eq "boolean") {
      if (&getClass($nb, $prevLine) eq "" or &getClass($nb,$prevLine) == 1)
          { return "B-CI"; }
      elsif (&getClass($nb,$prevLine) == 3) { return "B-OUT"; }
      else { return "I-"; }
    }
83  elsif ($level eq "type") {
      my $res;
      if (&getClass($nb,$line) !~ /(^1$|2)/ and &getClass($nb,$prevLine) =~
          /(^1$|2)/) { return "B-KERNEL"; } # Y < X
      elsif (&getClass($nb,$prevLine) eq "3") { return "B-POSTKERNEL"; }
      elsif (&getClass($nb,$prevLine) eq "4") { return "B-INT-POSTKERNEL";
          }
88    elsif (&getClass($nb,$prevLine) eq "5") { return "B-INKERNEL"; }
      elsif (&getClass($nb,$prevLine) eq "6") { return "B-INT-INKERNEL"; }
      elsif (&getClass($nb,$prevLine) eq "9") { return "B-OUT"; }
      elsif (&getClass($nb,$prevLine) eq "") { return "B-"; }
      elsif (&getClass($nb,$prevLine) ne "10") { $res = "B-"; } # cas des U
93    else { $res = "I-"; }

      if (&getClass($nb,$line) eq "1") { $res .= "PREKERNEL"; } # | X < Y
      elsif (&getClass($nb,$line) eq "2") { $res .= "INT-PREKERNEL"; } # X
          X < Y
      return $res;
98  }
  }
}

sub fill {
103 my $data = shift(@_);
  my $nb = shift(@_);
  my $chunk = 0;

  my @col = &getColumn($nb, $data);
108 foreach my $label (@col) {
    if ($label =~ /B-([A-Z-]+?)$/) { $chunk = $1; }
    elsif ($label =~ /B-[1-9]$/) {
      $label =~ s/B-.$/B-UI/;
      $chunk = "UI";
113   }
    elsif ($label eq "B-") { if ($chunk) { $chunk = "";} }
    elsif ($label eq "I-") { if ($chunk) { $label .= $chunk;} }
  }

118 return @col;
}
```

```
sub reverseFill {
  my $data = shift(@_);
  my $nb = shift(@_);
  my $chunk = 0;

  my @col = &getColumn($nb, $data);

  for (my $x=$#col; $x>0; $x--) {
    if ($col[$x] =~ /I-([A-Z-]+?)$/) { $chunk = $1; }
    elsif ($col[$x] eq "I-") { if ($chunk) { $col[$x] .= $chunk;} }
    elsif ($col[$x] eq "B-") { if ($chunk) { $col[$x] .= $chunk; $chunk = "
        "; } }
    elsif ($col[$x] =~ /B-[A-Z-]+?$/) { $chunk = ""; }
  }
  return @col;
}

sub getColumn {
  my $nb = shift(@_);
  my $data = shift(@_);

  my @column = map { (split)[$nb] } split /\n/, $data;
  return @column;
}

sub clean {
  my $line = shift(@_);
  $line =~ s/(B|I)-OUT/O/g;
  $line =~ s/-$/-KERNEL/g;
  return $line;
}
```

Annexe B

Arbres de décision J48

B.1 Classification des UI

Arbre de décision sur données complètes

```
context_duree(1) <= 20.599
|       context_hes(0) = _: false (0.0)
|       context_hes(0) = 0
|       |       pos(1) = ADJ, _ADJ, ADJWH: // (2.0/1.0), ADV, ADVWH: false (6.0/1.0),
    CC, _CC: false (0.0), CLO: false (17.0/8.0), CLR: false (4.0/1.0), CLS, _CLS:
    false (1.0), CS, _CS: false (7.0/1.0), DET, _DET: // (2.0/1.0), DETWH: false
    (0.0), ET, I, _I: false (0.0), NC, _NC, NPP, _NPP, P: false (382.0/98.0), _P:
    false (15.0), P+D, P+PRO: false (2.0), PREF: false (0.0), PRO, _PRO: // (5.0/1.0)
    , PROREL: false (102.0/8.0), PROWH: false (0.0), V, _V, VIMP: false (0.0), VINF:
    false (22.0/4.0), _VINF: // (3.0/1.0), VPP, _VPP: false (3.0/1.0), VPR: false
    (1.0), _VPR: false (0.0), VS: false (5.0/1.0), 0: // (51.0/2.0)
|       context_hes(0) = H: false (767.0/77.0)
|       context_hes(0) = x: false (90.0/25.0)
context_duree(1) > 20.599
|       context_syllableSlowing(0) <= 0.151
|       |       pos(0) = ADJ, _ADJ: false (0.0), ADJWH: false (0.0), ADV: false
    (52.04/19.0), _ADV, ADVWH: false (1.0), CC: false (21.02/1.0), _CC: false (0.0),
    CLO: false (6.0), CLR: false (0.0), CLS: false (20.04), _CLS: false (1.0), CS:
    false (6.0), _CS, DET, _DET: false (1.0), DETWH: false (0.0), ET, I: false
    (79.0/38.0), _I: false (0.0), NC, _NC, NPP, _NPP, P: false (31.02/2.0), _P: false
     (12.0/1.0), P+D: false (1.0), P+PRO: false (0.0), PREF: false (0.0), PRO, _PRO:
    // (1.0), PROREL: false (7.0), PROWH: false (0.0), V, _V: // (2.0), VIMP: false
    (0.0), VINF, _VINF: false (4.0), VPP, _VPP: ( (2.0/1.0), VPR: false (0.0), _VPR:
    false (0.0), VS: false (0.0), 0: false (0.0)
|       context_syllableSlowing(0) > 0.151: false (28011.81/1300.0)
```

Arbre de décision sur données partielles

```
context_duree(1) <= 20.599
|   pos(1) = ADJ
|   |   pos(0) = ADJ, _ADJ: false (0.0), ADJWH: false (0.0), ADV, _ADV: false (2.0),
    ADVWH: false (0.0), CC: false (2.0), _CC: false (0.0), CLO: false (0.0), CLR:
    false (0.0), CLS: false (0.0), _CLS: false (0.0), CS: false (0.0), _CS: false
    (0.0), DET: false (7.0), _DET: false (0.0), DETWH: false (0.0), ET: // (1.0), I,
    _I: false (0.0), NC, _NC: false (0.0), NPP: false (1.0), _NPP: false (0.0), P:
    false (1.0), _P: false (0.0), P+D: false (0.0), P+PRO: false (0.0), PREF: false
    (0.0), PRO: false (0.0), _PRO: false (0.0), PROREL: false (0.0), PROWH: false
    (0.0), V: false (8.0), _V: false (0.0), VIMP: false (0.0), VINF: false (4.0),
    _VINF: false (0.0), VPP: false (0.0), _VPP: false (0.0), VPR: false (0.0), _VPR:
    false (0.0), VS: false (1.0), 0: false (0.0)
|   pos(1) = _ADJ
|   |   context_hauteur_moy(2) <= 87.885227: false (4.0)
|   |   context_hauteur_moy(2) > 87.885227: // (2.0)
|   pos(1) = ADJWH: // (2.0/1.0)
|   pos(1) = ADV
|   |   chunk(1) = B-AdP, B-AP: false (0.0), B-CONJ: false (0.0), B-NP: false (0.0), B
    -PP: false (0.0), B-_UNKNOWN_: false (0.0), B-VN, I-AdP: false (0.0), I-AP:
    false (0.0), I-CONJ: false (0.0), I-NP, I-PP, I-_UNKNOWN_: false (0.0), I-VN, O
    : false (0.0), 0: false (0.0)
|   pos(1) = _ADV
|   |   pos(0) = ADJ: false (0.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV: false
    (58.0/23.0), _ADV, ADVWH: false (0.0), CC: false (0.0), _CC: false (0.0), CLO:
```

```
    false (0.0), CLR: false (0.0), CLS: false (0.0), _CLS: false (0.0), CS: false
    (0.0), _CS: false (0.0), DET: false (0.0), _DET: false (0.0), DETWH: false (0.0),
     ET: false (0.0), I, _I: false (0.0), NC: false (0.0), _NC: false (0.0), NPP:
    false (0.0), _NPP: false (0.0), P: false (0.0), _P: false (0.0), P+D: false (0.0)
    , P+PRO: false (0.0), PREF: false (0.0), PRO: false (0.0), _PRO: false (0.0),
    PROREL: false (0.0), PROWH: false (0.0), V: false (0.0), _V: false (0.0), VIMP:
    false (0.0), VINF: false (0.0), _VINF: false (0.0), VPP: false (0.0), _VPP: false
     (0.0), VPR: false (0.0), _VPR: false (0.0), VS: false (0.0), 0: false (0.0)
|   pos(1) = ADVWH: false (9.0/2.0)
|   pos(1) = CC
|   |   pos(2) = ADJ, _ADJ: // (0.0), ADJWH: // (0.0), ADV: // (74.0/19.0), _ADV: //
    (0.0), ADVWH: // (1.0), CC, _CC, CLO: // (4.0), CLR: false (1.0), CLS, _CLS: //
    (0.0), CS, _CS: // (0.0), DET, _DET: // (0.0), DETWH: // (0.0), ET: // (1.0), I,
    _I: // (0.0), NC, _NC: // (0.0), NPP: false (3.0/1.0), _NPP: // (0.0), P, _P: //
    (0.0), P+D, P+PRO: // (0.0), PREF: // (0.0), PRO, _PRO: // (0.0), PROREL: false
    (14.0/1.0), PROWH: // (0.0), V, _V: // (0.0), VIMP: // (0.0), VINF: false
    (3.0/1.0), _VINF: // (0.0), VPP: // (0.0), _VPP: // (0.0), VPR: // (0.0), _VPR:
    // (0.0), VS: // (0.0), 0: // (0.0)
|   pos(1) = _CC: false (0.0)
|   pos(1) = CLO: false (24.0/8.0)
|   pos(1) = CLR: false (5.0/1.0)
|   pos(1) = CLS
|   |   pos(0) = ADJ, _ADJ: // (3.0/1.0), ADJWH: // (0.0), ADV, _ADV, ADVWH: // (0.0),
     CC: false (7.0), _CC: // (0.0), CLO: false (1.0), CLR: false (1.0), CLS: //
    (0.0), _CLS: // (0.0), CS: false (12.0/1.0), _CS: false (6.0/1.0), DET: // (0.0),
     _DET: // (2.0/1.0), DETWH: // (0.0), ET: false (1.0), I, _I: // (0.0), NC, _NC:
    // (2.0/1.0), NPP: // (6.0/1.0), _NPP: // (1.0), P: [ (2.0/1.0), _P: // (0.0), P+
    D: false (1.0), P+PRO: false (1.0), PREF: // (0.0), PRO, _PRO: // (0.0), PROREL:
    false (5.0/1.0), PROWH: // (0.0), V, _V: // (2.0/1.0), VIMP: // (0.0), VINF,
    _VINF: // (2.0), VPP: // (9.0/1.0), _VPP: // (0.0), VPR: // (0.0), _VPR: // (0.0)
    , VS: // (1.0), 0: // (0.0)
|   pos(1) = _CLS: false (1.0)
|   pos(1) = CS
|   |   context_locuteur(1) = _: false (0.0), $L1, $L1-$L2, $L1-$L3: // (2.0/1.0), $L1
    -$L4: false (0.0), $L1-$L5: false (0.0), $L2, $L2-$L1: false (3.0/1.0), L2-$L1:
    false (0.0), $L2-$L3: // (2.0/1.0), $L3: // (4.0), $L3-$L1: false (1.0), $L3-$L2:
     false (0.0), $L4: false (0.0), $L4-$L1: false (0.0), $L4-$L3: false (0.0), $L5:
    false (0.0), 0: false (0.0)
|   pos(1) = _CS: false (20.0/1.0)
|   pos(1) = DET
|   |   pos(2) = ADJ, _ADJ: false (0.0), ADJWH: false (0.0), ADV: false (4.0), _ADV:
    false (0.0), ADVWH: false (0.0), CC: false (0.0), _CC: false (0.0), CLO: false
    (0.0), CLR: false (0.0), CLS: false (0.0), _CLS: false (0.0), CS: false (0.0),
    _CS: false (0.0), DET: false (3.0/1.0), _DET: false (3.0), DETWH: false (0.0), ET
    : false (2.0), I, _I: false (0.0), NC: false (298.0/75.0), _NC: false (0.0), NPP,
     _NPP: false (0.0), P: false (1.0), _P: false (0.0), P+D: // (1.0), P+PRO: false
    (0.0), PREF: false (0.0), PRO: false (1.0), _PRO: false (0.0), PROREL: false
    (0.0), PROWH: false (0.0), V: false (0.0), _V: false (0.0), VIMP: false (0.0),
    VINF: false (0.0), _VINF: false (0.0), VPP: false (0.0), _VPP: false (0.0), VPR:
    false (0.0), _VPR: false (0.0), VS: false (0.0), 0: false (0.0)
|   pos(1) = _DET: false (6.0/1.0)
|   pos(1) = DETWH: false (0.0)
|   pos(1) = ET
|   |   pos(3) = ADJ: // (2.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV: // (1.0),
    _ADV: false (0.0), ADVWH: false (0.0), CC: false (1.0), _CC: false (0.0), CLO:
    false (0.0), CLR: false (0.0), CLS: // (2.0/1.0), _CLS: false (0.0), CS: false
    (1.0), _CS: false (1.0), DET: // (1.0), _DET: false (0.0), DETWH: false (0.0), ET
    , I: false (9.0/2.0), _I: false (0.0), NC: false (0.0), _NC: false (0.0), NPP:
    false (0.0), _NPP: false (0.0), P: // (1.0), _P: false (0.0), P+D: false (1.0), P
    +PRO: false (0.0), PREF: false (0.0), PRO: false (0.0), _PRO: false (0.0), PROREL
    : false (0.0), PROWH: false (0.0), V: false (0.0), _V: false (0.0), VIMP: false
    (0.0), VINF: false (0.0), _VINF: false (0.0), VPP: false (0.0), _VPP: false (0.0)
    , VPR: false (0.0), _VPR: false (0.0), VS: false (0.0), 0: false (0.0)
|   pos(1) = I
|   |   context_periode-Analor(1) = _: false (0.0), -, !: false (119.0/23.0), $, 0,
    basdescendant: false (7.0/1.0), basmontant, basplat: // (1.0), hautdescendant:
    false (4.0), hautmontant: ( (2.0/1.0), hautplat: false (0.0), normaldescendant,
    normalmontant, normalplat: false (13.0/2.0)
|   pos(1) = _I: false (0.0)
|   pos(1) = NC
|   |   pos(0) = ADJ, _ADJ: false (0.0), ADJWH: false (0.0), ADV: false (2.0), _ADV:
    false (0.0), ADVWH: false (0.0), CC: false (7.0), _CC: false (0.0), CLO: false
    (0.0), CLR: false (0.0), CLS: false (0.0), _CLS: false (0.0), CS: false (0.0),
    _CS: false (0.0), DET: false (107.0/19.0), _DET: false (2.0), DETWH: false (0.0),
     ET: false (0.0), I, _I: false (0.0), NC: false (4.0), _NC: false (0.0), NPP:
    false (0.0), _NPP: false (0.0), P: false (21.0/1.0), _P: false (3.0), P+D: false
    (4.0), P+PRO: false (0.0), PREF: false (0.0), PRO: false (0.0), _PRO: false (0.0)
    , PROREL: false (0.0), PROWH: false (0.0), V: false (0.0), _V: false (0.0), VIMP:
     false (0.0), VINF: false (1.0), _VINF: false (0.0), VPP: false (0.0), _VPP:
```

```
     false (0.0), VPR: false (0.0), _VPR: false (0.0), VS: false (0.0), 0: false (0.0)
|    pos(1) = _NC
|    |    context_locuteur(3) = _: // (0.0), $L1, $L1-$L2, $L1-$L3: // (1.0), $L1-$L4:
     // (0.0), $L1-$L5: // (0.0), $L2: // (20.0/7.0), $L2-$L1: false (3.0), L2-$L1: //
      (0.0), $L2-$L3: // (0.0), $L3, $L3-$L1: // (0.0), $L3-$L2: // (0.0), $L4: //
     (0.0), $L4-$L1: // (0.0), $L4-$L3: // (0.0), $L5: // (0.0), 0: false (1.0)
|    pos(1) = NPP
|    |    context_syllableSlowing(-3) <= 0.963: false (7.0/1.0)
|    |    context_syllableSlowing(-3) > 0.963: // (15.0/4.0)
|    pos(1) = _NPP
|    |    context_syllableSlowing(0) <= 2.163: // (31.0/14.0)
|    |    context_syllableSlowing(0) > 2.163: false (23.0/5.0)
|    pos(1) = P
|    |    context_periode-Analor(1) = _: false (0.0), -, !: false (162.0/33.0), $, 0:
     false (2.0), basdescendant: false (3.0), basmontant: false (5.0), basplat: false
     (4.0), hautdescendant: false (3.0), hautmontant: false (4.0), hautplat: false
     (0.0), normaldescendant: false (22.0/2.0), normalmontant: false (36.0/4.0),
     normalplat: false (8.0)
|    pos(1) = _P: false (47.0)
|    pos(1) = P+D
|    |    context_syllableSlowing(1) <= 3.774: false (29.0)
|    |    context_syllableSlowing(1) > 3.774
|    pos(1) = P+PRO: false (2.0)
|    pos(1) = PREF: false (0.0)
|    pos(1) = PRO
|    |    pos(0) = ADJ: // (3.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV: false
     (12.0/5.0), _ADV: // (4.0/1.0), ADVWH: false (0.0), CC: ( (1.0), _CC: false (0.0)
     , CLO: false (0.0), CLR: false (0.0), CLS: false (0.0), _CLS: false (0.0), CS:
     false (2.0), _CS: false (1.0), DET: // (5.0/1.0), _DET: false (0.0), DETWH: false
      (0.0), ET: false (0.0), I, _I: false (0.0), NC, _NC: false (0.0), NPP: false
     (0.0), _NPP: false (1.0), P: false (3.0), _P: // (1.0), P+D: false (0.0), P+PRO:
     false (0.0), PREF: false (0.0), PRO, _PRO: false (0.0), PROREL: false (1.0),
     PROWH: false (0.0), V: false (12.0/3.0), _V: false (0.0), VIMP: false (0.0), VINF
     : false (0.0), _VINF: false (0.0), VPP: // (2.0/1.0), _VPP: false (0.0), VPR:
     false (0.0), _VPR: false (0.0), VS: false (0.0), 0: false (0.0)
|    pos(1) = _PRO
|    |    context_syllableSlowing(3) <= 2.178: // (4.0)
|    |    context_syllableSlowing(3) > 2.178: false (2.0)
|    pos(1) = PROREL: false (128.0/10.0)
|    pos(1) = PROWH: false (0.0)
|    pos(1) = V
|    |    pos(0) = ADJ, _ADJ: false (1.0), ADJWH: false (0.0), ADV, _ADV: // (1.0),
     ADVWH: // (1.0), CC: false (5.0), _CC: false (0.0), CLO: false (4.0), CLR: false
     (1.0), CLS: false (44.0/1.0), _CLS: false (0.0), CS: false (1.0), _CS: false
     (0.0), DET: false (0.0), _DET: false (0.0), DETWH: false (0.0), ET: false (0.0),
     I, _I: false (0.0), NC, _NC: // (1.0), NPP, _NPP: false (0.0), P: false (0.0), _P
     : false (0.0), P+D: false (0.0), P+PRO: false (0.0), PREF: false (0.0), PRO:
     false (13.0/4.0), _PRO: false (0.0), PROREL: false (13.0), PROWH: false (0.0), V:
      false (3.0), _V: false (0.0), VIMP: false (0.0), VINF: false (0.0), _VINF: false
      (0.0), VPP: [ (2.0/1.0), _VPP: false (0.0), VPR: false (0.0), _VPR: false (0.0),
      VS: false (0.0), 0: false (0.0)
|    pos(1) = _V
|    |    context_duree_moy(0) <= 145: // (2.0)
|    |    context_duree_moy(0) > 145: false (6.0)
|    pos(1) = VIMP: false (0.0)
|    pos(1) = VINF: false (38.0/4.0)
|    pos(1) = _VINF
|    |    context_montee(0) <= 0.629609: false (3.0)
|    |    context_montee(0) > 0.629609: // (3.0)
|    pos(1) = VPP
|    |    context_duree_moy(0) <= 201.359
|    |    context_duree_moy(0) > 201.359: false (26.0/3.0)
|    pos(1) = _VPP: false (3.0/1.0)
|    pos(1) = VPR: false (1.0)
|    pos(1) = _VPR: false (1.0)
|    pos(1) = VS: false (5.0/1.0)
|    pos(1) = 0: // (55.0/2.0)
context_duree(1) > 20.599
|    context_syllableSlowing(0) <= 0.151
|    |    pos(0) = ADJ, _ADJ: false (0.0), ADJWH: false (0.0), ADV: false (52.04/19.0),
     _ADV, ADVWH: false (1.0), CC: false (21.02/1.0), _CC: false (0.0), CLO: false
     (6.0), CLR: false (0.0), CLS: false (20.04), _CLS: false (1.0), CS: false (6.0),
     _CS, DET, _DET: false (1.0), DETWH: false (0.0), ET, I, _I: false (0.0), NC, _NC,
      NPP, _NPP, P: false (31.02/2.0), _P: false (12.0/1.0), P+D: false (1.0), P+PRO:
     false (0.0), PREF: false (0.0), PRO, _PRO: // (1.0), PROREL: false (7.0), PROWH:
     false (0.0), V, _V: // (2.0), VIMP: false (0.0), VINF, _VINF: false (4.0), VPP,
     _VPP: ( (2.0/1.0), VPR: false (0.0), _VPR: false (0.0), VS: false (0.0), 0: false
      (0.0)
|    context_syllableSlowing(0) > 0.151: false (28011.81/1300.0)
```

B.2 Classification des CI

Arbre de décision sur données complètes

```
context_duree_moy(1) <= 0
|   pos(1) = ADJ
|   |   context_syllableShape(0) = _: /// (1.0), --, -+: /// (1.0), +-: false (0.0),
    ++: false (1.0), 0: false (0.0), +++1: false (0.0), -++2: false (0.0), +++2:
    false (0.0), -++3: false (0.0), +++3: false (0.0), -h: /// (1.0), +h: false (1.0)
    , h-: /// (1.0), h+, +h+1: false (0.0), h-+1: false (0.0), h++1: false (0.0), --
    h2: false (0.0), -h+2: false (0.0), +h+2: false (0.0), h-+2: false (0.0), h++2:
    /// (1.0), -h+3: false (0.0), h-+3: false (0.0), h++3: false (0.0), hh, hh+1:
    false (1.0), -hh2: false (0.0), h-h2: false (0.0), hh+2: /// (2.0/1.0), -hh3:
    false (0.0), hh+3: false (0.0), hhh1: false (0.0), hhh2: false (0.0), hhh3: false
     (0.0), hl, hl+1: false (0.0), hl+2: false (0.0), hl+3: false (0.0), hlh2: false
    (1.0), hm: /// (12.0/4.0), hm+1: false (0.0), hm+2: false (0.0), hm+3: false
    (0.0), hmh1: false (0.0), hmh2: false (0.0), hmh3: false (0.0), -l: /// (2.0), +l
    : false (0.0), l-: /// (5.0/1.0), l+, --l1: false (0.0), +l+1: false (0.0), --l2:
     false (0.0), +l+2: false (0.0), l-+2: false (0.0), l++2: ) (1.0), l++3: ///
    (1.0), lh, -lh1: false (0.0), lh+1: false (0.0), -lh2: false (0.0), l-h2: false
    (1.0), lh+2: false (0.0), l-h3: false (0.0), lh+3: false (1.0), lhh2: false (0.0)
    , lhh3: false (0.0), ll, -ll1: false (0.0), ll+1: false (0.0), -ll2: false (0.0),
     l-l2: false (0.0), ll+2: false (0.0), -ll3: false (0.0), ll+3: false (0.0), llh1
    : false (0.0), llh2: false (0.0), llh3: false (0.0), lll2: false (0.0), lll3:
    false (0.0), llm1: false (1.0), llm2: false (0.0), llm3: false (1.0), lm: false
    (6.0/1.0), -lm1: false (0.0), l-m1: false (0.0), -lm2: false (0.0), l-m2: false
    (0.0), lm+2: false (0.0), -lm3: false (0.0), l-m3: false (0.0), lm+3: false (0.0)
    , lmh1: false (0.0), lmh2: false (0.0), lmh3: > (1.0), lmm2: false (0.0), lmm3:
    false (0.0), -m: false (0.0), +m: false (0.0), m-: /// (5.0/1.0), m+, +m+1: false
     (0.0), m-+1: false (0.0), --m2: false (0.0), -m+2: false (0.0), +m+2: false
    (1.0), m-+2: false (0.0), m++2: false (0.0), m-+3: false (1.0), m++3: <+ (1.0),
    mh, m-h1: false (0.0), mh+1: false (0.0), -mh2: false (0.0), m-h2: false (0.0),
    mh+2: false (0.0), -mh3: false (0.0), m-h3: false (0.0), mh+3: false (0.0), mhh1:
     false (0.0), mhh2: false (0.0), mhh3: false (1.0), ml, ml+1: false (0.0), ml+2:
    false (0.0), ml+3: false (0.0), mlh1: false (0.0), mlh2: false (0.0), mlh3: false
     (0.0), mlm1: false (0.0), mlm2: false (0.0), mlm3: false (0.0), mm, m-m1: false
    (1.0), mm+1: false (0.0), -mm2: false (0.0), m-m2: false (0.0), mm+2: false (1.0)
    , -mm3: false (0.0), mm+3: false (0.0), mmh1: false (1.0), mmh2: false (0.0),
    mmh3: false (0.0), mmm1: false (0.0), mmm2: false (0.0), mmm3: false (0.0)
|   pos(1) = _ADJ
|   |   context_hauteur_moy(2) <= 87.885227: false (4.0)
|   |   context_hauteur_moy(2) > 87.885227: /// (2.0)
|   pos(1) = ADJWH: /// (2.0)
|   pos(1) = ADV
|   |   context_hes(0) = _: /// (0.0)
|   |   context_hes(0) = 0
|   |   context_hes(0) = H: false (67.0/16.0)
|   |   context_hes(0) = x
|   pos(1) = _ADV
|   |   context_hes(0) = _: false (0.0)
|   |   context_hes(0) = 0
|   |   context_hes(0) = H: false (29.0/5.0)
|   |   context_hes(0) = x: false (10.0/3.0)
|   pos(1) = ADVWH
|   |   context_hauteur_moy(3) <= 84.501298: false (4.0)
|   |   context_hauteur_moy(3) > 84.501298
|   pos(1) = CC: O (459.0/55.0)
|   pos(1) = _CC: false (0.0)
|   pos(1) = CLO: false (23.0/9.0)
|   pos(1) = CLR: false (5.0/1.0)
|   pos(1) = CLS
|   |   context_hes(0) = _: /// (0.0)
|   |   context_hes(0) = 0: /// (303.0/93.0)
|   |   context_hes(0) = H
|   |   context_hes(0) = x
|   pos(1) = _CLS: false (1.0)
|   pos(1) = CS
|   |   context_syllableShape(0) = _: false (0.0), --, -+: false (0.0), +-: false
    (0.0), ++: false (2.0), 0: false (0.0), +++1: false (0.0), -++2: false (0.0),
    +++2: false (0.0), -++3: false (0.0), +++3: false (0.0), -h: < (1.0), +h: false
    (0.0), h-: O (2.0/1.0), h+: O (2.0), +h+1: false (0.0), h-+1: false (0.0), h++1:
    false (0.0), --h2: false (0.0), -h+2: false (0.0), +h+2: false (0.0), h-+2: false
     (0.0), h++2: false (1.0), -h+3: false (0.0), h-+3: false (0.0), h++3: false
    (0.0), hh: O (4.0/1.0), hh+1: false (0.0), -hh2: false (0.0), h-h2: false (0.0),
    hh+2: /// (1.0), -hh3: false (0.0), hh+3: false (0.0), hhh1: false (0.0), hhh2:
    false (0.0), hhh3: false (0.0), hl: false (4.0), hl+1: O (1.0), hl+2: false (0.0)
    , hl+3: false (0.0), hlh2: false (0.0), hm: false (4.0/1.0), hm+1: false (0.0),
    hm+2: false (0.0), hm+3: false (0.0), hmh1: false (0.0), hmh2: false (0.0), hmh3:
```

```
      false (0.0), −l: false (0.0), +l: false (1.0), l−: /// (7.0/1.0), l+: false
     (1.0), −−l1: false (0.0), +l+1: false (0.0), −−l2: false (0.0), +l+2: false (0.0)
     , l−+2: false (0.0), l++2: false (0.0), l++3: false (0.0), lh, −lh1: false (0.0),
      lh+1: false (0.0), −lh2: false (0.0), l−h2: O (1.0), lh+2: false (0.0), l−h3:
     false (0.0), lh+3: false (0.0), lhh2: false (0.0), lhh3: false (0.0), ll, −ll1:
     false (0.0), ll+1: false (0.0), −ll2: false (0.0), l−l2: false (0.0), ll+2: false
      (0.0), −ll3: false (0.0), ll+3: false (0.0), llh1: false (0.0), llh2: false
     (0.0), llh3: false (0.0), lll2: false (0.0), lll3: false (0.0), llm1: false (0.0)
     , llm2: false (0.0), llm3: false (0.0), lm: false (3.0/1.0), −lm1: false (0.0), l
     −m1: false (0.0), −lm2: false (1.0), l−m2: false (0.0), lm+2: false (0.0), −lm3:
     false (0.0), l−m3: false (0.0), lm+3: false (0.0), lmh1: false (0.0), lmh2: false
      (0.0), lmh3: false (0.0), lmm2: false (0.0), lmm3: false (0.0), −m: false (0.0),
      +m: false (0.0), m−: /// (3.0/1.0), m+, +m+1: false (0.0), m−+1: false (0.0), −−
     m2: false (0.0), −m+2: false (0.0), +m+2: false (0.0), m−+2: false (0.0), m++2:
     false (0.0), m−+3: false (0.0), m++3: false (1.0), mh, m−h1: false (0.0), mh+1:
     false (0.0), −mh2: false (0.0), m−h2: false (0.0), mh+2: false (0.0), −mh3: false
      (0.0), m−h3: false (0.0), mh+3: false (0.0), mhh1: false (0.0), mhh2: ) (1.0),
     mhh3: false (0.0), ml: false (10.0/4.0), ml+1: false (0.0), ml+2: false (0.0), ml
     +3: false (0.0), mlh1: false (0.0), mlh2: O (1.0), mlh3: false (0.0), mlm1: false
      (0.0), mlm2: false (0.0), mlm3: false (0.0), mm: false (19.0/6.0), m−m1: false
     (0.0), mm+1: false (0.0), −mm2: false (0.0), m−m2: false (0.0), mm+2: false (0.0)
     , −mm3: false (0.0), mm+3: false (1.0), mmh1: false (0.0), mmh2: false (0.0),
     mmh3: false (0.0), mmm1: false (0.0), mmm2: false (0.0), mmm3: false (0.0)
|    pos(1) = _CS
|    |    context_montee(−2) <= 3.965103: false (18.0)
|    |    context_montee(−2) > 3.965103: /// (2.0/1.0)
|    pos(1) = DET
|    |    context_hes(0) = _: false (0.0)
|    |    context_hes(0) = 0
|    |    context_hes(0) = H: false (70.0/6.0)
|    |    context_hes(0) = x
|    pos(1) = _DET: false (6.0/1.0)
|    pos(1) = DETWH: false (0.0)
|    pos(1) = ET
|    |    context_locuteur(−1) = _: /// (0.0), $L1, $L1−$L2: false (3.0), $L1−$L3: false
      (1.0), $L1−$L4, $L1−$L5: /// (0.0), $L2: false (3.0), $L2−$L1: /// (3.0/1.0), L2
     −$L1: /// (0.0), $L2−$L3: /// (0.0), $L3: /// (4.0/2.0), $L3−$L1: false (6.0/2.0)
     , $L3−$L2: /// (0.0), $L4: /// (0.0), $L4−$L1: /// (0.0), $L4−$L3: /// (0.0), $L5
     : /// (0.0), 0: /// (0.0)
|    pos(1) = I
|    |    context_hes(2) = _: false (1.0)
|    |    context_hes(2) = 0
|    |    context_hes(2) = H
|    |    context_hes(2) = x: /// (2.0)
|    pos(1) = _I: false (0.0)
|    pos(1) = NC
|    |    context_locuteur(1) = _: false (1.0), $L1: false (107.0/39.0), $L1−$L2, $L1−
     $L3: false (1.0), $L1−$L4: false (0.0), $L1−$L5: false (0.0), $L2, $L2−$L1: false
      (30.0/5.0), L2−$L1: false (0.0), $L2−$L3, $L3: /// (3.0/1.0), $L3−$L1: false
     (2.0), $L3−$L2: false (3.0), $L4: false (0.0), $L4−$L1: false (0.0), $L4−$L3:
     false (0.0), $L5: < (2.0/1.0), 0: false (1.0)
|    pos(1) = _NC
|    |    context_hes(0) = _: /// (1.0)
|    |    context_hes(0) = 0
|    |    context_hes(0) = H: false (22.0/4.0)
|    |    context_hes(0) = x: /// (1.0)
|    pos(1) = NPP: /// (22.0/10.0)
|    pos(1) = _NPP
|    |    pos(−3) = ADJ: O (1.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV: ) (1.0),
     _ADV: false (0.0), ADVWH: false (0.0), CC: false (0.0), _CC: false (0.0), CLO:
     false (0.0), CLR: false (0.0), CLS: false (0.0), _CLS: false (0.0), CS: false
     (0.0), _CS: false (0.0), DET: /// (2.0/1.0), _DET: false (0.0), DETWH: false
     (0.0), ET: false (0.0), I: false (0.0), _I: false (0.0), NC, _NC: false (0.0),
     NPP, _NPP, P, _P: false (0.0), P+D: false (0.0), P+PRO: false (0.0), PREF: false
     (0.0), PRO: false (1.0), _PRO: false (0.0), PROREL: false (0.0), PROWH: false
     (0.0), V: O (1.0), _V: false (0.0), VIMP: false (0.0), VINF: /// (1.0), _VINF:
     false (0.0), VPP: false (0.0), _VPP: false (0.0), VPR: false (0.0), _VPR: false
     (0.0), VS: false (0.0), 0: false (0.0)
|    pos(1) = P: false (453.0/141.0)
|    pos(1) = _P: false (47.0)
|    pos(1) = P+D
|    |    context_syllableSlowing(1) <= 3.774: false (29.0)
|    |    context_syllableSlowing(1) > 3.774
|    pos(1) = P+PRO: false (2.0)
|    pos(1) = PREF: false (0.0)
|    pos(1) = PRO: /// (70.0/34.0)
|    pos(1) = _PRO
|    |    context_montee(−2) <= 0.660857: O (3.0/1.0)
|    |    context_montee(−2) > 0.660857: /// (3.0)
```

```
|   pos(1) = PROREL: false (128.0/17.0)
|   pos(1) = PROWH: false (0.0)
|   pos(1) = V
|   |   context_hes(0) = _: /// (2.0/1.0)
|   |   context_hes(0) = 0
|   |   context_hes(0) = H: false (48.0/5.0)
|   |   context_hes(0) = x: false (8.0/1.0)
|   pos(1) = _V
|   |   context_duree_moy(0) <= 145: /// (2.0)
|   |   context_duree_moy(0) > 145: false (6.0)
|   pos(1) = VIMP: false (0.0)
|   pos(1) = VINF: false (38.0/5.0)
|   pos(1) = _VINF
|   |   context_montee(0) <= 0.629609: false (3.0)
|   |   context_montee(0) > 0.629609: /// (3.0)
|   pos(1) = VPP
|   |   context_hes(0) = _: false (0.0)
|   |   context_hes(0) = 0
|   |   context_hes(0) = H: false (16.0/2.0)
|   |   context_hes(0) = x: false (5.0/1.0)
|   pos(1) = _VPP: false (3.0/1.0)
|   pos(1) = VPR: false (1.0)
|   pos(1) = _VPR: false (1.0)
|   pos(1) = VS: false (5.0/1.0)
|   pos(1) = 0: /// (55.0/1.0)
context_duree_moy(1) > 0
|   intro = B: false (170.0/30.0)
|   intro = I: false (85.0/3.0)
|   intro = L
|   |   pos(1) = ADJ: false (6.0/1.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV:
    false (12.0/1.0), _ADV: false (8.0), ADVWH: false (1.0), CC: false (0.0), _CC:
    false (0.0), CLO: /// (1.0), CLR: false (0.0), CLS: /// (22.0/3.0), _CLS: false
    (0.0), CS: false (4.0), _CS: /// (14.0/6.0), DET, _DET: false (0.0), DETWH: false
     (0.0), ET: /// (2.0/1.0), I, _I: false (0.0), NC: false (4.0), _NC: false
    (3.0/1.0), NPP: false (1.0), _NPP: false (1.0), P, _P: false (0.0), P+D: false
    (2.0), P+PRO: false (0.0), PREF: false (0.0), PRO: false (6.0/1.0), _PRO: false
    (0.0), PROREL: false (10.0), PROWH: false (0.0), V, _V: false (0.0), VIMP: false
    (0.0), VINF: false (1.0), _VINF: false (0.0), VPP: false (1.0), _VPP: false (0.0)
    , VPR: false (0.0), _VPR: false (0.0), VS: false (0.0), 0: false (0.0)
|   intro = U
|   |   pos(1) = ADJ, _ADJ: /// (0.0), ADJWH: /// (0.0), ADV, _ADV: false (12.0/3.0),
    ADVWH: /// (5.0), CC: /// (2.0), _CC: false (1.0), CLO, CLR: /// (0.0), CLS: ///
    (124.0/9.0), _CLS: /// (0.0), CS: false (12.0/1.0), _CS: /// (0.0), DET, _DET:
    /// (0.0), DETWH: /// (0.0), ET: /// (2.0/1.0), I, _I: /// (0.0), NC, _NC, NPP:
    /// (1.0), _NPP: /// (6.0/1.0), P: false (49.0/14.0), _P: /// (0.0), P+D: false
    (3.0), P+PRO: /// (0.0), PREF: /// (0.0), PRO, _PRO: /// (0.0), PROREL: /// (1.0)
    , PROWH: /// (0.0), V: /// (34.0/2.0), _V: /// (0.0), VIMP: /// (1.0), VINF: ///
    (0.0), _VINF: /// (0.0), VPP: /// (1.0), _VPP: /// (0.0), VPR: /// (0.0), _VPR:
    /// (0.0), VS: /// (0.0), 0: /// (0.0)
|   intro = 0
|   |   context_syllableSlowing(0) <= 0.171
|   |   context_syllableSlowing(0) > 0.171
```

Arbre de décision sur données partielles

```
context_duree_moy(1) <= 0
|   pos(1) = ADJ
|   |   context_periode-Analor(1) = _: false (0.0), -, !, $, 0: /// (2.0/1.0),
    basdescendant: > (1.0), basmontant: false (0.0), basplat: false (0.0),
    hautdescendant: /// (2.0/1.0), hautmontant: false (0.0), hautplat: /// (1.0),
    normaldescendant, normalmontant, normalplat: false (5.0/1.0)
|   pos(1) = _ADJ
|   |   context_hauteur_moy(2) <= 87.885227: false (4.0)
|   |   context_hauteur_moy(2) > 87.885227: /// (2.0)
|   pos(1) = ADJWH: /// (2.0)
|   pos(1) = ADV
|   |   chunk(1) = B-AdP, B-AP: /// (0.0), B-CONJ: /// (0.0), B-NP: /// (0.0), B-PP:
    /// (0.0), B-__UNKNOWN__: /// (0.0), B-VN, I-AdP: /// (0.0), I-AP: /// (0.0), I-
    CONJ: /// (0.0), I-NP, I-PP, I-__UNKNOWN__: /// (0.0), I-VN, O: /// (0.0), 0: ///
     (0.0)
|   pos(1) = _ADV
|   |   context_periode-Analor(1) = _: false (0.0), -, !, $, 0: ) (2.0/1.0),
    basdescendant: false (3.0/1.0), basmontant: false (1.0), basplat: false (1.0),
    hautdescendant: false (4.0/1.0), hautmontant: /// (1.0), hautplat: O (1.0),
    normaldescendant, normalmontant, normalplat
|   pos(1) = ADVWH
|   |   context_hauteur_moy(3) <= 84.501298: false (4.0)
|   |   context_hauteur_moy(3) > 84.501298
|   pos(1) = CC: O (459.0/55.0)
```

```
|   pos(1) = _CC: false (0.0)
|   pos(1) = CLO: false (23.0/9.0)
|   pos(1) = CLR: false (5.0/1.0)
|   pos(1) = CLS
|   |   context_syllableSlowing(-1) <= 1.711: /// (313.0/101.0)
|   |   context_syllableSlowing(-1) > 1.711
|   pos(1) = _CLS: false (1.0)
|   pos(1) = CS
|   |   pos(3) = ADJ: O (2.0/1.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV, _ADV:
    false (0.0), ADVWH: O (2.0/1.0), CC: false (0.0), _CC: false (0.0), CLO, CLR: (
    (1.0), CLS, _CLS: false (0.0), CS: false (0.0), _CS, DET, _DET: false (0.0),
    DETWH: false (0.0), ET: false (0.0), I: >+ (1.0), _I: false (0.0), NC, _NC: false
     (0.0), NPP: false (0.0), _NPP: false (0.0), P: false (1.0), _P: false (0.0), P+D
    : false (0.0), P+PRO: false (0.0), PREF: false (0.0), PRO: O (3.0), _PRO: false
    (0.0), PROREL: false (0.0), PROWH: false (0.0), V, _V: false (0.0), VIMP: false
    (0.0), VINF: false (1.0), _VINF: false (0.0), VPP: false (0.0), _VPP: false (0.0)
    , VPR: false (0.0), _VPR: false (0.0), VS: false (1.0), 0: false (0.0)
|   pos(1) = _CS
|   |   context_montee(-2) <= 3.965103: false (18.0)
|   |   context_montee(-2) > 3.965103: /// (2.0/1.0)
|   pos(1) = DET
|   |   context_syllableSlowing(1) <= 4.601: false (225.0/74.0)
|   |   context_syllableSlowing(1) > 4.601
|   pos(1) = _DET: false (6.0/1.0)
|   pos(1) = DETWH: false (0.0)
|   pos(1) = ET
|   |   context_locuteur(-1) = _: /// (0.0), $L1, $L1-$L2: false (3.0), $L1-$L3: false
     (1.0), $L1-$L4, $L1-$L5: /// (0.0), $L2: false (3.0), $L2-$L1: /// (3.0/1.0), L2
    -$L1: /// (0.0), $L2-$L3: /// (0.0), $L3: /// (4.0/2.0), $L3-$L1: false (6.0/2.0)
    , $L3-$L2: /// (0.0), $L4: /// (0.0), $L4-$L1: /// (0.0), $L4-$L3: /// (0.0), $L5
    : /// (0.0), 0: /// (0.0)
|   pos(1) = I
|   |   context_periode-Analor(1) = _: /// (0.0), -, !: false (120.0/30.0), $, 0,
    basdescendant: false (7.0/1.0), basmontant: /// (5.0/1.0), basplat: /// (1.0),
    hautdescendant: false (4.0), hautmontant: < (2.0/1.0), hautplat: /// (0.0),
    normaldescendant, normalmontant, normalplat
|   pos(1) = _I: false (0.0)
|   pos(1) = NC: false (243.0/76.0)
|   pos(1) = _NC
|   |   pos(2) = ADJ: /// (0.0), _ADJ: /// (0.0), ADJWH: /// (0.0), ADV: false (1.0),
    _ADV: /// (0.0), ADVWH: /// (0.0), CC: O (3.0), _CC: /// (0.0), CLO: /// (0.0),
    CLR: /// (0.0), CLS: /// (1.0), _CLS: /// (0.0), CS: /// (1.0), _CS: /// (0.0),
    DET: /// (0.0), _DET: /// (0.0), DETWH: /// (0.0), ET: /// (0.0), I, _I: ///
    (0.0), NC: /// (0.0), _NC, NPP: /// (0.0), _NPP: /// (0.0), P: /// (0.0), _P: ///
     (0.0), P+D: /// (1.0), P+PRO: /// (0.0), PREF: /// (0.0), PRO: /// (0.0), _PRO:
    /// (0.0), PROREL: /// (0.0), PROWH: /// (0.0), V: false (2.0), _V: /// (0.0),
    VIMP: /// (0.0), VINF: /// (0.0), _VINF: /// (0.0), VPP: /// (0.0), _VPP: ///
    (0.0), VPR: /// (0.0), _VPR: /// (0.0), VS: /// (0.0), 0: /// (0.0)
|   pos(1) = NPP: /// (22.0/10.0)
|   pos(1) = _NPP
|   |   pos(-3) = ADJ: O (1.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV: ) (1.0),
    _ADV: false (0.0), ADVWH: false (0.0), CC: false (0.0), _CC: false (0.0), CLO:
    false (0.0), CLR: false (0.0), CLS: false (0.0), _CLS: false (0.0), CS: false
    (0.0), _CS: false (0.0), DET: /// (2.0/1.0), _DET: false (0.0), DETWH: false
    (0.0), ET: false (0.0), I: false (0.0), _I: false (0.0), NC, _NC: false (0.0),
    NPP, _NPP, P, _P: false (0.0), P+D: false (0.0), P+PRO: false (0.0), PREF: false
    (0.0), PRO: false (1.0), _PRO: false (0.0), PROREL: false (0.0), PROWH: false
    (0.0), V: O (1.0), _V: false (0.0), VIMP: false (0.0), VINF: /// (1.0), _VINF:
    false (0.0), VPP: false (0.0), _VPP: false (0.0), VPR: false (0.0), _VPR: false
    (0.0), VS: false (0.0), 0: false (0.0)
|   pos(1) = P
|   |   pos(2) = ADJ, _ADJ: false (0.0), ADJWH: false (0.0), ADV, _ADV: false (0.0),
    ADVWH: >+ (1.0), CC: false (0.0), _CC: false (0.0), CLO, CLR: false (0.0), CLS:
    false (0.0), _CLS: false (0.0), CS: false (0.0), _CS: false (0.0), DET, _DET:
    false (0.0), DETWH: false (0.0), ET: false (0.0), I, _I: false (0.0), NC, _NC:
    false (0.0), NPP, _NPP: false (0.0), P, _P, P+D: false (0.0), P+PRO: false (0.0),
     PREF: false (0.0), PRO, _PRO: false (0.0), PROREL: /// (2.0/1.0), PROWH: false
    (1.0), V: false (0.0), _V: false (0.0), VIMP: false (0.0), VINF, _VINF: false
    (0.0), VPP: false (0.0), _VPP: false (0.0), VPR, _VPR: false (0.0), VS: false
    (0.0), 0: false (0.0)
|   pos(1) = _P: false (47.0)
|   pos(1) = P+D
|   |   context_syllableSlowing(1) <= 3.774: false (29.0)
|   |   context_syllableSlowing(1) > 3.774
|   pos(1) = P+PRO: false (2.0)
|   pos(1) = PREF: false (0.0)
|   pos(1) = PRO
|   |   pos(0) = ADJ: /// (3.0), _ADJ: /// (0.0), ADJWH: /// (0.0), ADV, _ADV: ///
    (4.0/1.0), ADVWH: /// (0.0), CC: false (1.0), _CC: /// (0.0), CLO: /// (0.0), CLR
```

```
    : /// (0.0), CLS: /// (0.0), _CLS: /// (0.0), CS: /// (2.0/1.0), _CS: false (1.0)
    , DET: /// (5.0/1.0), _DET: /// (0.0), DETWH: /// (0.0), ET: /// (0.0), I, _I:
    /// (0.0), NC: /// (13.0/2.0), _NC: /// (0.0), NPP: /// (0.0), _NPP: < (1.0), P:
    false (3.0), _P: false (1.0), P+D: /// (0.0), P+PRO: /// (0.0), PREF: /// (0.0),
    PRO, _PRO: /// (0.0), PROREL: false (1.0), PROWH: /// (0.0), V, _V: /// (0.0),
    VIMP: /// (0.0), VINF: /// (0.0), _VINF: /// (0.0), VPP: /// (2.0/1.0), _VPP: ///
     (0.0), VPR: /// (0.0), _VPR: /// (0.0), VS: /// (0.0), 0: /// (0.0)
|   pos(1) = _PRO
|   |   context_montee(-2) <= 0.660857: O (3.0/1.0)
|   |   context_montee(-2) > 0.660857: /// (3.0)
|   pos(1) = PROREL: false (128.0/17.0)
|   pos(1) = PROWH: false (0.0)
|   pos(1) = V
|   |   pos(0) = ADJ, _ADJ: ) (1.0), ADJWH: false (0.0), ADV, _ADV: /// (1.0), ADVWH:
    /// (1.0), CC, _CC: false (0.0), CLO: false (4.0), CLR: false (1.0), CLS: false
    (43.0/2.0), _CLS: false (0.0), CS: /// (1.0), _CS: false (0.0), DET: false (0.0),
     _DET: false (0.0), DETWH: false (0.0), ET: false (0.0), I, _I: false (0.0), NC,
    _NC: /// (1.0), NPP, _NPP: false (0.0), P: false (0.0), _P: false (0.0), P+D:
    false (0.0), P+PRO: false (0.0), PREF: false (0.0), PRO, _PRO: false (0.0),
    PROREL: false (13.0/1.0), PROWH: false (0.0), V: false (3.0), _V: false (0.0),
    VIMP: false (0.0), VINF: false (0.0), _VINF: false (0.0), VPP: /// (2.0/1.0),
    _VPP: false (0.0), VPR: false (0.0), _VPR: false (0.0), VS: false (0.0), 0: false
     (0.0)
|   pos(1) = _V
|   |   context_duree_moy(0) <= 145: /// (2.0)
|   |   context_duree_moy(0) > 145: false (6.0)
|   pos(1) = VIMP: false (0.0)
|   pos(1) = VINF: false (38.0/5.0)
|   pos(1) = _VINF
|   |   context_montee(0) <= 0.629609: false (3.0)
|   |   context_montee(0) > 0.629609: /// (3.0)
|   pos(1) = VPP
|   |   pos(-2) = ADJ: false (5.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV, _ADV:
    /// (1.0), ADVWH: false (0.0), CC, _CC: false (0.0), CLO: false (1.0), CLR: false
     (0.0), CLS, _CLS: false (0.0), CS: false (2.0), _CS: false (0.0), DET, _DET:
    false (0.0), DETWH: false (0.0), ET: false (0.0), I: /// (1.0), _I: false (0.0),
    NC: false (15.0/3.0), _NC: /// (1.0), NPP: false (0.0), _NPP: false (0.0), P:
    false (2.0), _P: false (0.0), P+D: false (0.0), P+PRO: false (0.0), PREF: false
    (0.0), PRO: /// (4.0/1.0), _PRO: false (0.0), PROREL, PROWH: false (0.0), V, _V:
    false (0.0), VIMP: false (0.0), VINF: /// (2.0/1.0), _VINF: false (0.0), VPP:
    false (0.0), _VPP: false (0.0), VPR: false (0.0), _VPR: false (0.0), VS: false
    (0.0), 0: false (0.0)
|   pos(1) = _VPP: false (3.0/1.0)
|   pos(1) = VPR: false (1.0)
|   pos(1) = _VPR: false (1.0)
|   pos(1) = VS: false (5.0/1.0)
|   pos(1) = 0: /// (55.0/1.0)
context_duree_moy(1) > 0
|   context_syllableSlowing(0) <= 0.217
|   |   pos(1) = ADJ: false (39.0/14.0), _ADJ: false (0.0), ADJWH: false (0.0), ADV,
    _ADV, ADVWH: ( (1.0), CC: O (29.0/4.0), _CC: false (0.0), CLO, CLR: false (0.0),
    CLS: /// (27.02/7.0), _CLS: false (1.0), CS, _CS: false (13.0/3.0), DET: false
    (40.0/13.0), _DET: false (1.0), DETWH: false (0.0), ET: false (29.0/10.0), I, _I:
     false (0.0), NC, _NC, NPP: false (2.0), _NPP, P, _P: false (11.02), P+D: false
    (1.0), P+PRO: false (0.0), PREF: false (0.0), PRO, _PRO: /// (1.0), PROREL: ///
    (3.0), PROWH: false (0.0), V, _V: /// (1.0), VIMP: false (0.0), VINF: false (5.0)
    , _VINF: false (3.0), VPP: false (10.0/1.0), _VPP: false (1.0), VPR: false (0.0),
     _VPR: false (0.0), VS: false (0.0), 0: false (0.0)
|   context_syllableSlowing(0) > 0.217
|   |   pos(1) = ADJ: false (1649.0/132.0), _ADJ: false (62.0/3.0), ADJWH, ADV: false
    (2127.0/321.0), _ADV: false (1146.98/159.0), ADVWH: false (39.0/11.0), CC, _CC,
    CLO: false (350.0/42.0), CLR: false (112.0/1.0), CLS, _CLS: false (9.0/1.0), CS:
    false (468.0/87.0), _CS: false (271.0/25.0), DET: false (3165.0/227.0), _DET:
    false (43.0), DETWH: ( (1.0), ET: false (154.0/28.0), I, _I: false (0.0), NC:
    false (4873.96/114.0), _NC: false (564.98/100.0), NPP: false (292.0/19.0), _NPP:
    false (241.0/42.0), P: false (2642.0/217.0), _P: false (640.98/5.0), P+D: false
    (417.0/16.0), P+PRO: false (1.0), PREF: false (1.0), PRO, _PRO: false (37.0/5.0),
     PROREL: false (511.0/26.0), PROWH: false (3.0), V: false (3083.96/195.0), _V:
    false (142.0/9.0), VIMP: /// (1.0), VINF: false (695.0/5.0), _VINF: false
    (27.0/1.0), VPP: false (891.0/87.0), _VPP: false (17.0/2.0), VPR: false
    (41.0/2.0), _VPR: false (2.0), VS: false (37.0/1.0), 0: /// (2.0)
```

Index