HAL
open science

Robust French syntax analysis: reconciling statistical methods and linguistic knowledge in the Talismane toolkit

Assaf Urieli

To cite this version:

Assaf Urieli. Robust French syntax analysis: reconciling statistical methods and linguistic knowledge in the Talismane toolkit. Computation and Language [cs.CL]. Université Toulouse le Mirail - Toulouse II, 2013. English. NNT : . tel-00979681

HAL Id: tel-00979681

https://theses.hal.science/tel-00979681v1

Submitted on 16 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *Université Toulouse 2 Le Mirail (UT2 Le Mirail)*

Présentée et soutenue le *17 décembre 2013* **par :**

Assaf URIELI

Robust French syntax analysis: reconciling statistical methods and linguistic knowledge in the Talismane toolkit

Analyse syntaxique robuste du français : concilier méthodes statistiques et connaissances linguistiques dans l'outil Talismane

---

JURY

Alexis NASR	Professeur, LIF, Marseille	Rapporteur
Eric WEHRLI	Professeur, LATL, Genève	Rapporteur
Marie CANDITO	Maître de Conférences, ALPAGE, Paris Diderot	Examinatrice
Nabil HATHOUT	Directeur de Recherche, CLLE-ERSS, Toulouse	Examinateur
Ludovic TANGUY	Maître de Conférences, CLLE-ERSS, Toulouse	Directeur de Thèse

---

École doctorale et spécialité :
CLESCO : Sciences du langage
Unité de Recherche :
CLLE-ERSS (UMR 5263)
Directeur de Thèse :
Ludovic TANGUY
Rapporteurs :
Alexis NASR et Eric WEHRLI

Contents

List of abbreviations and acronyms **9**

List of Figures **10**

List of Tables **11**

Introduction **13**

I Robust dependency parsing: a state of the art **21**

1 Dependency annotation and parsing algorithms **23**
1.1 Dependency annotation of French . . . 23
1.1.1 Token and pos-tag annotations . . . 23
1.1.2 Dependency annotation standards . . . 24
1.1.3 A closer look at certain syntactic phenomena . . . 28
1.1.3.1 Relative subordinate clauses . . . 28
1.1.3.2 Coordination . . . 30
1.1.3.3 Structures not covered by the annotation scheme . . . 32
1.1.4 Projectivity . . . 33
1.1.5 Standard output formats . . . 35
1.2 Dependency parsing algorithms . . . 36
1.2.1 Rationalist vs. empiricist parsers . . . 36
1.2.2 Graph-based parsers . . . 37
1.2.3 Transition-based parsers . . . 41
1.3 Discussion . . . 45

2 Supervised machine learning for NLP classification problems **47**
2.1 Preliminary definitions . . . 47
2.2 Annotation . . . 51
2.3 Linguistic Context . . . 51
2.4 Features . . . 52
2.5 Training . . . 53
2.6 Analysis . . . 55
2.6.1 Pruning via a beam search . . . 55
2.7 Evaluation . . . 58
2.8 Classifiers . . . 59

2.8.1 Converting non-numeric features to numeric values 60
2.8.2 Perceptrons 61
2.8.3 Log-linear or maximum entropy models 64
2.8.3.1 GIS algorithm for maximum entropy training 65
2.8.3.2 Additive smoothing 66
2.8.3.3 Inverting numeric features 66
2.8.4 Linear SVMs 68
2.8.5 Classifier comparison 69
2.9 Supervised machine learning project examples 70
2.9.1 Authorship attribution 70
2.9.2 Jochre: OCR for Yiddish and Occitan 72
2.9.3 Talismane—Syntax analysis for French 73
2.10 Discussion 73

II Syntax analysis mechanism for French 75

3 The Talismane syntax analyser - details and originality 77
3.1 Philosophy 77
3.2 Architecture 78
3.3 Problem definition for Talismane's modules 79
3.3.1 Sentence boundary detection 79
3.3.2 Tokenisation 80
3.3.2.1 Tokenisation mechanism 83
3.3.3 Pos-tagging 85
3.3.4 Parsing 86
3.3.4.1 Transition-based parsing algorithm 87
3.3.4.2 Measuring parser confidence 87
3.3.4.3 Applying a beam search to parsing 88
3.3.4.4 Incremental parse comparison strategies 88
3.4 Formally defining features and rules 91
3.4.1 Using named and parametrised features 92
3.4.2 Defining feature groups to simplify combination 94
3.4.3 Features returning multiple results 94
3.5 He who laughs last: bypassing the model with rules 95
3.6 Filtering the raw text for analysis 97
3.7 Comparison to similar projects 98
3.8 Discussion 99

4 Incorporating linguistic knowledge 101
4.1 Training corpora 101
4.1.1 Errare humanum est: Annotation reliability 102
4.1.2 French Treebank 103
4.1.3 French Treebank converted to dependencies 107
4.2 Evaluation corpora 108
4.2.1 Sequoia 109
4.2.2 Wikipedia.fr discussion pages 109

4.2.3 Unannotated corpora . . . 112
4.3 External Resources . . . 113
4.3.1 Generalising features using external resources . . . 113
4.3.2 Talismane's definition of a lexicon . . . 114
4.3.3 LeFFF . . . 115
4.4 Baseline features . . . 115
4.4.1 Cutoff . . . 116
4.4.2 Sentence detector baseline features . . . 116
4.4.3 Tokeniser baseline features . . . 117
4.4.4 Pos-tagger baseline features . . . 119
4.4.5 Parser baseline features . . . 121
4.5 Baseline rules . . . 125
4.5.1 Tokeniser . . . 125
4.5.2 Pos-tagger . . . 125
4.6 Discussion . . . 126

III Experiments 127

5 Evaluating Talismane 129
5.1 Evaluation methodology . . . 129
5.1.1 Parse evaluation metrics . . . 131
5.1.2 Statistical significance . . . 131
5.2 Evaluating classifiers and classifier parameters . . . 132
5.2.1 Evaluating classifiers for parsing . . . 132
5.2.1.1 Tuning perceptron parameters for parsing . . . 132
5.2.1.2 Tuning MaxEnt parameters for parsing . . . 134
5.2.1.3 Tuning linear SVM parameters for parsing . . . 135
5.2.1.4 Comparing the best configurations for parsing . . . 136
5.2.2 Evaluating classifers for pos-tagging . . . 137
5.2.2.1 Tuning perceptron parameters for pos-tagging . . . 138
5.2.2.2 Tuning MaxEnt parameters for pos-tagging . . . 139
5.2.2.3 Tuning linear SVM parameters for pos-tagging . . . 139
5.2.2.4 Comparing the best configurations for pos-tagging . . . 139
5.2.3 Combining the pos-tagger and the parser . . . 140
5.3 Experiments with system confidence . . . 142
5.4 Experiments with beam search . . . 145
5.4.1 Applying the beam to the pos-tagger . . . 146
5.4.2 Applying the beam to the parser . . . 147
5.5 Experiments with beam propagation . . . 147
5.5.1 Using the parser and pos-tagger to correct tokenisation errors . . . 148
5.5.2 Using the parser to correct pos-tagging errors . . . 149
5.5.3 Using beam propagation to improve parsing . . . 151
5.6 Comparison to similar studies . . . 152
5.7 Discussion . . . 153

6 Targeting specific errors with features and rules 157

6.1 Features or rules? . . . 158
6.2 Using targeted pos-tagger features . . . 159
6.2.1 Identifying important pos-tagger errors . . . 159
6.3 Improving the tagging of *que* . . . 161
6.3.1 Recognising *que* as a negative adverb . . . 161
6.3.1.1 Development corpus error analysis . . . 162
6.3.1.2 Feature list . . . 163
6.3.1.3 Results . . . 165
6.3.2 Recognising *que* as a relative or interrogative pronoun . . . 165
6.3.2.1 Development corpus error analysis . . . 165
6.3.2.2 Feature list . . . 166
6.3.2.3 Results . . . 172
6.3.3 Effect of targeted pos-tagger features on the parser . . . 173
6.4 Using targeted parser features . . . 175
6.4.1 Parser coordination features . . . 175
6.4.1.1 Development corpus error analysis . . . 177
6.4.1.2 Feature list . . . 179
6.4.1.3 Results . . . 182
6.5 Using rules . . . 184
6.5.1 Pos-tagger closed vs. open classes . . . 185
6.5.2 Pos-tagger rules for *que* . . . 185
6.5.3 Parser rules: prohibiting duplicate subjects . . . 187
6.5.4 Parser rules: prohibiting relations across parentheses . . . 190
6.6 Discussion . . . 191

7 Improving parsing through external resources 193
7.1 Incorporating specialised lexical resources . . . 194
7.2 Augmenting the lexicon with GLÀFF . . . 195
7.3 Injecting resources built by semi-supervised methods . . . 197
7.3.1 Semi-supervised methods for domain adaptation . . . 198
7.3.2 Distributional semantic resources . . . 200
7.3.3 Using the similarity between conjuncts to improve parsing for coordination 201
7.4 Discussion . . . 206

Conclusion and perspectives 209

A Evaluation graphs 217

Bibliography 225

Acknowledgements

This thesis was not written in isolation, but rather within a strongly supportive context, both inside the CLLE-ERSS linguistics research laboratory, and in my wider personal and professional life. Before embarking on the subject matter itself, I would like to thank all of those without whom this work would not have been possible.

First and foremost, I would like to thank my thesis adviser, Ludovic Tanguy. Ludovic gave me his enthusiastic support from our very first meeting in his apartment, when the University was closed down due to student strikes, a support that didn't slacken even after I spilled beer on him (twice!) in Amsterdam. His guidance throughout my studies, and especially after I started writing my dissertation, has been priceless. He gave me the freedom to explore to my heart's content, all the while gently nudging me in the right direction. In hindsight I find that his advice has always been thoroughly sound, no matter how dubitative I may have been on first hearing it. And although his numerous revision marks and questions on my printed thesis have been a challenge to decipher ("*des pattes de mouche*" as we say in French), they were of countless help towards making this thesis worth reading. Where my writing is good and clear, it is thanks to him. Where it leaves something to be desired, it is invariably because I refused to listen!

I would also like to thank the members of the jury—Alexis Nasr and Eric Wehrli, for accepting to write the detailed reports, and Marie Candito and Nabil Hathout, for accepting to be oral examiners.

My gratitude goes out to the other members of the CLLE-ERSS lab, and especially those of the NLP group in Toulouse with whom I was the most in contact. In alphabetical order (as no other will do), these include Basilio Calderone, Cécile Fabre, Bruno Gaume, Lydia-Mai Ho-Dac, Marie-Paule Péry-Woodley, and Franck Sajous. Franck, who never complained when I asked him at the very last minute to construct yet another enormous resource, deserves a special mention.

I also wish to thank my fellow doctoral students and post-doctoral researchers, and most especially those who shared my office in room 617: Clémentine Adam, François Morlane-Hondère, Nikola Tulechki, Simon Leva, Lama Allan, Maxime Warnier and Marin Popan, as well as those from nearby offices who often stopped in for a chat— Fanny Lalleman, Stéphanie Lopez, Aurélie Guerrero, Florian Savreux and Marie-France Roquelaure among others—not to mention the Prox team members: Yannick Chudy, Yann Desalle, Benoit Gaillard and Emmanuel Navarro. A special thanks goes to Matteo Pascoli and Marianne Vergez-Couret, who drove all the way down to Foix in the pouring rain to hear me sing in Yiddish. Thanks to Marianne, my work on Yiddish OCR has taken an international turn, and expanded to include Occitan.

As this thesis was unfinanced, it is only thanks to the understanding of my professional colleagues and customers, allowing me to take time off for study, that I have been able to

progress. First of all, there is the team in Coup de Puce: Ian Margo, who welcomed me in Toulouse twenty years ago and has accompanied me ever since, Estelle Cavan, who was one of the first to test Talismane in a real work environment, and all of the others. In the French Space Agency, Daniel Galarreta deserves my thanks for his trust and interest in my work. At the Yiddish Book Center, I would like to thank Aaron Lansky, Katie Palmer Finn, Catherine Madsen, Josh Price and Agnieszka Ilwicka for their support and enthusiasm around the Jochre Yiddish OCR project. There are also my many former colleagues at Storm Technology, Avaya, and Channel Mechanics: Karl Flannery, Brendan Walsh, Olivia Walsh and Kenneth Fox, to name but a few. Finally, there are my recent colleagues at CFH, and in particular Eric Hermann and Michel Mazeau thanks to whom I've managed to put natural language processing at the heart of my working life.

With respect to Talismane itself, I need to thank Jean-Phillipe Fauconnier, for his excellent work testing and documenting my software, and Marjorie Raufast, for accepting the arduous task of correcting Talismane's syntax analysis of Wikipedia discussion pages.

If I have taken an interest in computational linguistics, it is largely thanks to my family: my father Izzi, for transmitting to me his love for computers and algorithms; my mother Nili for transmitting to me her love for languages; my grandparents Lova and Miriam, may they rest in peace, whose presence in my childhood has meant so much to me; and my sisters and their families—Sharon and little Topaz, and Michal and Philip.

Last but not least, this thesis would never have been possible without the help and support of my wife Joelle, and my two wonderful boys, Mischa and Dmitri. It is thanks to Joelle's insistence that I left my computer from time to time, to climb on my bicycle and go riding through the surrounding countryside. Joelle and the boys have stood by me throughout these years of study, and have made this life worth living. Thank you!

List of abbreviations and acronyms

- **CNES**: Centre National d'Etudes Spatiales (French Space Agency)
- **FTB**: French Treebank
- **FTBDep**: French Treebank automatically converted to dependencies
- **GLÀFF**: Gros Lexique À tout Faire du Français
- **LAS**: Labeled attachment score
- **LeFFF**: Lexique des Formes Fléchies du Français
- **MST**: Maximum Spanning Tree
- **NLP**: Natural Language Processing
- **POS**: Part Of Speech
- **SVM**: Support Vector Machine
- **UAS**: Unlabeled attachment score
- **WSJ**: Wall Street Journal

List of Figures

1.1 Graph-based MST parsing, step 1: fully connected digraph 39
1.2 Graph-based MST parsing, step 2: maximum incoming arcs 39
1.3 Graph-based MST parsing, step 3: cycles as nodes 40
1.4 Graph-based MST parsing, step 4: final graph 40

2.1 Classification through supervised machine learning 48
2.2 Pos-tagging through supervised machine learning 49
2.3 Supervised machine learning evaluation 50
2.4 Classification using perceptrons 61
2.5 Applying the kernel trick to a 2-class SVM: $y \rightarrow x^2$ 69

5.1 Evaluation corpora LAS for a perceptron classifier using different values for iterations i and cutoff 133
5.2 Evaluation corpora LAS for a perceptron classifier using lower values for iterations i and cutoff 134
5.3 Training time and analysis time (FTBDep-dev) for a MaxEnt classifier using different values for iterations i and cutoff 135
5.4 Training time and analysis time (FTBDep-dev) for a linear SVM using different values for C and ϵ 136
5.5 Parsing classifier comparison: LAS by corpus 137
5.6 Pos-tagging classifier comparison: accuracy by corpus 140
5.7 Accuracy loss (LAS) from gold-standard pos-tags, with and without jackknifing . 142
5.8 Correct answers and remaining dependencies based on confidence cutoff, for the FTBDep dev corpus 144
5.9 Accuracy and remaining dependencies based on confidence cutoff, for various evaluation corpora 144
5.10 Mean confidence vs LAS/UAS 146
5.11 LAS with and without propagation for the FTBDep dev and FrWikiDisc corpora 151
5.12 LAS by beam size, with propagation 152
5.13 LAS for all dependencies where distance $\leq n$, at different beam widths, FTBDep test corpus 153

A.1 Parser evaluation corpora LAS for a MaxEnt classifier using different values for iterations i and cutoff 218
A.2 Parser evaluation corpora LAS for a linear SVM using different values of C and ϵ 219
A.3 Parser evaluation corpora LAS for a linear SVM using different values of C and cutoff 220

A.4 Pos-tagger evaluation corpora accuracy for a MaxEnt classifier using different values for iterations i and cutoff . . . 221
A.5 Pos-tagger evaluation corpora accuracy for a perceptron classifier using different values for iterations i and cutoff . . . 222
A.6 Pos-tagger evaluation corpora LAS for a linear SVM using different values of C and ϵ . . . 223
A.7 Pos-tagger evaluation corpora LAS for a linear SVM using different values of C and cutff . . . 224

List of Tables

1.1 French tagset used in this thesis [Crabbé and Candito, 2008] . . . 24
1.2 Dependency labels for verbal governors [Candito et al., 2011b] . . . 26
1.3 Dependency labels for non-verbal governors [Candito et al., 2011b] . . . 27
1.4 Additional more specific relations, currently reserved for manual annotation only [Candito et al., 2011b] . . . 28
1.5 Sample Talismane output using the CoNLL-X format . . . 35
1.6 The arc-standard transition system for shift-reduce dependency parsing . . . 41
1.7 Sample transition sequence using the arc-standard shift-reduce transition system 42
1.8 Alternative parse using the arc-standard shift-reduce transition system . . . 43
1.9 The arc-eager transition system for shift-reduce dependency parsing . . . 43
1.10 Deterministic parse using an arc-eager shift-reduce transition system . . . 44
1.11 Principle differences between transition-based and graph-based parsers . . . 45

2.1 Beam search example—correct solution in bold . . . 56
2.2 Classifier comparison . . . 69
2.3 Authorship attribution results . . . 71
2.4 Jochre Yiddish OCR results . . . 73
2.5 Jochre Occitan OCR results . . . 73

3.1 Incremental parse comparison strategies, labelled accuracy at different beams . . 90
3.2 Talismane feature syntax: operators . . . 92
3.3 Talismane feature syntax: a subset of generic functions . . . 93
3.4 Examples of Talismane filters . . . 97

4.1 Top 30 common nouns in the French Treebank . . . 106
4.2 Example sentence from the FTBDep corpus . . . 108
4.3 Sequoia corpus interannotator agreement (f-score average) from Candito et al. [2012] 109
4.4 FrWikiDisc corpus inter-annotator agreement (Cohen's kappa) . . . 110
4.5 FrWikiDisc corpus characteristics . . . 111
4.6 FrWikiDisc corpus non-projective arc types . . . 112

4.7 FrWikiDisc corpus manual dependency label count 112

5.1 Baseline method vs. alternative method contingency table 131
5.2 Comparison of the best classifier configurations for parsing 138
5.3 Comparison of the best classifier configurations for pos-tagging 141
5.4 Pos-tagging and parsing accuracy combined, with and without jackknifing 142
5.5 Point-biserial correlation for system confidence to correct parses on the FTBDep development corpus 143
5.6 Point-biserial correlation by label for system confidence to correct parses on the FTBDep dev corpus 145
5.7 Pos-tagger accuracy at various beam widths for different classifiers, on the FTBDep dev corpus 146
5.8 Parser accuracy at various beam widths for different classifiers, for the FTBDep dev corpus 147
5.9 Beam propagation from the tokeniser to the pos-tagger 148
5.10 Testing tokeniser beam propagation on unannotated corpora 149
5.11 Pos-tagger accuracy at various beam widths for different classifiers, for the FTBDep dev corpus, with and without propagation 150
5.12 Testing pos-tagger beam propagation on unannotated corpora 150
5.13 Change in precision, recall and f-score from beam 1 by label, for all corpora combined 154
5.14 Comparison of the Talismane baseline to other parsers for French 154

6.1 Pos-tagger function word and number errors causing the most parsing errors in the FTBDep dev corpus 160
6.2 Baseline confusion matrix for *que* 161
6.3 Confusion matrix for *que* with targeted negative adverb features 165
6.4 Confusion matrix for *que* with targeted relative pronoun features 172
6.5 Confusion matrix for *que* with all targeted features combined 173
6.6 Parsing improvement for using a pos-tagger model with targeted *que* features . . 174
6.7 Arc-eager transition sequence for coordination, with difficult decisions in bold . . 176
6.8 Parsing improvement for the relation `coord` using targeted features 182
6.9 Parsing improvement for the relation `dep_coord` using targeted features 183
6.10 Contingency table for `coord` with and without targeted features, at beam 1 . . . 183
6.11 Contingency table for `coord` with and without targeted features, at beam 2 . . . 183
6.12 Pos-tagging accuracy improvement using closed class rules 185
6.13 Confusion matrix for *que* with targeted features and rules 186

7.1 Coverage by lexicon (percentage of unknown words) 196
7.2 Number of word-pairs per corpus and confidence cutoff 203
7.3 Parsing improvement for coordination using distributional semantic features . . . 204

Introduction

Preamble

I'm often asked: why does a man nearing 40, with a comfortable career as a software engineer, begin to study for a doctorate in linguistics? Before plunging into the heart of the matter, I'll attempt to answer this question by placing my thesis in its personal, local, and global context.

A few months before starting my thesis, I was approached by the French Space Agency CNES with a request: "*can you give us a quote for an automatic bilingual terminology extractor?*" I had completed my Master's almost 15 years earlier, had worked for a few years as a translator, and then as a software engineer for over ten years. One of my freelance software projects had been Aplikaterm, a web application for terminology management, still used by the CNES and its translators today. So, given what I knew of terminology, I answered "*do you have a budget for a few years of full-time development?*"

Nevertheless, I decided to do some research and see what was available on the market. Because of my relative strength in algorithms as opposed to statistics, I was attracted by a more formal approach based on preliminary syntax analysis, as opposed to a more "naïve" statistical approach based on collocation. My surprise was great when I discovered that a syntax analyser, Syntex, had been developed in the nearby Université de Toulouse, by a certain Didier Bourigault. A project started forming in my head: why not adapt this syntax analyser, if it's available, to bilingual terminology extraction? And, if not, why not build a syntax analyser of my own? Finally, why not fund this project by fulfilling a longstanding latent ambition of mine: to pursue a doctorate? Indeed, the daily humdrum of web and database application development was beginning to bore me, and I was yearning for something that would engage my intellect more fully. I wrote an e-mail containing a proposal for such a doctorate to the Natural Language Processing section of the CLLE-ERSS linguistics lab, and received an answer not long afterwards: the NLP section was interested in my proposal, and would like to meet me in person to discuss it. That's how I met Ludovic Tanguy, who was to become my thesis adviser.

Professionally, I felt I was making a move in the right direction. We have only begun to tap into the possibilities of information analysis offered by the vast quantities of multilingual text available both on the Web and inside corporate intranets. Certainly, huge progress had been made by the likes of Google, but this progress remained to a large extent locked behind the walls of corporate intellectual property. I have been a consumer of open source software for many years, and am convinced that this is one of the best paradigms for encouraging innovation, both in academia and in private enterprise, especially in the case of smaller companies without the muscle, time or money to enforce patents.

As it turned out, my proposal to write a robust open source syntax analyser had come

just at the right moment. The NLP section of the CLLE-ERSS lab needed just such an analyser for many of their projects, but Syntex had been purchased by Synomia, and was no longer available. On the other hand, I had no formal education in linguistics, and my passive knowledge was limited to that of a man who speaks several languages and had worked as a professional translator. What is more, there was no direct funding available for my thesis, and so I would have to continue my daily job as a web and database application developer while pursuing the doctorate. This situation was difficult to manage, but at the same time it gave me far more freedom than that of a doctoral student funded by a company, forced to examine only those problems directly affecting the company's earning capacity, and to handle only the technical corpora directly related to the company's business.

Unbeknownst to me, as a software engineer with little prior knowledge of linguistics, I embodied an ongoing debate within the NLP world [Cori and Léon, 2002, Tanguy, 2012]: what was the role of the linguist in a field increasingly dominated by large-scale statistical algorithms locked up in a "black box" of mathematical complexity? I myself had decided to write a statistical, rather than a rule-bases system, both because I was convinced, after reading the state of the art, that such a system would be more robust and simpler to maintain, and because it would allow me to make the most of my algorithmic strengths. Over time, I became aware of my privileged role in a linguistics laboratory as an experienced software engineer who had helped to build many robust enterprise systems. Yet I would have to find ways to make the most of an often contradictory situation: my statistical system left little room for traditional linguistics in the form of rules and grammars; and yet, I wanted to allow as many hooks as possible to the team of linguists surrounding me, so that I could make the most of their contributions.

The end result is a thesis which explores neither computer science nor linguistics as a field of study in and of itself. Rather, its feet are firmly anchored in software engineering but its eyes are set on the practical applicability of statistical machine learning algorithms to the large-scale resolution of linguistic ambiguity. My approach aims to be as didactic as possible in those areas where I had to spend many months mastering a particular concept or technique. I delve into the statistical methods used by machine learning algorithms to a certain extent, but only as deeply as is required to gain insights into the effect of different configurations on the final result. This thesis thus explores practical problems within natural language processing, and possible solutions to these problems.

Overview

This thesis concentrates on the application of supervised statistical machine learning techniques to French corpus linguistics, using the purpose-built Talismane syntax analyser. As such, it is directly concerned with all the steps needed to construct a system capable of analysing French text successfully: from corpus selection, to corpus annotation, to the selection and adaptation of machine learning algorithms, to feature selection, and finally to tuning and evaluation. I had little control over some of these areas (e.g. selection and annotation of the training corpus), and so will only examine the choices made by others and the effects they had on my work. In those areas where I could exercise full control, I will attempt to convey the possibilities that were available, the reasoning behind my choices, and the evaluation of these choices.

The main achievements of this thesis are:

- proposal of a practical and highly configurable open source syntax analyser to the scientific community, with an out-of-the-box implementation for French, attaining state-of-the-art performance when compared to similar tools using standard evaluation methodology;
- insights into which statistical supervised machine learning techniques can lead to significant improvement in results, and on the limits of these techniques;
- creation of a flexible grammar for feature and rule definition, and insights into how to write features that generalise well beyond the training corpus, when to use rules instead of features, and what type of phenomena can be successfully analysed via the introduction of highly targeted features;
- insights into the types of external resources that can be most effective in helping the syntax analyser, whether manually constructed or automatically derived in a semi-supervised approach; and into the best ways to integrate these resources into the analysis mechanism.

The choice of French as a language of study was of course motivated by the preparation of the doctorate in a French university. French is an interesting case, in that it has far fewer linguistic resources available than English, but far more than many other languages. The question of resources is thus central to the thesis. However, it is hoped that many of the lessons learned can be applied to other languages as well, and can provide guidance on the resources that can or should be constructed. Indeed, Talismane was designed to be usable with any language, and anything specific to French has either been externalised as a configuration file, or as a separate software package that implements interfaces defined by the main package.

Before starting, I wish to examine the angle from which this thesis tackles certain key concepts, and the questions it will attempt to resolve.

Robust syntax analysis

In Bourigault [2007, pp. 9, 25], robustness is defined as the ability of a syntax analyser to parse a large corpus in a reasonable amount of time. In a robust system, emphasis is thus placed on practical usability as opposed to theoretical linguistic aspects. In this thesis, the term *robust* is extended to include its accepted meaning within software engineering: a system capable of functioning regardless of anomalies or abnormalities in the input. By "capable of functioning", we imply that the system should provide as complete an analysis as possible within a reasonable amount of time. A full analysis is preferable to a partial one, and a partial analysis is preferable to no analysis at all. Of course, "reasonable" depends on what you are trying to do—a web-site may need a response within seconds, whereas it may be reasonable to analyse a 100 million word corpus in a week. Because of the time constraint, robustness implies compromises in terms of analysis complexity, which force us to select algorithms and feature sets compatible with our computational resources.

Syntax analysis here falls under the tradition of dependency syntax [Tesnière, 1959]. It involves finding a single governor for each word in the sentence, and a label for the dependency relation drawn between the governor and the dependent. A distinction is made here between *syntax analysis*, which is the full transformation of raw text into a set of dependency trees, and *parsing*, which is typically the last step of syntax analysis, taking a sequence of pos-tagged and lemmatised tokens as an input, and producing a dependency tree as output.

Supervised machine learning

Supervised machine learning is a technique whereby a software system examines a corpus annotated or at least corrected by a human (the *training corpus*), and attempts to learn how to apply similar annotations to other, similar corpora. One weakness in this definition is the use of the word "similar": how does one measure the "similarity" between two corpora, and how does one measure the "similarity" between annotations? I am specifically interested in *statistical* machine learning techniques, in which the linguist defines *features*, and the machine learning algorithm automatically decides how much weight to assign to each feature, in view of the data found in the annotated training corpus, by means of a *probabilistic classifier*. Moreover, I concentrate on the re-definition of all linguistic annotation tasks as *classification* problems. Supervised machine learning is covered in chapter 2, and its adaptation to the specific problem of syntax analysis in chapter 3.

Corpus

According to Sinclair [2005]:

> A corpus is a collection of pieces of language text in electronic form, selected according to external criteria to represent, as far as possible, a language or language variety as a source of data for linguistic research.

The notion of *corpus* is central to supervised machine learning, first and foremost through the necessity for a *training corpus*, that should be as representative as possible of the type of language that we wish to analyse, and of course as large as possible (following the adage "There is no data like more data", attributed to Bob Mercer of IBM). Moreover, the simplest evaluation of success involves using the system to analyse a manually annotated *evaluation corpus* and comparing system annotations to manual annotations. The degree of similarity between the training corpus and each evaluation corpus is bound to affect the accuracy of analysis results. One of the questions most central to this thesis is: what methods and resources allow us to generalise beyond the information directly available in the training corpus, so that the system is capable of correctly analysing text different in genre, register, theme, etc.?

In addition to their use in the processing chain for a given application, tools such as Talismane can be directly applied to corpus linguistics, enabling, for example, the syntax annotation and subsequent comparative analysis of two corpora.

The corpora used by our present study are presented in chapter 4.

Annotation

In this thesis, *annotation* is the addition of meta-textual information to a corpus, which indicates how to interpret ambiguities within the text. Because this thesis redefines linguistic questions as classification problems, it is concerned with those annotations which can be transformed into a closed set of pre-defined classes.

The areas we are specifically interested in are:

- Segmentation: segmenting raw text into sentences (sentence boundary detection) and segmenting a sentence into minimal syntactic units (tokenisation). In terms of clas-

sification, this is redefined as: is the border between two symbols separating or non-separating?

- Classification: Assigning parts-of-speech to tokens (POS tagging). This is a classical classification problem and needs no redefining.
- Relation: drawing governor-dependent arcs between tokens and labelling these arcs. In terms of classification, this is redefined as: given two words in a sentence, is there a dependency arc between them and, if so, what is the arc direction and what is the label?

For each classification problem, the degree to which the classes are coarse or fine-grained will affect ease of annotation, ease of automatic classification and usefulness of the annotation to downstream tasks. The needs of each of these three areas are often contradictory. Ill-defined or badly chosen classes can either lead to a situation where the initial ambiguity cannot be resolved by the classes available, or where classification forces a somewhat arbitrary choice for an unambiguous case. Moreover, classification attempts to subdivide a problem into clear-cut classes, whereas real data includes many borderline cases. The annotation conventions used in this these are discussed in chapter 1.

Linguistic resources

A *linguistic resource* is any electronic resource (i.e. file or database) describing a particular aspect of a language or sub-language (e.g. dialect, technical field, etc.), and organised in a manner that can be directly interpreted by a software system. These resources can include annotated corpora, lexical resources, ontologies, formal grammars, etc. It is fairly straightforward to make use of "generic" resources with pretence to cover the language as a whole rather than a sub-language. A practical question in supervised machine learning is whether any use can be made of resources specific to the sub-language of an evaluation corpus. If a resource is not applicable to the training corpus, how can we take it into account in the learning process? Resources are initially introduced in chapter 4. Experiments with the incorporation of external resources are presented in chapter 7.

Applications

Only a subset of linguists are interested in syntax analysis as an end in and of itself—and these are generally horrified by the approximate results provided by statistical machine learning systems. Most others are only interested in syntax analysis as a means to an end: automatic translation, automatic transcription of speech, terminology extraction, information retrieval, linguistic resource construction, etc. In this thesis, an *application* refers to the "client" of the syntax analyser, being the next step in the chain: the extraction of term candidates, the construction of an automatic semantic resource, etc. A first question is: Does syntax analysis produce better results than approaches which ignore syntax? This thesis, prepared in a laboratory with a long tradition of syntax analysis using the Syntex software, will take this as a given. A next question is: is the same type of syntax analysis required for all applications? Some applications may only be concerned with verb predicate structure, others may not be concerned at all with long-distance relations. Is it possible to tune the syntax analyser to increase precision or recall for certain phenomena at the cost of others? Applications also differ by their target corpora: terminology extraction, for example, typically concerns

technical domains and genres. To what extent can we tune the syntax analyser for these corpora, in terms of methodology and resources? While we do not tackle these questions directly in the present thesis, they guide us in terms of our system design, and the future perpectives to which we would like it to open.

Thesis plan

This thesis is divided into three parts. **Part 1, "Robust dependency parsing: a state of the art"**, contains the following chapters:

- **Chapter 1: Dependency annotation and parsing algorithms.** This chapter presents the final annotation results we wish our system to reproduce, explaining how to annotate certain complex syntactic phenomena in French. It then gives a brief history of dependency parsing, and the various types of parsing algorithms that have been used, concentrating on transition-based parsing used in the present thesis.

- **Chapter 2: Supervised machine learning for NLP classification problems.** This chapter explains the formal concepts behind supervised machine learning, in terms of training, analysis and evaluation, illustrated by the case study of part-of-speech tagging. Finally, it gives an overview of the three major supervised machine learning projects in which I was involved: Jochre (for OCR), authorship attribution, and the Talismane syntax analyser.

Part 2, "Syntax analysis mechanism for French", takes the concepts presented in the previous chapters and shows how they are implemented in the current thesis. It is divided into the following chapters:

- **Chapter 3: The Talismane syntax analyser: details and originality.** In this chapter, we delve into the philosophy behind the Talismane syntax analyser, and its general architecture. We examine how each of the steps performed by Talismane is transformed into a classification problem within the four modules: sentence detection, tokenisation, pos-taging and parsing. We examine the methods used to define features, rules and filters. Finally, we compare Talismane with similar projects.

- **Chapter 4: Incorporating linguistic knowledge.** In this chapter we look into the various methods and resources used to incorporate linguistic knowledge into the syntax analysis mechanism. We also analyse the specific resources used for the default French implementation of Talismane: the French Treebank (and a version automatically converted into dependencies), various other evaluation corpora, and the LeFFF lexicon. Finally, we describe the baseline features used by all four of Talismane's modules, and some of the reasoning behind the selection of these features.

Part 3, "Experiments", takes the baseline system defined in the previous chapters, evaluates it, and then attempts to improve on the initial evaluation results. It is divided into the following chapters:

- **Chapter 5: Evaluating Talismane.** In this chapter we perform an evaluation of Talismane with the baseline features from the previous chapter, using a variety of robust probabilistic classifiers and parameters, in order to select the best baseline configuration.

We also evaluate the contribution of the beam search and beam propagation, and explore the concept of system confidence.

- **Chapter 6: Targeting specific errors with features and rules.** In this chapter, we attempt to correct certain specific errors through the use of highly specific features and rules that tackle specific linguistic phenomena.

- **Chapter 7: Improving parsing through external resources.** In this chapter, we extend the study in the previous chapter, reviewing various ways for incorporating external resources into the features and rules. We give a state-of-the-art for semi-supervised approaches, describe distributional semantic resources, and present an experiment where these resources are used to improve the parsing of coordination.

We finish the thesis with our conclusions and future perspectives.

Part I

Robust dependency parsing: a state of the art

Chapter 1

Dependency annotation and parsing algorithms

This chapter examines the basic nuts-and-bolts of dependency parsing. However, before we begin to look at the parsing algorithms themselves, section 1.1 describes the end-result of the parsing task: what annotations is our system meant to produce? Section 1.2 then examines the various algorithms capable of producing such annotations, while leaving the statistical machine learning aspects to the following chapter. Our analysis of parsing algorithms will be colored by our primary objective: to maximize the amount of linguistic knowledge that can be injected into the system without sacrificing maintainability or robustness.

1.1 Dependency annotation of French

1.1.1 Token and pos-tag annotations

Before delving into dependency annotation itself, we examine two preliminary annotation tasks which are generally assumed as input to the parsing task: tokenising and pos-tagging.

A **token** is a single syntactic unit, which may or may not correspond to a single word as it appears graphically on the page. In French, for example, the subordinating conjunction *bien que* is a single token comprised of two separate words on the page, and known as a **compound word**. The word *duquel* is an agglutinated form, composed of two tokens, corresponding to the words *de* and *lequel*. Tokenisation is the task of automatically dividing a sentence into tokens. This thesis follows the fairly standard convention of marking compound words with an underscore (e.g. "bien_que"). The few existing aggultinated forms are not tokenised in any specific way (e.g. empty token insertion), because of their non-productive nature in French. Instead, they are assigned a compound pos-tag (`P+D` for "du", `P+PRO` for "duquel").

Pos-tagging is the task of assigning a part-of-speech (or POS) to each token. A **tagset** is the full set of POS-tags used for annotation. The tagset used in this dissertation is as per Crabbé and Candito [2008], with the exception of interrogative adjectives (`ADJWH`), which have been assimilated with interrogative determiners (`DETWH`), and includes the tags shown in table 1.1.

In our case, the pos-tagger also attempts to find a token's **lemma**: its basic non-inflected form that would be used as a dictionary entry, e.g. the infinitive for a conjugated verb, the singular for a noun, and the masculine singular for an adjective. Furthermore, it attempts

Tag	**Part of speech**
ADJ	Adjective
ADV	Adverb
ADVWH	Interrogative adverb
CC	Coordinating conjunction
CLO	Clitic (object)—A "clitic" is a pronoun always appearing in a fixed position with respect to the verb
CLR	Clitic (reflexive)
CLS	Clitic (subject)
CS	Subordinating conjunction
DET	Determiner
DETWH	Interrogative determiner
ET	Foreign word
I	Interjection
NC	Common noun
NPP	Proper noun
P	Preposition
P+D	Preposition and determiner combined (e.g. "du")
P+PRO	Preposition and pronoun combined (e.g. "duquel")
PONCT	Punctuation
PRO	Pronoun
PROREL	Relative pronoun
PROWH	Interrogative pronoun
V	Indicative verb
VIMP	Imperative verb
VINF	Infinitive verb
VPP	Past participle
VPR	Present participle
VS	Subjunctive verb

Table 1.1: French tagset used in this thesis [Crabbé and Candito, 2008]

to identify additional *sub-specified* morpho-syntactic information, such as the gender, number, person and tense. By "sub-specified", we mean that each piece of information (gender, number, etc.) is only provided when it is available—unlike the postag itself, which is always provided.

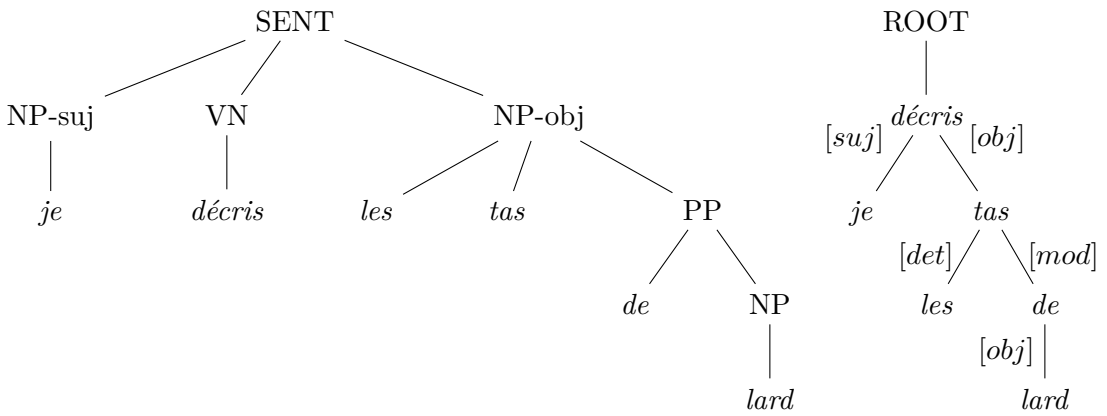
1.1.2 Dependency annotation standards

Automatic syntax analysis generally generates either constituency and dependency representations of a sentence, both of which represent the sentence as a tree. In constituency representations (also known as phrase-structure representations), directly inspired by the phrase-structure grammars found in [Chomsky, 1957], the intermediate nodes of the tree are

phrases whereas the leaf nodes are words. In dependency representations, first formalised by Tesnière [1959], all of the tree's nodes are words, with each word except for the central verb governed by another word.

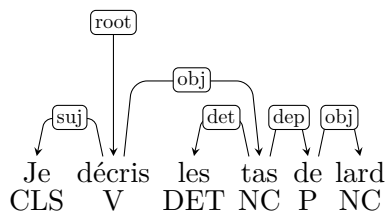
Example 1.1 shows a typical sentence in French, for which we have constructed a constituency tree (left) and a dependency tree (right). In the dependency tree, note in particular the addition of a root node, an artifact whose purpose is to guarantee that each word in the sentence has a governor, including the central verb.

Example 1.1 *Je décris les tas de lard.*
(I describe the piles of bacon = I'm describing the piles of bacon.)



However, it is far more common to visualise the dependency tree as a set of directed labelled arcs above the sentence, and this convention will be used in the present dissertation, as in example 1.2. Arcs will be directed from the governor to the dependent. When referring to individual arcs within this tree, we'll use the convention *label(governor, dependent)*, as in *suj(décris, je)* in the previous example. If a word appears more than once, we'll add a numeric index after the word, to indicate which occurrence is being referenced.

Example 1.2 *Je décris les tas de lard.*
(same sentence in typical dependency tree representation)



Note that we can easily convert a constituency tree into a dependency tree, except that a heuristic is required to determine the head of each phrase. Similarly, we can convert a dependency tree into a constituency tree, except that a heuristic is required to determine the phrase label.

In the above examples, the constituency tree uses structures defined for the French Treebank [Abeillé et al., 2003], hereafter FTB, which is the manually syntax-annotated resource most often used for training statistical French parsers. The dependency tree uses structures

from the automatic conversion of the French Treebank to a dependency treebank [Candito et al., 2010a], hereafter FTBDep. Indeed, except where otherwise indicated, all of the dependency trees in this section directly follow the annotation guide at Candito et al. [2011b]. These labels are summarised in table 1.2, table 1.3, and table 1.4[1]. In these three tables, the *dependent* is shown in italics, whereas the **governor** is shown in bold.

Label	**Description**	**Example**
suj	subject	*Il* **mange**.
obj	direct object	Il **mange** une *pomme*.
de_obj	argument introduced by *de*, non locative	Je **parle** *de* lui.
a_obj	argument introduced by *à*, non locative	Je **pense** *à* lui.
p_obj	argument introduced by another preposition	Je **lutte** *contre* les méchants.
ats	predicative adjective or nominal over the subject, following a copula	Il **est** *triste*.
ato	predicative adjective or nominal over the object	Je le **trouve** *triste*.
mod	adjunct (non-argumental preposition, adverb)	Je **pars** *après* lui.
aux_tps	tense auxiliary verb	Il *a* **mangé**.
aux_pass	passive auxiliary verb	Il a *été* **renvoyé**.
aux_caus	causative auxiliary verb	Je le *fais* **corriger**.
aff	clitics in fixed expressions (including refexive verbs)	Je *me* **lave**.

Table 1.2: Dependency labels for verbal governors [Candito et al., 2011b]

The dependency paradigm may thus be summarised as follows: for each word in the sentence, draw an arc attaching it to a single syntactic governor[2], and add a dependency label to each arc from a set of predefined labels.

This approach is attractive for several reasons: first, it directly translates the predicate-argument structure of verbs (and potentially other governors), by attaching the arguments directly to their governor and adding labels to indicate what type of arguments they are. Example 1.3 shows the predicate-argument structure for the verb "parler".

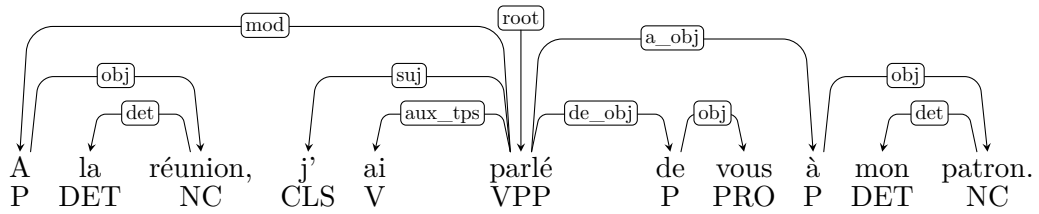
[1]We differ from the manual annotations in Candito *et al* in one aspect: the `mod_cleft` relation for cleft sentences marks the copula's predicate, rather the verb *être*, as the governor for the subordinate clause verb. In this respect, we annotate it identically to a normal relative subordinate clause.

[2]In this sense, the vast majority of modern dependency parsers differ from Tesnière's original work, which allowed multiple governors for each word

Label	**Description**	**Example**
root	governs the central verb of a sentence by an imaginary root node	[**root**] Je *mange* une pomme
obj	objects of prepositions and subordinating conjunctions	Le chien **de** ma *tante*. Il faut **que** je m'en *aille*.
mod	modifiers other than relative phrases	Une **pomme** *rouge*
mod_rel	links a relative pronoun's antecedent to the verb governing the relative phrase	La **pomme** que je *mange*
coord	links a coordinator to the immediately preceding conjunct	Je mange une **pomme** *et* une orange.
dep_coord	links a conjunct (other than the first one) to the previous coordinator	Je mange une pomme **et** une *orange*.
det	determiners	Je mange *une* **pomme**.
ponct	relation governing punctuation, except for commas playing the role of coordinators	Je **mange** une pomme *[.]*
dep	sub-specified relation for prepositional dependents of non-verbal governors (currently, no attempt is made to distinguish between arguments an adjuncts for non-verbal governors)	une **tarte** *aux* pommes
arg	used to tie together linked prepositions	Je mange **de** midi *à* 1h.

Table 1.3: Dependency labels for non-verbal governors [Candito et al., 2011b]

Example 1.3 *A la réunion, j'ai parlé de vous à mon patron.*
(At the meeting I have spoken of you to my boss = At the meeting, I spoke about you to my boss)

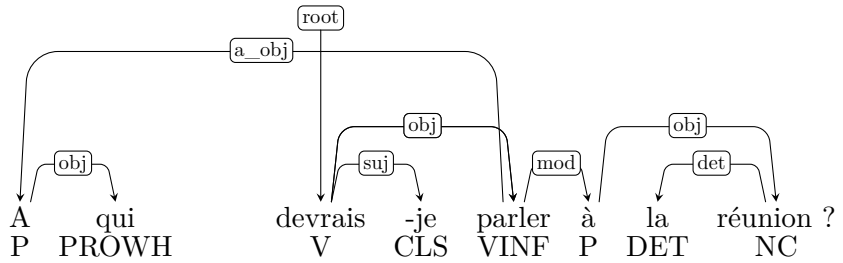


It is of course possible to include functional labels on the phrases in a constituency structure (as was done for `suj` and `obj` in example 1.1), mirroring the predicate-argument structure, but their interpretation is less direct. Furthermore, in dependency structures, it is straightforward to represent these predicate-argument structures even when they cross other arcs, as in example 1.4, where the `root` arc to the central verb *devrais* is crossed by the `a_obj` argument of the verb *parler*.

Label	Description	Example
p_obj_loc	locative verbal arguments (source, destination or localistion)	Il *y* **va**. Il **rentre** *chez* lui.
mod_loc	locative adjuncts	Je **mange** *à* la maison.
mod_cleft	in a cleft sentence, links the copula's predicate to the verb governing the relative clause	C'est **lui** qui a *volé* la pomme.
p_obj_agt	the preposition introducing the agent in the case of passive or causative	Il a été **mangé** *par* un ogre.
suj_impers	the impersonal subject *il*	*Il* **faut** partir.
aff_moyen	for the so-called "middle" reflexive construction in French (*"se moyen"*)	Ce champignon *se* **mange**.
arg_comp	links a comparative "que" with its adverbial governor	Il est **plus** grand *que* moi.
arg_cons	links a consecutive phrase to its adverbial governor	Il fait **trop** chaud *pour* sortir.

Table 1.4: Additional more specific relations, currently reserved for manual annotation only [Candito et al., 2011b]

Example 1.4 *A qui devrais-je parler à la réunion ?*
(To whom should I talk at the meeting?)



Another advantage of dependency representations is the relative ease with which they can be incorporated into machine learning algorithms. Indeed, the yes-or-no question of whether or not an arc should be drawn between two nodes is more straightforward for a machine than the open-ended question of where phrase frontiers start and end (see section 1.2 for algorithm details).

1.1.3 A closer look at certain syntactic phenomena

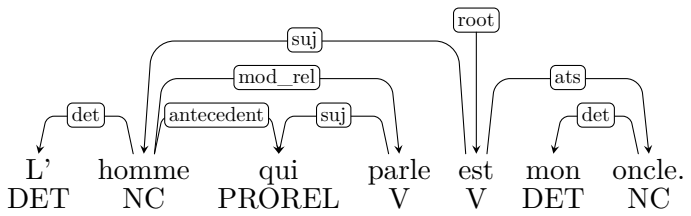
We have now outlined the general nature of dependency annotation. The specifics of this paradigm are best illustrated by applying it to several more complex syntactic structures.

1.1.3.1 Relative subordinate clauses

As mentioned previously, the vast majority of dependency parsers add a constraint (not in Tesnière's original work) that every word in the sentence can have only a single governor. This affects, for example, encoding standards for relative pronouns. Since we cannot have

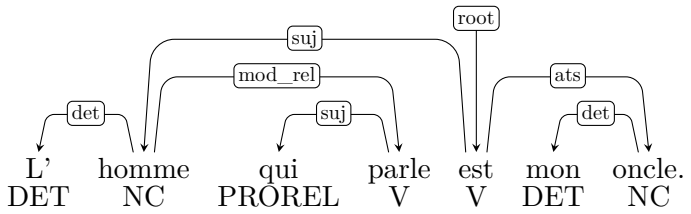
both the antecedent and the relative clause verb govern the relative pronoun, we have to choose one or the other. Take, for example, the sentence "*L'homme qui parle est mon oncle*" ("The man who's speaking is my uncle"). If we were to allow more than one governor, we might be tempted to annotate this sentence as in example 1.5, with both the antecedent and the relative clause verb governing the relative pronoun *qui*. However, we still have to decide who is to govern the central verb of the relative clause—the most likely candidate being the antecedent.

Example 1.5 *L'homme qui parle est mon oncle.*
(The man who speaks is my uncle = The man who's speaking is my uncle.)



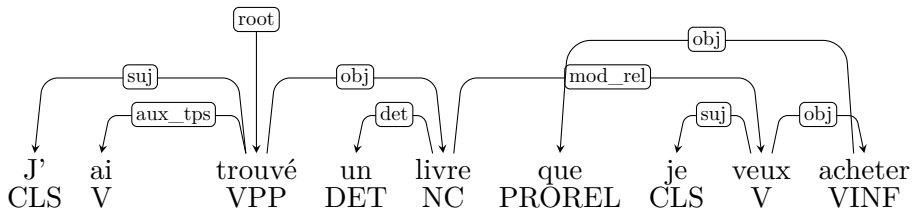
Now, one of the governors for *qui* has to be removed. Removing the arc *suj(parle,qui)* would hide the argument structure of the verb *parler*. Instead, we decide to remove the arc *antecedent(homme,qui)*, so that the antecedent has to be inferred by following the arcs backwards from *qui* until we reach the governor of the `mod_rel` relation. This solution is shown in example 1.6.

Example 1.6 *L'homme qui parle est mon oncle.*
(The man who speaks is my uncle = The man who's speaking is my uncle.)



Relatives can easily give rise to crossed dependencies, whenever the relative pronoun is the object of a verb other than the main verb of the relative clause. Example 1.7 gives an example of such a sentence.

Example 1.7 *J'ai trouvé un livre que je veux acheter.*
(I have found a book that I want to_buy.)



Now, representing this sentence in a constituency tree would be a challenge, since the verb phrase *acheter que* is discontinuous. It can be done only if we allow the leaf nodes of the constituency tree to violate the original word order. Moreover, the context-free grammars which are most commonly used to generate constituency trees would be incapable of producing such a tree, since they cannot generate discontinuous phrases.

1.1.3.2 Coordination

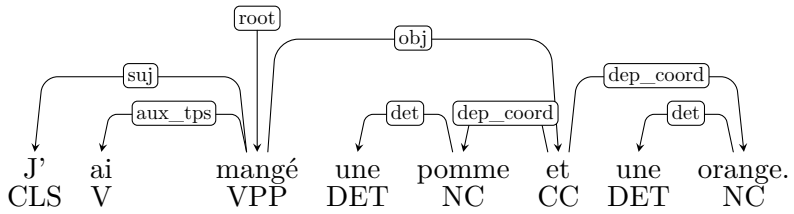
One syntactic structure which is inherently difficult to parse automatically is coordination. Indeed, the first conjunct of a coordinated structure can typically only be identified by examining the rest of the sentence. Consider the following three sentences:

1. J'ai mangé une pomme rouge et mûre.
2. J'ai mangé une pomme rouge et une orange.
3. J'ai mangé une pomme rouge et Georges a bu du thé.

In these fairly simple cases, it is enough to examine the parts-of-speech immediately following the coordinating conjunction in order to choose the correct conjuncts, except in the last case, where we have to decide whether or not Georges gets eaten. Nevertheless, we see that nothing preceding the coordinating conjunction can allow us to determine the first conjunct.

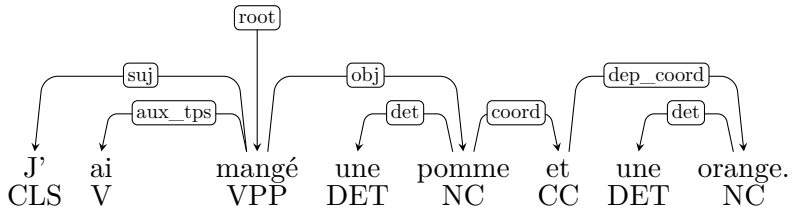
Two options are typically used in dependency annotation schemes for coordination: either we mark the coordinating conjunction as the head, or we mark the first conjunct as the head. Example 1.8 shows the first scheme, which is logically more attractive, in that it puts both of the conjuncts at the same level.

Example 1.8 *J'ai mangé une pomme rouge et une orange.*
(I have eaten an apple and an orange = I ate an apple and an orange)



In example 1.9, we see an alternative annotation scheme, which is the one used by Candito et al. [2011b], and hence the one used in this thesis. Here the head of the coordination structure is the first conjunct. While at first this is less logically appealing, it has a considerable practical advantage over the previous annotation scheme: any governors or dependents of the coordination as a whole are directly linked to the first conjunct (generally a content word), rather than to the coordinating conjunction. Therefore, the same set of features is useful regardless of whether the governor/dependent is a coordinated structure or a simple token.

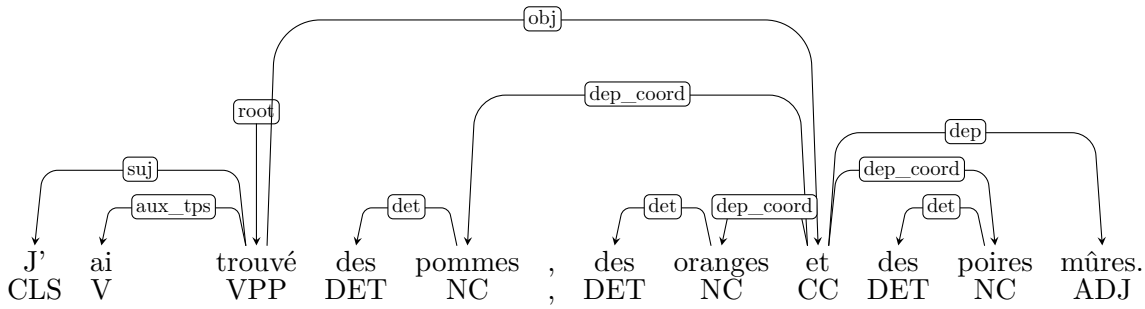
Example 1.9 *J'ai mangé une pomme rouge et une orange.*
(same as previous example, different annotation)



Consider the case shown in example 1.8. When trying to find the object of the verb *manger*, a parser would need to decide between *pomme* and *et*. This would require it to decide whether or not *pomme* is inside a coordinating structure, prior to identifying the object of *manger*. If the parser works left-to-right, it would have to recognise the coordinating structure prior to having constructed it. In the sentence *J'ai mangé une pomme rouge et mûre*, the parser would need to recognise that *et* coordinates *rouge* and *mûre* before being able to take any decision regarding the object of *manger*. The situation is far simpler in the first-conjunct headed scheme shown in example 1.9. Regardless of whether *pomme* is coordinated or not, it is always the object of *manger*. The decision as to what is coordinated by *et* can be taken separately, and does not have non-local consequences to other dependencies in the sentence.

Now, let us consider the case of three or more conjuncts, where the coordinated structure as a whole is modified by a dependent, as shown in example 1.10. In this case, there is an unresolvable ambiguity as to whether the adjective *mûres* modifies the entire coordination or the last conjunct only. In our annotation, we've assumed it modifies the entire coordination. This solution is quite elegant, and makes it immediately clear which tokens govern or are governed by the coordinated structure as a whole, and which ones are governed by a single conjunct.

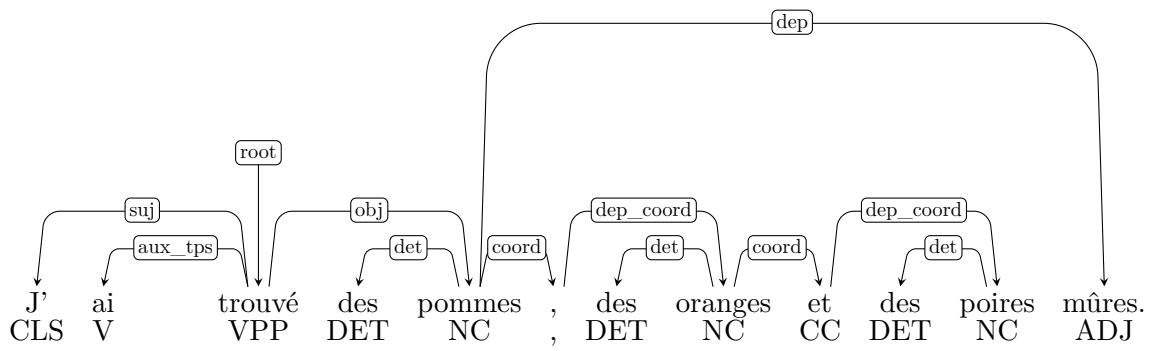
Example 1.10 *J'ai trouvé des pommes, des oranges et des poires mûres.*
(I have found some apples, some oranges and some pears ripe = I found some ripe apples, oranges and pears)



The same sentence is shown in example 1.11, but with Candito *et al*'s annotation scheme. In this conjunct-headed coordination scheme, the conjunction itself is governed by the conjunct immediately preceding it. Because the conjunction can have at most one governor, this coordination scheme needs to make use of punctuation as soon as more than two conjuncts are involved, in this case the coordinating comma (another option would have been to tie the first two conjuncts directly with a `dep_coord` relationship). The question of how to attach

the final adjective (assuming it modifies the entire coordination) is left open in Candito *et al*'s guide. In our own manual annotation projects, we have decided to attach it to the first conjunct as a fairly unambiguous sign that it is attached to the structure as a whole, just as the governor of the coordinated structure is made to govern the 1st conjunct.

Example 1.11 *J'ai trouvé des pommes, des oranges et des poires mûres.*
(Same as previous sentence, different annotation scheme)



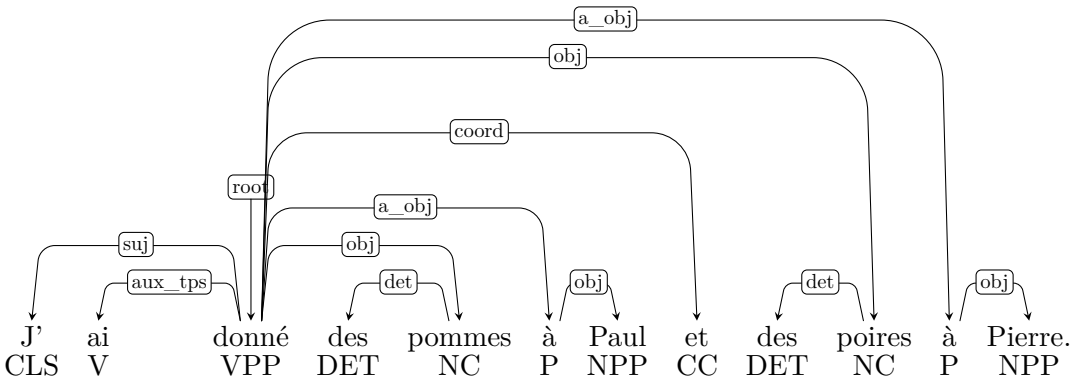
The question of specific features that help make coordination decisions within the context of the latter annotation scheme is handled in section 6.4.1.

1.1.3.3 Structures not covered by the annotation scheme

The surface annotation scheme described here is not capable of handling all syntactic constructs in French.

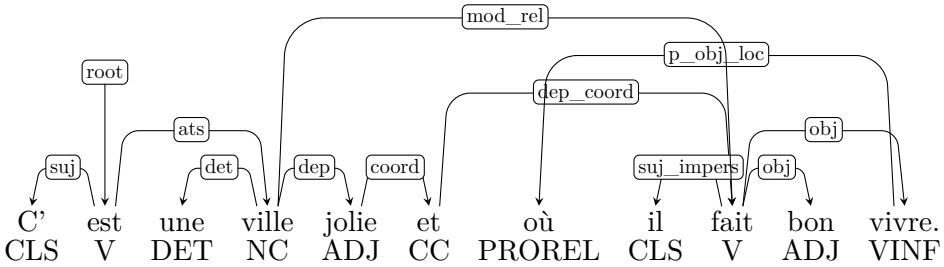
In particular, it is quite difficult to handle various elliptic structures (head-gapping, right-node raising, left-subject elision). Example 1.12 shows a head-gapping example. Without introducing an empty placeholder token after the conjunction *et*, there is no clear solution to indicating the second conjunct of the coordinated structure. Moreover, we'd need to show that the subject is governed by the coordination as a whole, whereas the direct and indirect objects are distributed among the conjuncts. The annotation shown gives the most satisfying solution on the assumption that empty placeholders are never introduced in surface annotation. At least it has the merit of uniquely identifying this type of elliptic structure: a verb with two arguments of the same type, separated by a coordinated conjunction with no dependent conjunct.

Example 1.12 *J'ai donné des pommes à Paul et des poires à Pierre.*
(I have given some apples to Paul and some pears to Peter = I gave apples to Paul and pears to Peter)



Other structures pose problems as well. Example 1.13 shows an example where we have to decide whether to privilege the coordination or the subordinate clause, since otherwise the verb *fait* has two governors. Neither solution is satisfying.

Example 1.13 *C'est une ville jolie et où il fait bon vivre.*
(It's a town pretty and where it makes good to_live = It's a pretty town where life is pleasant)



Such phenomena are presumably quite rare in usage, and statistical systems are notoriously bad at correctly annotating rare cases, even if we did manage to define clear annotation rules in such cases. In cases when no clearly correct annotation solution can be found, our secondary annotation objective becomes to annotate in such a way that can be easily and unambiguously identified when searching for these structures in a corpus.

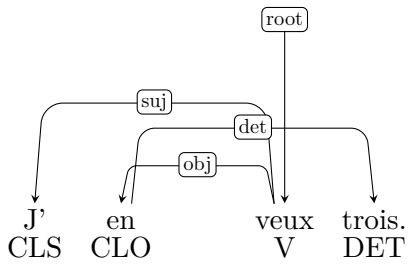
This enables us to measure a phenomenon's rarity in manually annotated corpora, even if its rarity implies that our statistical parser is unlikely to annotate the phenomenon correctly.

1.1.4 Projectivity

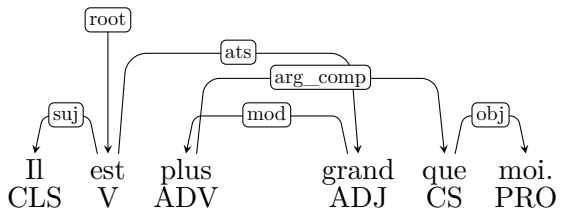
A distinction can now be made between *projective* and *non-projective* dependency trees. A projective dependency tree is one in which the dependency arcs can be drawn without any of the arcs crossing. An alternative and equivalent definition is that for a projective dependency tree, for any governor G and dependent D, all nodes lying between G and D are descendants of G. Languages such as French and English are largely projective, although there are some exceptions, as was already shown in example 1.4 and example 1.7.

Other typical cases of non-projectivity in French include the clitic *en* shown in example 1.14, the comparative shown in example 1.15, and the relative pronoun *dont* shown in example 1.16, which is a similar structure to the previous example with *que*. Finally, we have structures where the non-projectivity is optional and chosen by the speaker, as in example 1.17.

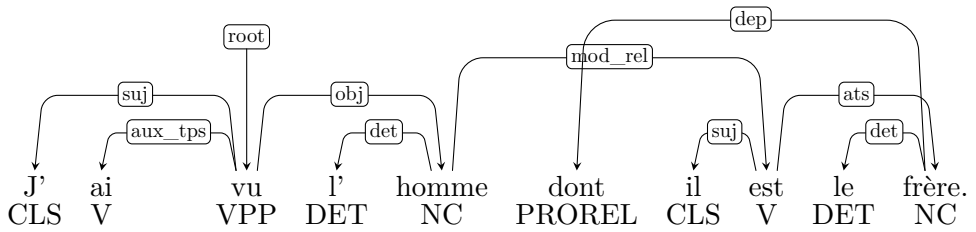
Example 1.14 *J'en veux trois.*
(I of_them want three = I want three of them)



Example 1.15 *Il est plus grand que moi.*
(He is more big than me = He's bigger than me)



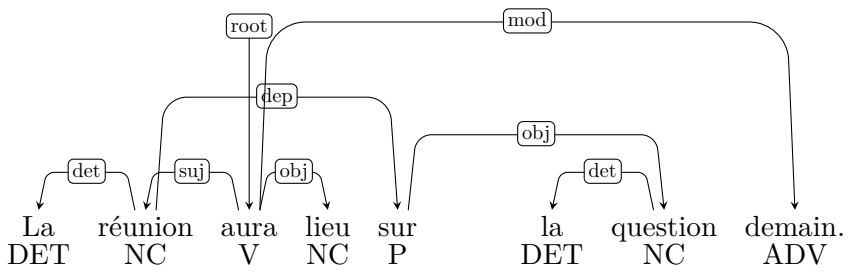
Example 1.16 *J'ai vu l'homme dont il est le frère.*
(I have seen the man of_whom he is the brother = I saw the man whose brother he is)



1	Je	je	CLS	cln	n=s|p=1	2	suj
2	décris	décrire	V	v	n=s|p=1|t=pst	0	root
3	les	le	DET	det	n=p	4	det
4	tas	tas	NC	nc	g=m	2	obj
5	de	de	P	P		4	dep
6	lard	lard	NC	nc	g=m|n=s	5	obj
7	.	.	PONCT	PONCT		2	ponct

Table 1.5: Sample Talismane output using the CoNLL-X format

Example 1.17 *La réunion aura lieu sur la question demain.*
(The meeting will take place on the question tomorrow)



1.1.5 Standard output formats

With the advent of dependency parsing evaluation campaigns such as Nivre et al. [2007a], most dependency parsers now propose a standardised output format derived from the CoNLL X task. Table 1.5 shows sample Talismane output using the CoNNL-X output format, for the sentence given in example 1.1.

In this structure, the columns are as follows:

1	Sequential index
2	Token
3	Lemma
4	Pos-tag
5	Grammatical category
6	Morpho-syntactic information
7	Index of governor (from column 1)
8	Dependency label

As can be seen, the individual dependency arcs can be extracted by taking columns 2 (dependent) and 8 (label), and then referencing column 7 to column 1 for the governor. Let us say we want to extract noun phrases for a terminology extractor: for a noun located on line n, this comes down to finding all of the nodes where n appears in column 7, and then recursively finding these nodes' children in a similar manner.

Many works in areas requiring a large number of frequency-weighted features to characterise a document take raw dependency arcs as input, typically in the form POS-label-POS, accepting any artifacts introduced by the dependency annotation scheme without further

analysis. This includes applications such as document classification/comparison [Goldberg and Orwant, 2013] and authorship attribution [Tanguy et al., 2011]. Other applications require the reconstruction of more or less complex relationships, including deep syntax (logical subject, etc.) from the surface annotations. Our annotation scheme attempts to make it possible to extract such information whenever possible.

1.2 Dependency parsing algorithms

Having examined the dependency annotation of French, we now turn to the algorithms used to generate these annotations. In this chapter, we will concentrate on the non-probabilistic part of the algorithm: the nuts and bolts used to construct the dependency tree, while abstracting away the machine learning system which gives probabilities to each decision at each step of the algorithm. In chapter 2 we will examine the working of probabilistic machine learning algorithms. Although there is a long tradition of constituency-tree generating algorithms, ranging from PCFG (probablistic context-free grammar) parsers [Collins, 1999, 2003, Charniak, 2000, Petrov et al., 2006] to TAG (tree-adjoining grammar) parsers [De La Clergerie et al., 2009], to parsers such as FIPS inspired by Chomsky's generative grammar [Wehrli, 2007], we will limit ourselves to parsers which directly generate dependency trees. Certain studies have automatically converted PCFG-generated trees to dependency trees [Candito et al., 2010b], but we will not explore such parsers in the present work.

Will make two distinctions: a first one between rationalist and empiricist parsers, and a second one, among empiricist parsers, between graph-based and transition-based parsers.

1.2.1 Rationalist vs. empiricist parsers

We follow Church [2011] in defining NLP techniques as either rationalist or empiricist. Although Church does not discuss parsing in particular, the general definition is easily applicable. In a rationalist parser, the linguist attempts to describe the language as completely as possible via a set of rules, and the system uses this description to attempt to parse unknown sentences. A rationalist parser can either use a formal grammar (CFG, HPSG, LFG, TAG, etc.), or a series of heuristic rules. It can make use of statistics to make certain decisions, but statistics are secondary, and are only called in to help when the built-in linguistic knowledge is insufficient to make a decision.

Empiricist methods, on the other hand, are completely data-driven, and use machine learning techniques to learn statistical probabilities from the data. They can be rules-based (e.g. PCFG), but rules and rule probabilities are learned directly from the training data. Linguistic knowledge can be incorporated into the system as features to describe the training data, but is not used to constrain decisions.

I will take Syntex [Bourigault, 2007] as an example of a rationalist dependency parser, in this case based on a series of heuristic rules. The choice of Syntex is motivated by that fact that (a) it was developed in our CLLE-ERSS laboratory and (b) it was the best-scoring parser in the EASY evaluation campaign for syntactic parsers of French [Paroubek et al., 2006]. The EASY campaign came about at time when rationalist parsers were at their apex for French, and empiricist parsers had not yet entered the arena, for the simple reason that the first (and only) treebank for French—the French Treebank [Abeillé et al., 2003]—had only just been completed. Indeed, one of the stated goals of the EASY campaign was the automatic generation of a treebank.

Syntex is built of a series of modules, each of which examines a particular type of syntactic relation (subject, direct object, etc.). It uses a waterfall approach, in which a sentence is analysed iteratively by different modules, each one using the dependency arcs added by the previous modules to make its decisions. In the first pass, local dependencies are added (determiners governed by nouns, objects governed by prepositions). In the second pass, unambiguous dependencies are added. In the third pass, ambiguous dependencies are resolved. In each pass, the various modules use a series of heuristic rules to make their decisions. The heuristic rules are coded within the modules, and no attempt was made to extract them into any sort of formal grammar. The order in which modules tackle a sentence in each pass is coded into the system as well. Thus, this system represents an attempt by a computational linguist, with skills in both programming and linguistics, to encode a vast amount of linguistic knowledge of French into a parser, thus enabling it to parse any sentence using this knowledge.

The fact that this system performed better than formal grammar driven parsers seems to indicate that formal grammars are not flexible enough to encode the linguistic knowledge required to parse a sentence correctly, or else that the grammars written for French had not yet reached the required maturity.

Syntex also uses so-called "endogenous" statistics to attempt to resolve the most ambiguous questions, e.g. identifying the governors of prepositional phrases and relative clauses. Endogenous here refers to search inside the entire corpus being analysed for cases of unambiguous attachment using the same dependent and one of the potential governors. This use of statistics is radically different to data-driven approaches, where the entire parsing model is determined from annotated data in the first place.

There are two main criticisms that can be leveled against rationalist parsers.

First of all, by definition, a strictly rule-based system cannot be robust: all input is divided into grammatical and ungrammatical sentences, and only grammatical sentences can be parsed. To become robust, such systems need to include a principled method for relaxing constraints. This is particularly true of formal-grammar-driven systems. It is much more difficult to judge the robustness of heuristics-driven systems, since the lack of a formal grammar means it is very hard to determine to what extent the system can analyse a particular sentence.

The second and more important question is that of maintainability. Because of the complexity of human language, a rationalist system has to encode a huge quantity of rules (whether grammatical or heuristic). Making even a minor change, such as the order in which two rules are applied, can have drastic consequences to parsing decisions, far beyond the capabilities of a linguist to imagine ahead of time. Testing modifications can only be made through trial-and-error on regression test suites, and thus, in the long run, the system is no less opaque than a statistical model, and far more difficult to maintain.

1.2.2 Graph-based parsers

Purely empirical approaches started appearing in the 1990's for English. They were first tested on French in Candito et al. [2010b], following the availability of an annotated treebank [Abeillé et al., 2003]. Unlike rationalist systems, which attempt to encode a linguist's knowledge of the language as a whole, empiricist systems are limited to those phenomena directly observable in the training corpus. On the other hand, they tend to be robust, with no *a priori* concept of grammaticality. Furthermore, they tend to be fairly easy to maintain: the linguistic knowledge is encoded in a set of features, whose weights are determined automatically by the machine

learning algorithm. Thus, adding or modifying a given feature is not likely to have drastic effects on the system as a whole.

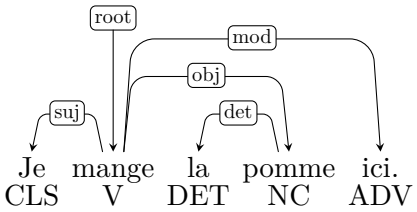
We will follow Kübler et al. [2009] in dividing empiricist dependency parsers into two main categories: graph-based and transition-based. Graph-based systems treat the dependency tree as a graph, and apply algorithms originating in graph theory. Central to this type of parsing is the notion of a graph score, typically calculated as a recursive function of the sub-graphs forming the total graph. This type of parser was first introduced by Eisner [1996] for projective sentences, and was considerably extended by McDonald et al. [2005] with an implementation of an MST (Maximum Spanning Tree) parser for both projective and non-projective sentences.

The basic MST system is non-projective: it begins with a fully connected digraph between all nodes (tokens) in the sentence, and attempts to remove arcs until it finds the best-scoring directed tree that is spanning (e.g. connected to all nodes) and rooted in the *root* artifact. This system uses variants of the Chu-Liu-Edmonds algorithm [Chu and Liu, 1965, Edmonds, 1967] with a complexity of $O(n^2)$. The projective version, requiring the application of an additional constraint, has a complexity of $O(n^3)$. The projective graph-based system is very similar to PCFG-based systems described above, and uses variants of the Cocke-Kassami-Younger (CKY) algorithm [Younger, 1967]. As per Kübler et al. [2009, p.90]:

> [...] projective graph-based models are in fact just context-free dependency grammars where the underlying grammar generates all possible strings [...] and all possible trees [...]

Now, let us say we wish to parse the sentence in example 1.18 using the non-projective MST algorithm described in McDonald et al. [2005], Kübler et al. [2009].

Example 1.18 *Je mange la pomme ici.*
I eat the apple here = I'm eating the apple here



The graph based system begins with a fully-connected digraph shown in fig. 1.1. A set of features is then used to define a score for each directed arc. These can include the two tokens on either end of the arc as well as the sentence containing them. Thus, assuming the tokens have been pos-tagged and lemmatised prior to parsing, we can construct features out of their pos-tags, lemmas, lexical forms, relative or absolute positions, as well as any intervening tokens, etc. However, in this version of the algorithm where features are applied to individual dependency arcs (known as a 1st order system), we cannot construct features based on partial parses (e.g. the node's most likely head or left/right dependents).

Dependency labels are handled very simply as follows: for each directed arc from a governor G to a dependent D, we give a score to each label, and retain only the single highest scoring label. Labels are thus chosen independently of one another. The scoring mechanism

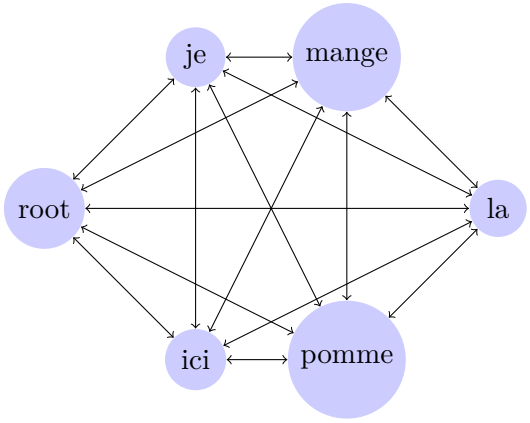


Figure 1.1: Graph-based MST parsing, step 1: fully connected digraph

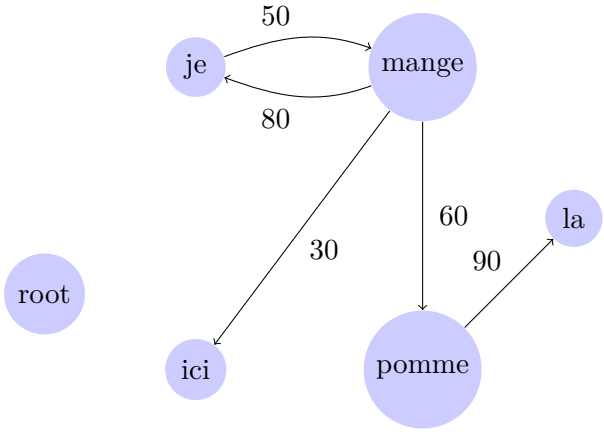


Figure 1.2: Graph-based MST parsing, step 2: maximum incoming arcs

itself is the responsibility of the machine learning algorithm, and will not be discussed in this chapter.

Next, we eliminate all arcs except for the highest-scoring incoming arc for each node, as shown in fig. 1.2. For the sake of illustration, we artificially assumed that the highest incoming arc for *mange* is *je*, with a score of 50, even if this is an unlikely result in any real system.

If this creates a cycle, the group of nodes contained in the cycle is treated as a single node, and we attempt to find the highest-scoring incoming arc for this entire group. In fig. 1.3, we have transformed the cycle between *je* and *mange* into a single node. The highest-scoring incoming arc into this single node now comes from *root*, with a score of 40.

Finally, we retain the highest-scoring incoming arc for the entire group, in this case *root* → *mange*, removing the previous incoming arc into the dependent *mange*. This gives us the final acyclic parse shown in fig. 1.4. We now have a maximum spanning tree for this sentence.

By starting the analysis from a fully-connected digraph, non-projective graph-based systems do not privilege short-distance relations over long-distance ones, except when the features indicate that this is an important factor, nor do they privilege projective solutions over non-projective solutions.

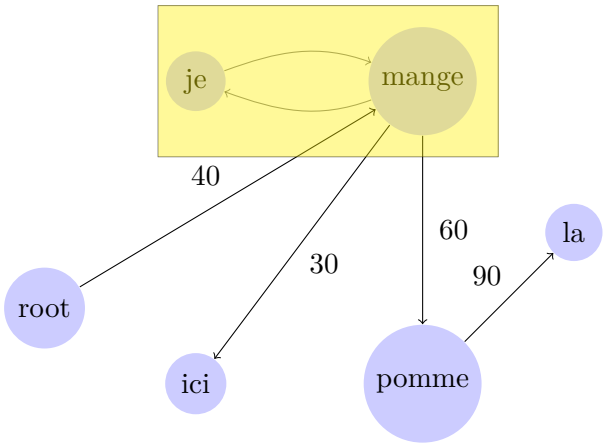


Figure 1.3: Graph-based MST parsing, step 3: cycles as nodes

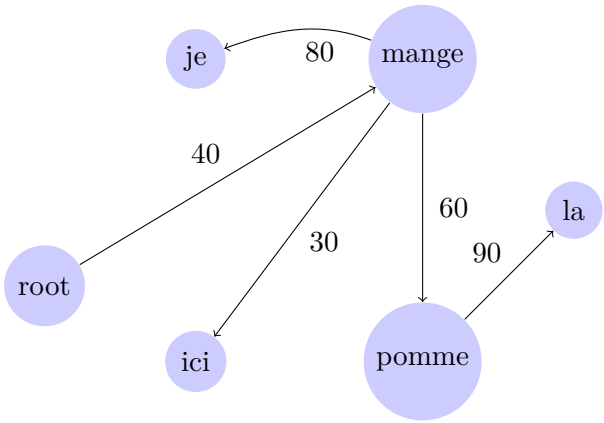


Figure 1.4: Graph-based MST parsing, step 4: final graph

Note that it is possible to use larger sub-graphs as the unitary scoring element of the MST algorithm. In the system described above, the unitary scoring element, for which features were calculated, was a single arc connecting 2 nodes. This is known as a 1st order system. In a 2nd order system, the unitary scoring element is a group of 3 nodes connected by 2 arcs. This allows us to take into account small partial tree structures, but at the cost of higher complexity. In the case of the 1st order model, we need to consider $(n-1)k$ outgoing arcs for each node, where n is the number of nodes and k is the number of labels, for a total of $n(n-1)k$ unitary scoring elements. In the 2nd order model, we need to consider $3n(n-1)(n-2)k^2$ partial trees, for the configurations $n_1 \rightarrow n_2 \rightarrow n_3$, $n_1 \rightarrow n_3 \rightarrow n_2$ and $n_2 \leftarrow n_1 \rightarrow n_3$. Moreover, we have to deal with statistical dispersion in the training corpus, as each partial tree will be much more rare. McDonald and Pereira [2006], show that the 2nd order algorithm is NP-hard, but that certain reasonable assumptions can be made to reduce the complexity to $O(n^3)$. Considerable work has been performed on graph-based systems for French by Mirroshandel et al. [2012]—this will be discussed further in the final chapter, section 7.3.1, page 198.

1.2.3 Transition-based parsers

By contrast to graph-based systems, which learn how to produce the tree structure directly from a graph, transition-based systems tackle the sentence indirectly, by attempting to learn a series of transitions that will enable the system to construct the tree one step at a time. This approach was explored in depth by Nivre et al. [2007b] in his freely available MaltParser. We will explore this paradigm in greater detail than the others, as it is the paradigm on which Talismane is based.

The basic data structure used for parsing can be represented as follows:

- σ: a stack, or ordered sequence of tokens which have been partially processed
- β: a buffer, or ordered sequence of tokens which have not yet been processed
- Δ: a set of dependency arcs of the form *label(governor, dependent)* that have already been added
- τ: a sequence of transitions allowing us to reach the current state from an initial state

A subscript is used to indicate the position of a token on the stack or buffer, where the σ_0 indicates the token most recently added to the stack (the "top" of the stack), always to the right of the remaining tokens on the stack, and β_0 indicates the next token on the buffer (the "head" of the buffer), always to the left of the remaining tokens on the buffer. By extension, σ_1 represents the token just below the top of the stack (notation extensible to any σ_n), and β_1 represents the token just after the head of the buffer (notation extensible to any β_n). At each step in the algorithm, we need to determine whether a dependency exists or not between σ_0 and β_0. We are said to "push" to the stack when we move the token at the head of the buffer to the top of the stack, and to "pop" the stack when we remove the token currently on top of the stack.

For a given sentence, the initial state is one where the stack contains only the *root* artifact, the buffer contains all of the sentence's tokens in their original order. A final state is one where the buffer is empty, and no more transitions are possible.

Moving from state to state is performed via a **transition system**: a set of allowable transitions from a shift-reduce style algorithm. In the simplest case, known as the *arc-standard* transition system, these include the transitions shown in table 1.6.

transition	**effect**	**precondition**
left-arc$_{\text{label}}$	Create the dependency arc *label*(β_0,σ_0) and pop the stack	σ_0 is not the *root* node.
right-arc$_{\text{label}}$	Create the dependency arc *label*(σ_0,β_0), remove the head of the buffer, and place the top of the stack at the head of the buffer	
shift	push the head of the buffer to the top of the stack	

Table 1.6: The arc-standard transition system for shift-reduce dependency parsing

We will now return to the sentence in example 1.18, and attempt to parse it using a transition-based parser. Table 1.7 shows a sequence of transitions that will parse this sentence correctly, as well as the stack, buffer, and dependency set after each transition.

	transition	**stack**	**buffer**	**dependencies added**
0		*root*	je, mange, la, pomme, ici	
1	shift	*root*, je	mange, la, pomme, ici	
2	left-arc$_{\text{suj}}$	*root*	mange, la, pomme, ici	suj(mange,je)
3	shift	*root*, mange	la, pomme, ici	
4	shift	*root*, mange, la	pomme, ici	
3	left-arc$_{\text{det}}$	*root*, mange	pomme, ici	det(pomme,la)
4	right-arc$_{\text{obj}}$	*root*	mange, ici	obj(mange,pomme)
5	shift	*root*, mange	ici	
6	right-arc$_{\text{mod}}$	*root*	mange	mod(mange,ici)
7	right-arc$_{\text{root}}$	$\emptyset$	*root*	root(*root*,mange)
8	shift	*root*	$\emptyset$	

Table 1.7: Sample transition sequence using the arc-standard shift-reduce transition system

Note that, unlike graph-based systems, not all word pairs are examined. For example, *je* and *pomme* are never compared to each other. More specifically, it has been shown [Nivre et al., 2007b] that, unlike graph-based systems, transition-based systems are only capable of generating projective dependency trees, and, in a given transition sequence, only word-pairs respecting this projectivity constraint will be examined for dependency. Furthermore, Nivre shows that the transition system above is capable of generating *all* projective dependency trees for this sentence.

Note also that when we first compare *root* and *mange* before step 3, we cannot yet generate the dependency *root(root, mange)* between them, since this would remove *mange* from the buffer before we attached its object *pomme*. We can only perform this attachment at step 7, after all of the right-hand dependents of *mange* have been attached. In other words, we can only attach a right-dependent to a governor after all of the dependent's own right-dependents have already been attached.

Other than this recursive restriction, the sequence of transitions to apply is not deterministic. A left-dependency can either be attached the first time the governor and dependent are compared, when building up the stack, or at any other time they are compared, as the stack is getting smaller again. The transition sequence in table 1.8 will also produce the correct parse, albeit with a different attachment order.

When training a machine learning system, we will be given a correctly annotated sentence, and will have to determine the correct transition sequence used to construct this parse. The algorithm used for this translation will have some say in the order in which dependencies are generated. In practice, however, these algorithms tend to attach a dependent as soon as possible.

Regarding the restriction on right-dependent attachment order, Nivre suggests that we replace the classical shift-reduce algorithm above with what he calls an *arg-eager* system. This system contains four transitions, as shown in table 1.9.

	transition	**stack**	**buffer**	**dependencies added**
0		*root*	je, mange, la, pomme, ici	
1	shift	*root*, je	mange, la, pomme, ici	
2	shift	*root*, je, mange	la, pomme, ici	
3	shift	*root*, je, mange, la	pomme, ici	
4	left-arc$_{\text{det}}$	*root*, je, mange	pomme, ici	det(pomme,la)
5	right-arc$_{\text{obj}}$	*root*, je	mange, ici	obj(mange,pomme)
6	shift	*root*, je, mange	ici	
7	right-arc$_{\text{mod}}$	*root*, je	mange	mod(mange,ici)
8	left-arc$_{\text{suj}}$	*root*	mange	suj(mange,je)
9	right-arc$_{\text{root}}$	$\emptyset$	*root*	root(*root*,mange)
10	shift	*root*	$\emptyset$	

Table 1.8: Alternative parse using the arc-standard shift-reduce transition system

transition	**effect**	**precondition**
left-arc$_{\text{label}}$	Create the dependency arc *label*(β_0,σ_0) and pop the stack	The reverse dependency *any*(σ_0,β_0) does not exist. σ_0 is not the *root* node.
right-arc$_{\text{label}}$	Create the dependency arc *label*(σ_0,β_0), and push the head of the buffer to the top of the stack	
reduce	pop the top of the stack	the head of the stack has a governor
shift	push the head of the buffer to the top of the stack	

Table 1.9: The arc-eager transition system for shift-reduce dependency parsing

This new transition system allows us to generate all dependencies the first time the governor/dependent pair is found at the σ_0/β_0 position (or vice versa), as shown in table 1.10. In fact, we *have* to create the dependency at this point, since, unlike the previous system, items previously pushed to the stack are never popped back onto the buffer. We thus have a deterministic transition system, with a guarantee that left dependencies will always precede right dependencies for a given governor, and that dependencies will be created from the inside out (the dependency closest to the governor is created before those farther away). This is the equivalent to a sequential left-to-right creation order (in terms of dependents, not governors), with the exception of multiple left dependents (e.g. "la grande pomme"), which are inversed. To illustrate this last point, we added the adjective *grande* to table 1.10: if we read the last word in each dependency arc - the sequence of dependents is identical to the original word sequence, with the exception of *la* and *grande*. Note that the burden of determining whether a word has any more right-dependencies has now been shifted from the `right-arc` transition to the `reduce` transition.

Let us now look at the features, i.e. the information that can be brought to bear on

	transition	**stack**	**buffer**	**dependencies added**
0		*root*	je, mange, la, grande, pomme, ici	
1	shift	*root*, je	mange, la, grande, pomme, ici	
2	left-arc$_{\text{suj}}$	*root*	mange, la, grande, pomme, ici	suj(mange,je)
3	right-arc$_{\text{root}}$	*root*, mange	la, grande, pomme, ici	root(*root*,mange)
4	shift	*root*, mange, la	grande, pomme, ici	
5	shift	*root*, mange, la, grande	pomme, ici	
6	left-arc$_{\text{mod}}$	*root*, mange, la	pomme, ici	mod(pomme,grande)
7	left-arc$_{\text{det}}$	*root*, mange	pomme, ici	det(pomme,la)
8	right-arc$_{\text{obj}}$	*root*, mange, pomme	ici	obj(mange,pomme)
9	reduce	*root*, mange	ici	
10	right-arc$_{\text{mod}}$	*root*, mange, ici	$\emptyset$	mod(mange,ici)

Table 1.10: Deterministic parse using an arc-eager shift-reduce transition system

each parsing decision. In graph-based systems, we were limited to features concerning the token itself and the sentence containing it. In transition-based systems, we can also draw information from the partial dependency tree that has already been constructed and, if we thought it might be useful, from the sequence of transitions already applied. Since we know the order of dependency creation (mostly left-to-right for the arc-eager transition system), it is fairly straightforward to imagine features that translate linguistic knowledge into partial tree structures. We can write features that take into account a potential governor's governor, or its current left and right-hand dependents, as well as the dependency labels assigned to all of these various dependencies. This opens the door to a whole slew of rich features aimed at constructing complex syntactic structures one step at a time.

We return to our initial objective: to maximize the amount of linguistic knowledge that can be injected into the system without sacrificing robustness. Transition-based parsers are robust, in that they are linear in complexity (analysis time component of robustness) and empirical (no *a priori* concept of grammaticality). Furthermore, they allow the injection of considerable linguistic knowledge, in the form of features, when compared to graph-based parsers.

On the other hand, the linear order of comparison means that short-distance dependencies are always examined before longer-distance dependencies, and in practice such systems tend to generate shorter-distance dependencies at the expense of longer ones. Furthermore, certain word pairs are never examined for a relationship, because earlier decisions would make an arc between these word pairs non-projective.

We are now in a position to summarise some of the main differences between graph-based and transition-based parsers, shown in table 1.11.

The Talismane syntax analyser described in this thesis is directly inspired by Nivre's work on parsing, but places it organically within the context of full syntax analysis, from raw text to parse trees, extends it from a purely deterministic system to a beam-search approach, and

Parser type	**transition-based**	**graph-based**
Complexity	$O(n)$	$O(n^2)$
Projective	Yes	No
Token features	Yes	Yes
Sentence features	Yes	Yes
Partial tree features	Yes	No
Dependency distance preference	Short	Neutral
Examines all word pairs	No	Yes

Table 1.11: Principle differences between transition-based and graph-based parsers

introduces a rich syntax for feature and rule description. These differences will all be covered in more detail in chapter 3.

1.3 Discussion

We have attempted in this chapter to examine some of the more interesting questions in the dependency annotation of French, and to explain the annotation choices made, with the objective of correctly annotating the syntactic structures if possible, and, if not, at least unambiguously identifying certain rare structures.

The second half of the chapter discussed and compared various parsing algorithms and the types of features that can be used in these algorithms, especially the shift-reduce transition-based parser with which the rest of this thesis is concerned. At one extreme, we have rationalistic parsers, allowing the injection of a huge amount of linguistic knowledge, but at the expense of less robustness and maintainability. On the other extreme, we have graph-based algorithms, which tend to make few simplifying assumptions about possible sentence structures, but allow the incorporation of minimal linguistic knowledge. The choice of a transition-based parser for Talismane allows us to inject more linguistic knowledge in the form of features covering partially constructed syntactic structures, all the while remaining robust in terms of both time performance and grammaticality assumptions.

Placing a detailed discussion of annotation at the start of this thesis is not an accident: it is a conscious effort to place the actual annotation at the heart of the thesis, rather than purely statistical measurements. We will attempt, in chapter 6, to use our understanding of both annotation difficulties and parsing algorithms to try to design specific features that are well-adapted to tackle specific linguistic phenomena within the framework of transition-based parsing. This will lead us to both a qualitative and a quantitative analysis of these phenomena within our training and evaluation corpora. But first, we need to gain an understanding of the supervised machine learning algorithms that handle these features.

Chapter 2

Supervised machine learning for NLP classification problems

In the previous chapter we covered syntax dependency annotation for French, and the algorithms capable of generating such annotations. However, these algorithms were assumed to make attachment decisions based on an oracle which somehow always knows the correct answer.

This chapter replaces the oracle with a supervised machine learning system, which learns which answers are the most probable by studying a manually annotated corpus. Using postagging as a case study, it explains various concepts behind supervised machine learning, including features, training and analysis algorithms and probabilistic classifiers. The main purpose of the chapter is to understand the mechanics of a supervised machine learning system sufficiently to enable a linguist to inject linguistic knowledge into the system in a way useful to the internal decision-making process.

Finally, the chapter gives an overview of the three major supervised machine learning projects in which I was involved: the PAN shared task for authorship attribution, the Jochre OCR system for Yiddish and Occitan, and the Talismane syntax analyser central to the present thesis.

2.1 Preliminary definitions

Supervised machine learning can be seen as a process whereby a machine learns how to make decisions enabling it to automatically produce similar annotations to those in a set of reference data, generally produced manually by a human. Mohri et al. [2012] define **machine learning** as follows:

> Machine learning can be broadly defined as computational methods using experience to improve performance or to make accurate predictions. Here, *experience* refers to the past information available to the learner, which typically takes the form of electronic data collected and made available for analysis. This data could be in the form of digitized human-labeled training sets, or other types of information obtained via interaction with the environment. In all cases, its quality and size are crucial to the success of the predictions made by the learner.

Mohri et al. [2012] then specify the **supervised** learning scenario as follows (emphasis mine):

> The learner receives a set of *labeled* examples as training data and makes predictions for all unseen points.

The goal in supervised machine learning is thus to make predictions for unseen data that imitate the labels used in the training set. The most typical scenario is **classification**, where the labels are categories, drawn from a closed set of nominal items, and the predictions apply the same set of categories to unseen data.

Supervised learning is defined in contrast to the *unsupervised* scenario, where the training examples are *unlabeled* data points. In unsupervised learning, the most typical scenario is **clustering**, where we attempt to group the unlabeled training data into *clusters*, each of which contains data points with similar characteristics.

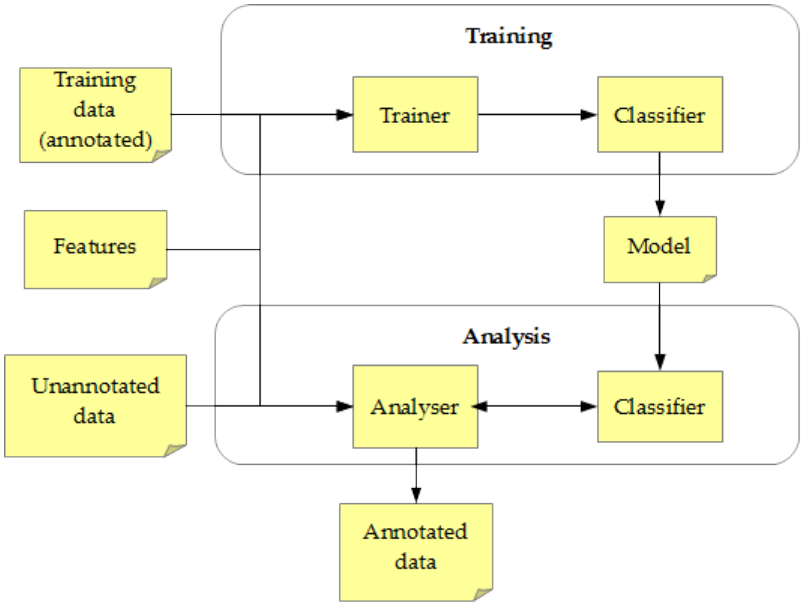


Figure 2.1: Classification through supervised machine learning

Figure 2.1 shows a schematic diagram for classification through supervised machine learning. In the **training** phase, a software module which we call the **trainer** constructs a model from annotated data, known as the training set. In addition to the annotated data itself, the trainer requires a set of features used to describe this data. This feature set enables the trainer to transform the data into a format readable by **classifier**, which is an interchangeable software module directly responsible for constructing a **model** that can be used to analyse unannotated data. Each classifier will have its own type of model, with a particular internal structure suitable for this classifer, which is not interchangeable with another classifier's model. When constructing the model, the classifier attempts to detect regularities in the training data which can be generalised to any set of data being analysed. In the **analysis** phase, performed by a software module known as the **analyser**, unannotated data is automatically annotated by the system. The analyser first takes the unannotated data, and converts it into a format readable by the classifier using the same feature set that was used for training.

The classifier then uses the previously constructed model to make individual classification decisions, which are applied by the analyser to produce the final annotated data.

Throughout this chapter, we will illustrate the concepts presented through the case study of a pos-tagger, responsible for learning how to assign part-of-speech tags to words in a raw corpus.

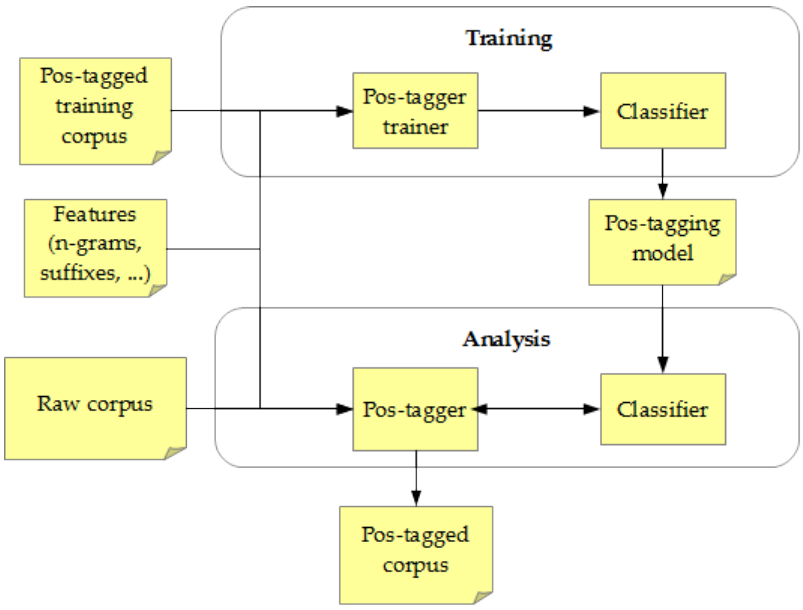


Figure 2.2: Pos-tagging through supervised machine learning

Figure 2.2 shows the same diagram as before, but instantiated for the case of pos-tagging. The training corpus is one in which each word has been annotated with a pos-tag (either manually, or automatically at first, and then manually corrected). The features used are information useful for the pos-tagging task, such as the suffix of a word. These will be discussed in far more detail in section 2.4 below. The pos-tagger trainer takes the pos-tagged corpus, and uses the features to transform it into a format usable by the classifier. The classifier in turn builds a model specific to this set of features, and to the pos-tagging task. For example, the model could contain a set of rules, e.g. "If the word ends with *-ion*, than it is a common noun." With this model constructed, we are now ready to ask the system to add pos-tags to a raw corpus. When analysing the raw corpus, the pos-tagger uses the same set of features transform the raw data into a format usable by the classifier. The classifier, with the help of the previously constructed model, tells the pos-tagger which pos-tag to assign to each word in the corpus, thus producing the final pos-tagged corpus.

In order to judge the quality of the model constructed, we often perform another phase, **evaluation**, as shown in fig. 2.3. In this phase, the analyser is applied to previously annotated data (the test set) rather than unannotated data. In order to ensure that our system performs well on unseen data, the test set is completely separate from the training set. Another software module, the **evaluator**, measures the system's **accuracy** by comparing the automatic annotations produced to the original manual annotations in the test set, and produces an evaluation report containing various evaluation measures. More detailed definitions for evaluation measures are given in section 2.7. We can also measure the system's **efficiency**:

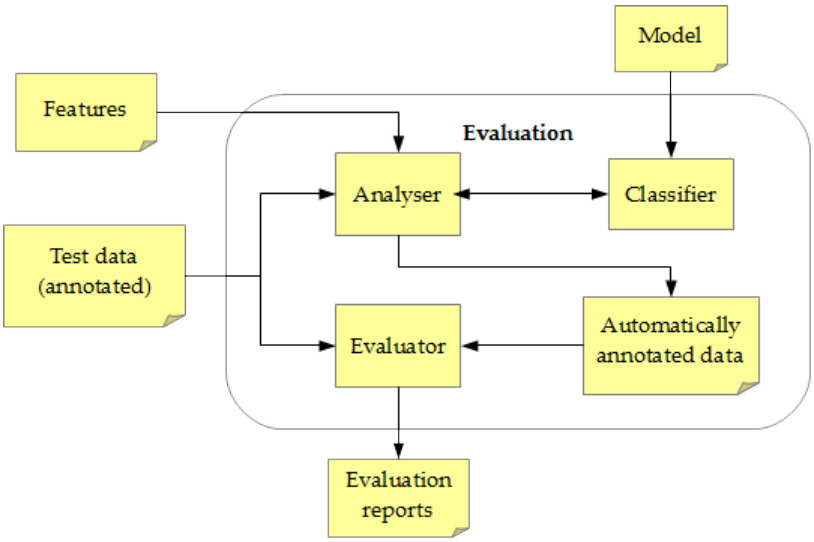


Figure 2.3: Supervised machine learning evaluation

the speed at which it can process data, regardless of the accuracy.

We will now turn to a more technical definition for supervised machine learning found in Murphy [2012]:

> In the **supervised learning** approach, the goal is to learn a mapping from inputs x to outputs y, given a labeled set of input-output pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$. Here $\mathcal{D}$ is called the training set, and N is the number of training examples.

We will call the set of inputs $\mathcal{X}$ and the set of outputs $\mathcal{Y}$. In the learning systems described in the present thesis, each $x \in \mathcal{X}$ is a **linguistic context** and each $y \in \mathcal{Y}$ is a nominal label (or category). Each (x, y) pair is known as a single **training example**. For our pos-tagging case study, we instantiate this definition as follows:

- linguistic context x: a single token within a sentence (described in more detail in section 2.3 below)
- label y: a label from the *tagset* defined in table 1.1 (e.g. NC, V, ADJ)
- training set $\mathcal{D}$: an annotated corpus, or a collection of sentences within which each token has been assigned a single pos-tag by a human annotator, with a total of N tokens

The goal of machine learning is to learn some sort of function of the form:

$$\phi : \mathcal{X} \rightarrow \mathcal{Y} \tag{2.1}$$

Various methods for defining the function ϕ and for estimating its associated parameters are described in section 2.8 below.

2.2 Annotation

In the present thesis, our training set is a manually annotated corpus of text, known as the **training corpus**. Any portions of the annotated corpus set aside for testing are known as the **test corpus** or **evaluation corpus**. The actual training and evaluation corpora used in our thesis will be described in chapter 4. In this section, we consider this annotated corpus from the perspective of training and evaluation in a machine learning system. It is assumed that this corpus is representative of the sub-language we wish to be capable of analysing, so that inferences may be drawn which will allow our machine learning system to correctly annotate unknown text. We use the term "sub-language", because it is naive to assume that a training corpus can help us correctly annotate any variant of a given language, be it edited (published material), unedited (e.g. blogs, wikis), or an oral transcription, and extracted from different domains: journalistic, technical, literary, etc. Thus, the training corpus can be viewed as a sample from some larger population, and, following Sinclair [2005], we would like to assume that the corpus has been selected using external criteria to be as representative and balanced as possible with respect to the sub-language in question.

In reality, however, because of the immense effort required to produce annotated corpora and the difficulty of obtaining large bodies of freely redistributable text, we often have little control over the sub-language being represented. Indeed, all of the systems described in the present thesis were trained using the French Treebank [Abeillé et al., 2003], a corpus of journalistic text drawn from a single newspaper (Le Monde) over the period of seven years, and yet attempts are made to evaluate these systems against a variety of genres and domains. This thesis will describe certain basic methods for attempting to generalise beyond the training corpus, including feature generalisation through the use of external resources (see section 4.3.1) and feature selection by means of a cutoff (see section 4.4.1). However, when analysing text clearly belonging to another sub-language, we can attempt to improve our inferences via more advanced methods for domain and genre adaptation, which will be described in more detail in section 7.3.1.

2.3 Linguistic Context

In our discussion of supervised machine learning, we introduce a concept we will call the **linguistic context**, which is the precise definition of the linguistic entity being considered for each classification decision. In our pos-tagging case study, we have chosen to design a left-to-right sequential pos-tagger, which analyses one token at a time from the start of the sentence to its end. Whereas a simple definition of pos-tagging might state that we want to assign a pos-tag to each token in a sentence, this definition fails to take into account the sequential nature of the pos-tagger, and the fact that we already know the pos-tags assigned to all previous tokens in the sentence. We thus define our linguistic context as the pair (*token, history*), where *token* is the next token to be analysed in the sentence, including information about the lexical form of the token, its index within the sentence, and the lexical forms of all other tokens in the sentence, and *history* is a list of pos-tags already assigned to previous tokens in the sentence.

A precisely defined and delimited linguistic context enables the linguist to understand the information available at any given moment in the task, on which the features can be based. This brings us to an initial discussion of features.

2.4 Features

In Mohri et al. [2012], features are defined as:

> The set of attributes, often represented as a vector, associated to an example.

We replace the term *example* above with our own term, *linguistic context*, defined in section 2.3. Thus, we will define **features** as:

> The set of attributes associated to a given linguistic context. In a supervised machine learning system, these attributes are the sole representation of the linguistic context available to the classifier.

In a biological study, these attributes might be the height, weight, age and sex of a person. In other words, as far as the study is concerned, the person is reduced to a vector containing exactly these four attributes.

These features can be *boolean* (providing a true or false answer), e.g. `ends-with-ion`: "Does the current token end with the characters *-ion*?" They can be numeric, e.g. `length`: "What is the length of the current token?" They can return a textual result (known as a *string*): e.g. `suffix3`: "What are the last three letters of the current token?".

In this thesis, we will use the term **feature** to refer to the type of information that is being requested, and **feature result** to refer to the actual information extracted from a given context. The **feature set** $\mathcal{F}$ is simply an ordered set of features. The **feature vector** is thus a list of *feature results*, each of which corresponds in position to a particular feature f in the ordered set $\mathcal{F}$. For example, let us say our feature set is comprised of the three features described in the previous paragraph (`ends-with-ion`, `length`, `suffix3`). For the context "*bavardage*" (associated with any history of previously assigned pos-tags), the corresponding *feature vector* will be (false, 9, "age").

In order to be useful, features must satisfy the following two criteria:

- predictive: a feature must provide information that can help the system select the correct label to apply. In other words, different feature results should generally be coupled with different labels.
- generalisable: a feature should generalise well to other corpora. In other words, if a feature result is associated with a given label in the training corpus, it should generally be associated with the same label in other corpora.

Selecting features that are both predictive and generalisable requires linguistic intuition, and is one of the main ways in which the linguist interacts with the machine learning system. For example, when examining words with various parts of speech in French, our initial intuition is that the word's suffix can help us predict the part of speech. Indeed, words ending in "ion" (e.g. *éducation*, *situation*) are more likely to be common nouns, while words ending in "ait" (e.g. *mangeait*, *parlait*) are more likely to be verbs in the imperfect tense. This leads us to define the `suffix3` feature above, since the last 3 letters seem sufficient in many cases. We can assume our `suffix3` feature is generalisable as well: words ending with "ion" are likely to be common nouns regardless of the corpus genre or domain. Now, it may well be better to define a specific list of suffixes, rather than taking all possible 3-letter suffixes: thus, the actual feature corresponding to this intuition can take on several forms, and sometimes only trial and error allow us to select the best one.

Let us take another example: our linguistic intuition tells us that certain sequences of parts-of-speech are more probable than others. For example, in the training corpus, we can easily notice that determiners are often followed by nouns. This is known as an **n-gram**, a sequences of length n for a particular linguistic entity type, in our case the pos-tag itself. Indeed n-gram models of language, despite their limitations (short-distance sequential coverage only), can be highly informative when sufficient data is available. This information is transformed into a predictive feature by using the $n-1$ preceding pos-tags to try to predict the nth postag. Thus, in the case of a 2-gram (or bigram) feature, knowing that the previous token's pos-tag is `DET` (determiner) helps predict that the current token's pos-tag is likely to be `NC` (noun) as opposed to `V` (an indicative verb). Similarly, the 3-gram (or trigram) feature tells us that, if the previous two pos-tags were (`DET`, `ADJ`), the current pos-tag is likely to be a `NC` and very unlikely to be another `DET`. Now, we may wish to define another feature based on the trigram observation that a number preceded by an determiner and followed by a noun is systematically tagged as an adjective (e.g. *"les 3 mousquetaires"*). However, our linguistic context (*token, history*) is missing information required to construct this feature: the system can check if the previous token was tagged as a determiner, and that the current token is a number, but it has no idea what the next token's pos-tag will be (one option here is to use an external lexicon, and check if the next token is listed as a noun in this lexicon).

Note that although the linguist may feel that the features he defined are likely to be predictive, it is up to the classifier to determine which features are actually most important in a given situation. This is particularly important because features can give contradictory results. For example, in the sentence *"Le lait est bon"*, when analysing the word *lait*, the `suffix3` feature gives a high probability for a verb, whereas the `bigram` feature gives a high probability for a noun, and only the classifier can decide between the two, based on the evidence found in the training corpus.

2.5 Training

The purpose of training is to take an annotated training corpus, and generate a predictive model which captures regularities in the training data and can be used to make decisions on unknown data.

The training algorithm for pos-tagging first needs to transform the annotated training corpus into a series of N contexts of the form (*token, history*), and then to convert them into N *training examples* of the form (x, y), where x is a feature vector representing a single context, and y is the pos-tag associated with this context.

The algorithm makes use of a classifier, whose responsibility it is to review the full sequence of training examples, and learn how to translate feature vectors into labels, automatically assessing the relative importance of various features or feature combinations. The actual function ϕ mapping a feature vector x to a label y can take on various forms, described in more details in section 2.8. After training, the classifier has constructed what is known as the **model**, which contains the values of the parameters used to replace various unknowns in the function ϕ. The set of parameters depends entirely on the classifier type, but the goal of training is to attempt to maximise the accuracy provided by these parameters for the full set of training examples in the training corpus. This accuracy is measured by using the model to predict the label y associated with each linguistic context x, and to compare the guessed label with the correct label.

We are not yet concerned in the present section with the internals of the model. As far as we're concerned, the classifier is simply a black box, with three interfaces:

- feed training example
  - inputs: feature vector for context x, correct label y
  - output: none
- calculate model (to be called after all training examples have been fed)
  - inputs: none
  - output: model α
- predict
  - inputs: feature vector for context x, model α
  - output: guessed label y

```
   Input: training corpus reader, feature set F, classifier
   Output: statistical model
 1 while training corpus reader has more sentences do
 2     S ← next sentence in reader ;                    // a list of pos-tagged tokens
 3     history ← empty list;
 4     foreach (token, postag) in S do
 5         featureVector ← empty list;
 6         foreach feature in F do
 7             featureResult ← apply feature to (token, history);
 8             add featureResult to featureVector;
 9         end
10         feed training example (featureVector, postag) to classifier;
11         add (token, postag) to history;
12     end
13 end
14 return classifier.calculateModel();
```

Algorithm 2.1: Basic pos-tagging training algorithm

A training algorithm for pos-tagging, using the classifier interface described above, is shown in algorithm 2.1. We tackle the training set one sentence at a time, using a "training corpus reader" whose responsibility it is to read token/pos-tag pairs in whatever format the training corpus was encoded in. The system next converts each context in the sentence into a feature vector on lines 5 to 9. More specifically, each feature in the feature set is applied to the context (*token, history*) on line 7, resulting in a *featureResult* which is then added to a specific position corresponding to this feature in the feature vector. On line 10, the training example (i.e. the feature vector and the resulting pos-tag) is fed to the classifier. For now, the classifier simply stores this training example—it will only be used later, when constructing the statistical model. On line 11, the pos-tagged token just processed is added to the history, after which we tackle the next pos-tagged token in the sentence. Thus, the *history* variable is

empty for the first token, it contains $\{(token_1, postag_1)\}$ for the 2nd token, $\{(token_1, postag_1), (token_2, postag_2)\}$ for the 3rd token, and so on. After all sentences have been processed, the classifier is asked to do its internal magic, and calculate the statistical model based on the full set of training examples that have been fed to it. Later, when analysing unknown data, this model will allow the classifier to take a feature vector as input, and return the best label (in our case, the best pos-tag) for this feature vector as output.

2.6 Analysis

After training has completed, the system has generated a model, and can analyse unknown data using this model, on the condition that we use an identical feature set to describe the unknown contexts as was used for the training contexts.

```
   Input: tokenised sentence S, feature set F, classifier
   Output: pos-tagged sentence
 1 history ← empty list;
 2 foreach token in S do
 3 |   featureVector ← empty list ;
 4 |   foreach feature in F do
 5 |   |   featureResult ← apply feature to (token, history);
 6 |   |   add featureResult to featureVector;
 7 |   end
 8 |   postag ← classifier.predictfeatureVector);
 9 |   add (token, postag) to history;
10 end
11 return history;
```

Algorithm 2.2: Basic pos-tagging analysis algorithm

An analysis algorithm for pos-tagging is shown in algorithm 2.2. The algorithm is similar in many ways to the training algorithm, except that an input sentence is defined as a series of tokens, rather than a series of pos-tagged tokens. On lines 3 to 7, the linguistic context (*token*, *history*) is converted into a feature vector, exactly as it was in the training algorithm, except that the history consists in our current list of best pos-tag guesses, instead of the pos-tags read from a training corpus. On line 8, the classifier takes the feature vector, and, based on the information found in the model, returns the most likely pos-tag for this feature vector. We then add the guessed token/pos-tag pair to the history, and continue our analysis with the next token to be classified.

2.6.1 Pruning via a beam search

In the algorithm described above, we always selected the single most likely pos-tag before continuing the analysis. This type of system is known as a *greedy search*, since the system immediately accepts the best solution provided at each analysis step. In practice, this means that the system has no way of correcting a locally probable error (e.g. in a "garden path" sentence). For example, in the sentence *"la prison ferme ses portes"* ("The prison is closing its doors"), the locally most probable label for the token *ferme* could well be an adjective,

1
la
DET 70 %
CLO 29
NC 1 %

2	
la	prison
DET	**NC** 95 %
CLO	NC 5 %

3		
la	prison	ferme
DET	NC	ADJ 55 %
DET	**NC**	**V** 50 %
DET	NC	NC 4 %
CLO	NC	ADJ 1 %

4			
la	prison	ferme	ses
DET	**NC**	**V**	**DET** 85 %
DET	NC	ADJ	DET 15 %

5				
la	prison	ferme	ses	portes
DET	**NC**	**V**	**DET**	**NC** 87 %
DET	NC	ADJ	DET	NC 10 %
DET	NC	V	DET	V 2 %
DET	NC	ADJ	DET	V 1 %

Table 2.1: Beam search example—correct solution in bold

reflecting the expression *"la prison ferme"* ("prison sentence without parole"), and the system would have no way of correcting this label to a verb as the analysis continues.

Now, let us assume that our classifier is a *probabilistic* classifier, which produces a probability distribution of pos-tags instead of the single most likely pos-tag.

In other words, the classifier's `predict` interface is now redefined as follows:

- predict
  - inputs: feature vector for context x, model α
  - output: set of labels y, each associated with a probability $prob$, where $\sum prob = 1$.

An alternative method can now be to keep the full probability distribution of pos-tags for each token, and construct multiple contending analyses in parallel, retaining only the most probable one at the end. However, this solution requires us to compare an exponential number of solutions. Indeed, if a sentence contains n ambiguous tokens with 2 possible pos-tags each, the number of solutions to be compared is 2^n. To keep such a system tractable, dynamic programming is required using lattice structures [Viterbi, 1967], but this limits the types of features we can apply and, in particular, we can only use n-gram features for some small n when analysing the history, and cannot look farther afield than this n when trying to make specific decisions.

Thus, some sort of method is required to "prune" the search space and keep only the most reasonable hypotheses at each step. In this thesis, we make heavy use of a middle-ground approach, the **beam search** [Bisiani, 1987], or breadth-first search. This algorithm retains only the k most probable analyses after each step of analysis, thus maintaining a linear time analysis, where k is known as the *beam width*.

Table 2.1 shows how a beam search might be applied to the sentence *"la prison ferme ses portes"* with a beam width of 2. The correct solution at each step is shown in bold. When analysing the first token, *"la"*, the system finds 3 possibilities: `DET`, `CLO` and `NC` (the musical note). Because of the beam width of 2, the 3rd possibility is discarded. When analysing the 3rd token, *"ferme"* the wrong choice, `ADJ`, is placed on top of the beam. Thus, without a beam search (or equivalently, with a beam width of 1), the correct analysis `V` would be discarded.

When we reach the fourth token *"ses"*, the correct analysis for *"ferme"* has been restored to the top, presumably due to n-gram features in the statistical model, thus permitting the system to place the correct analysis on the top of the beam for the entire sentence.

Note that placing the wrong solution on top at step 3 assumes our system has no forward-looking features, or that the forward-looking features' weights are insufficient to place the correct solution on top. Real systems could well contain a combination of backward- and forward-looking features. The former have more information to draw from (since we know the previously assigned pos-tags), but this doesn't stop us from looking at the word forms of tokens to the right of the current token, and checking their possible parts-of-speech in an external lexical resource.

When using a beam search, we are searching through a very small subset of the full possible search space. As stated before, if a sentence contains n ambiguous tokens each with 2 possible pos-tags, we would need a beam of width 2^n to retain all of the ambiguities. Much narrower beams are used in practice, which retain only the most ambiguous cases as the sentence gets analysed.

```
   Input: tokenised sentence S, feature set F, probabilistic classifier, beam width k
   Output: pos-tagged sentence
 1 beam ← empty heap;                          // ordered by decreasing probability
 2 history ← empty list;
 3 add history to beam;                                  // prime the initial beam
 4 foreach token in S do
 5     i ← 0;
 6     nextBeam ← empty heap;                    // create beam for current token
 7     while beam is not empty AND i<k do
 8         history ← pop history from top of beam;     // next most likely analysis
 9         featureVector ← empty list ;
10         foreach feature in F do
11             featureResult ← apply feature to (token, history);
12             add featureResult to featureVector;
13         end
14         postagsWithProbs ← classifier.predict(featureVector);
15         foreach (postag, probability) in postagsWithProbs do
16             newAnalysis ← copy of history;
17             add (token, postag, probability) to newAnalysis;
18             add newAnalysis to nextBeam;                  // add analysis to beam
19         end
20         i ← i + 1;
21     end
22     beam ← nextBeam;
23 end
24 return history on top of final beam;
```

Algorithm 2.3: Pos-tagging analysis algorithm with beam search

In algorithm 2.3, we introduce a beam search into the pos-tagging analysis algorithm. To this end, as we analyse each token in the sentence, we create a heap structure (line 6), which

is defined to automatically order the analyses by decreasing probability, where the probability for a given analysis is assumed to be the product of the probabilities for component pos-tags. In the loop beginning on line 7, we read only the k most probable analyses from the previous token's heap, where k is the beam width. For each of these k analyses, we analyse all of the features against the context (*token*, *history*), where *history* is the next most probable analysis currently on top of the beam. Since, in practice, feature analysis is generally the most time-consuming aspect of the algorithm, this makes a beam search effectively k times slower than a deterministic algorithm. In line 14, we return a full probability distribution of postags rather than the single most probable pos-tag. We are now in a position to add each pos-tag in the distribution to the current token's beam, which in turn will automatically place the most probable analyses on top. The algorithm completes when there are no more tokens to analyse, and the most probable analysis on the final beam is then returned.

2.7 Evaluation

One critical aspect of supervised machine learning is evaluation, in which we attempt to assess the accuracy of the predictions applied by the system using a model trained with a certain configuration.

The term **accuracy** refers to the percentage of correct labels out of the total label count. Sometimes, we are also interested in performance improvements for a specific labal y. For a given label y, we can thus measure the precision and recall, where **precision** indicates, for all of the contexts automatically labeled with y, the percentage where y was the correct label in the test corpus, and **recall** indicates, for all of the contexts where y is the correct label in the test corpus, the percentage of that was automatically labeled with y.

More specifically, we call a *true positive* a context that is labeled with y both in the test corpus and in the automatically annotated data (the correct result: the shepherd cried wolf and there really was a wolf). We call a *false positive* a context that was automatically labeled with y, when the expected label in the test corpus was y' (additional wrong result for y: the shepherd cried wolf but there wasn't a wolf). We call a *false negative* a context was automatically labeled with y', when the expected label in the test corpus was y (missing result for y: the shepherd didn't cry wolf, but there was a wolf). For a given label y, let tp be the total count of true positives, fp be the total count of false positives, and fn be the total count of false negatives. Then:

$$precision = \frac{tp}{tp + fp} \tag{2.2}$$

$$recall = \frac{tp}{tp + fn} \tag{2.3}$$

The **f-score** (or f-measure) for y is the harmonic mean of precision and recall, defined as follows:

$$fScore = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.4}$$

When training a system, there is always a danger of **over-fitting** the model to the training set or to a particular test set. This is the case where the model describes the data set too precisely, and assumes that phenomena in the set are linguistic regularities whereas in fact they are anomalies. For example, a pos-tagging training set might contain the word "lead"

as a noun only (the metal), and never as a verb. If our only training feature is the word form itself, we will assume that any occurrence of the word "lead" is a noun. Thus, over-fitting is often directly related to the feature set. Over-fitting is always a danger with the training set, but becomes a danger with a test set as we iteratively refine the feature set to get better test results.

In order to avoid over-fitting to a particular test set, the manually annoated corpus is often divided into three distinct parts:

1. **Training corpus:** the portion of the annotated corpus (typically 80%) set aside to train the system.

2. **Development corpus:** a portion of the annotated corpus (typically 10%) set aside to tune the training configuration. We are permitted to explore this corpus to our heart's content as we attempt to improve our system's accuracy.

3. **Test corpus:** a portion of the annotated corpus (typically 10%) set aside to evaluate the system. The goal is to completely avoid examining this corpus so that the evaluation remains an accurate measurement of accuracy on unknown data.

In practice, however, manually annotated corpora are often very expensive to produce, meaning that the resulting development and test corpora are too small to generalise reliably. In this case, we often use an evaluation technique known as **cross-validation**, where the corpus is divided into n equal portions, and we perform n evaluations, leaving out each time 1 of the portions for evaluation, and training on the remaining $n-1$ portions. Our system's accuracy is then assumed to be the average accuracy of the n evaluations, and the standard deviation gives a rough indicator of the system's ability to generalise (or, more often, of the training corpus' homogeneity).

2.8 Classifiers

The **classifier** is the element which infers a model from a set of training examples, and then uses this model to predict labels for arbitrary feature vectors. The precise nature of the model—meaning its internal structure and the way in which this structure is used—is determined by the classifier.

Up to now, the classifier was seen as a black box with an interface that can be used by the training and analysis algorithms, defining internally some sort of function $\phi : \mathcal{X} \rightarrow \mathcal{Y}$. We now peek into the black box to see how the classification algorithms actually work. Classifiers come in many flavours—indeed, the open source Weka data mining software[1] defines over 100 classifiers, including both symbolic classifiers (e.g. decision trees, defining a sequence of decision making rules) and numeric ones (e.g. classifiers based on linear equations making use of a matrix of calculated feature weights)

We have three requirements for our classifier:

- it must be able to handle large numbers of features (hundreds of thousands) and a large number of training examples (several million).

[1] `http://www.cs.waikato.ac.nz/ml/weka/`

- although training can be slow, it must be able to predict results for a given feature vector and model very quickly

- because of our heavy reliance on beam search techniques (see section 2.6.1), it must return a probability distribution for the different labels, rather than indicating the single best label.

We are thus interested in **probabilistic classifiers** only, defined as classifiers which, rather than mapping $\mathcal{X} \rightarrow \mathcal{Y}$, takes on the following form:

$$\phi : \mathcal{X} \rightarrow P(\mathcal{Y}|\mathcal{X}) \tag{2.5}$$

We have selected three well known robust classification algorithms for comparison: the perceptron algorithm, the log-linear or maximum entropy algorithm, and the linear SVM algorithm. In cases where the default version of the algorithm does not meet the probabilistic criterion, we present a version capable of presenting the results as a probability distribution.

These three classifiers are all "numeric" in that they all require us to convert features into numeric values, typically normalised to values between 0 and 1, so that, formally, each context x is transformed into a $|\mathcal{F}|$-dimensional feature vector $(f_1, f_2, \ldots, f_n)$, where each feature is represented as a value from 0 to 1. We thus have a group of functions f in $\mathcal{F}$ such that:

$$f : X \rightarrow [0, 1] \tag{2.6}$$

Non-numeric string and boolean features are converted to numeric features using the methods described below in section 2.8.1.

The insight behind numeric classifiers is that they enable us to solve the classification problem mathematically, by constructing some form of equation around the feature vector, and optimising the parameters (or unknowns) in this equation in a way that links features to the correct labels.

2.8.1 Converting non-numeric features to numeric values

Since we are dealing with numeric classifiers, we have to somehow transform our non-numeric boolean and string features into numeric values. This is achieved by transforming them into multiple numeric features, as follows:

- Boolean features are converted into two separate features, one of which has a result of 1 and the other a result of 0. For example, the `ends-with-ion` feature described above will be converted into two numeric features: $\texttt{ends-with-ion}_{\text{true}}$ and $\texttt{ends-with-ion}_{\text{false}}$. If our feature set is comprised of these two features, then for the token "*education*", the resulting feature vector will be (1,0), whereas for "*bavardage*" it will be (0,1). This may seem unintuitive: why duplicate the information? Is it not sufficient to generate a single feature with they values 1 and 0 to represent the boolean `true` and `false`? However, both the perceptron and the maxent algorithm handle a value of 0 exactly in the same way as a non-existent feature result (it has a null effect on the model being constructed), and we are forced to add a `false` feature if we want the "false" results to be taken into account explicitly as potentially predictive information.

- String features are treated as a closed set of categories resulting in multiple features, one per string result found in the training corpus. We would thus convert the `suffix3` feature described above into a very large number of numeric features, one per three-letter suffix found in the training corpus: $\texttt{suffix3}_{\text{ion}}$, $\texttt{suffix3}_{\text{ait}}$, $\texttt{suffix3}_{\text{ons}}$, etc. If our feature set is comprised of these three features, then for the token "*eduction*", the resulting feature vector will be (1,0,0), for "*mangeait*" it will be (0,1,0), and for "*bavardage*" it will be (0,0,0).

- Numeric features are normalised to [0,1]. For example, in the case of `length` defined above as the length l of a token, we can take `length` $= 1$ if $l > 10$, otherwise `length` $= l$ / 10.

2.8.2 Perceptrons

The perceptron learning algorithm, first defined by Rosenblatt [1958], has a long history in artificial neural networks, and was made popular in NLP contexts by Collins [2002]. In the present thesis, we use our own implementation of the perceptron algorithm, which can be found in the Talismane source code. Although the artificial intelligence community has developed much more complex neural networks, the perceptron algorithm used in robust machine learning for NLP since Collins is the simple single-layer perceptron.

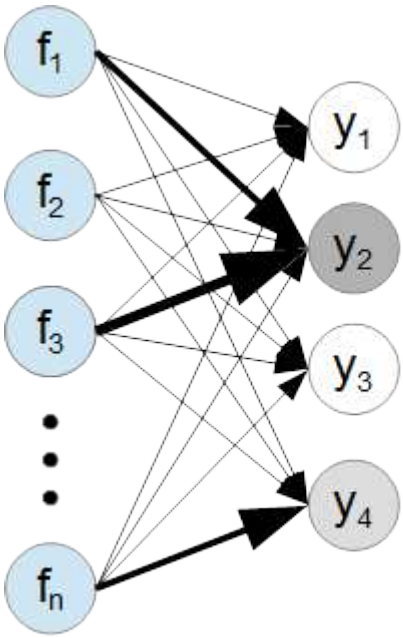


Figure 2.4: Classification using perceptrons

The basic intuition behind the perceptron algorithm is shown in fig. 2.4. The classification problem is modeled here as an artificial neuron network, where a layer of neurons "activates" the next layer of neurons by passing signals of various weights through the arrows (known as synapses). In the single-layer perceptron, we have in fact two layers of neurons: the input layer on the left, with one neuron per feature f, and the output layer on the right, with one

neuron per label y. The input layer of neurons is fully interconnected to the output layer through synapses, and, critically, each synapse between the input layer and the output layer has a weight assigned to it.

When analysing using single-layer perceptrons, we first project our feature set onto the linguistic context being analysed. This returns a feature vector, typically containing a long series of 1's and 0's. We then turn to the input layer neurons: any feature with a 1 is activated, and any feature with a 0 is not. Activating a neuron in turn activates all output neurons to a lesser or greater extent, depending on the synapse weights. We simply select the output layer neuron activated by the highest sum of synapse weights. If we imagine activation as a neuron lighting up, we would select the output layer neuron which flashes with the brightest light. Since each output neuron is activated to a different extent, it is possible to interpret these activations as a probability distribution, as will be shown further down in the present section.

Training is only slightly more complex. We begin by assigning a weight of zero to all synapses. Each training example in the training set will activate a different subset of input neurons, depending on the features associated with this example. Each of these features will in turn activate different output labels to a greater or lesser extent, depending on the current synapse weight for each label. Again, the output neuron with the highest activation score, depending on its total sum of activation by the full subset of features, is then selected. If the label which activated the output neuron with the strongest signal is the correct label for this training example, we do nothing. If it is incorrect, we react to this erroneous guess, by updating the synapse weights between the input layer and output layer accordingly, adding a weight of 1 to the synapse leading from each activated feature to the correct neuron, and subtracting a weight of 1 from the synapse leading from each activated feature to the incorrect neuron. We then continue on to the next training example.

Having covered the basic intuition behind this technique, we now move on to the gory details—which are, in fact, hardly any more complex than what has already been discussed. We begin, as above, with a $|\mathcal{F}|$-dimensional real-valued feature vector $(f_1, f_2, \ldots, f_n)$ representing a particular context x, where each f maps $\mathcal{X} \rightarrow [0, 1]$. These feature results are combined into a simple linear equation ψ defined in terms of a large matrix of synapse weights $\alpha_{f,y}$, one for each feature and label combination, as follows:

$$\psi(x, y) = \sum_{f \in \mathcal{F}} \alpha_{f,y} \cdot f(x) \tag{2.7}$$

In Collins [2002], the perceptron algorithm simply maps $\mathcal{X} \rightarrow \mathcal{Y}$ by taking $argmax_y(\psi(x, y))$. Since we wish to define a *probabilistic* classifier mapping $\mathcal{X} \rightarrow P(\mathcal{Y}|\mathcal{X})$, we need to convert $\psi(x, y)$ to a conditional probability $p(y|x)$. This is not a trivial conversion, since the $\alpha_{f,y}$ weights (and resulting sum) can be negative. We follow Titov and Henderson [2010] in defining the probability distribution as a normalised exponential:

$$p(y|x) = e^{\psi(x,y)/\text{absmax}(x)} / \sum_{y' \in \mathcal{Y}} e^{\psi(x,y')/\text{absmax}(x)} \tag{2.8}$$

where *absmax* is defined as the maximum absolute value of $\psi(x, y)$ for all y:

$$\text{absmax}(x) = \max_{y \in \mathcal{Y}}(|\psi(x, y)|) \tag{2.9}$$

Now, $\psi(x, y)/\text{absmax}(x)$ is in the range [-1,1], so $e^{\psi(x,y)/\text{absmax}(x)}$ is in the range $[1/e, e]$, ensuring that the results for all y are positive, and enabling us to construct a structurally valid probability distribution by dividing this term by the total sum for all $y \in \mathcal{Y}$.

```
   Input: training corpus D, feature set F, label set Y, iterations n
   Output: statistical model α
 1 α ← |F| · |Y| matrix initialised to 0 ∀α_{f,y};        // initialise statistical model
 2 for i = 1 to n do
 3     foreach (featureVector x, label y*) ∈ D do
 4         ψ ← vector of length |Y|;                        // vector to contain psi
 5         foreach featureResult in x indexed by f do
 6             foreach y ∈ Y do
 7                 ψ_y ← ψ_y + α_{f,y} · featureResult ;          // calculate psi
 8             end
 9         end
10         y' ← y with max ψ_y;                          // guessed label = argmax psi
11         if y' ≠ y* then                                  // have we guessed right?
12             foreach featureResult in x indexed by f do
13                 α_{f,y*} ← α_{f,y*} + featureResult ;        // update alpha weights
14                 α_{f,y'} ← α_{f,y'} − featureResult;
15             end
16         end
17     end
18 end
19 return α;
```

Algorithm 2.4: Training algorithm for perceptron classification

The training algorithm for estimating the various α synapse weights is shown in algorithm 2.4. The system iterates through through the entire training corpus k times. In each training iteration, the system iterates in turn through all of the training examples in $\mathcal{D}$, and, for each training example, if the maximum ψ is not associated with the correct label y (line 11), adds all feature results $f(x)$ to the $\alpha_{f,y}$ weights for the correct label y (line 13), and subtracts the same feature results $f(x)$ from the $\alpha_{f,y'}$ weights for the incorrect label y' (line 14).

Note that if a feature f co-occurs with one label y very often, and with another label y' very seldom, the feature is likely over time to get a positive $\alpha_{f,y}$ and a negative $\alpha_{f,y'}$. On the other hand, if the feature *never* co-occurs with y'', $\alpha_{f,y''}$ is guaranteed to remain at 0. Thus, f will penalise a relatively seldom co-occurring label more than one which never co-occurs. There is no generally accepted method for correcting this phenomenon in the case of perceptrons. The maximum entropy algorithm below solves this phenomenon by applying additive smoothing, as described in section 2.8.3.2.

A now standard variant of this algorithm, the *averaged perceptron*, defined by Collins [2002], maintains in memory the sum of each $\alpha_{f,y}$ weight for all iterations, and returns at the end the average weight for each $\alpha_{f,y}$ by dividing this sum by n, rather than the final value of $\alpha_{f,y}$. If the algorithm stabilizes before n iterations have completed (typically by checking if the overall accuracy hasn't changed by more than some tolerance over the previous m

iterations), the algorithm exits, so as to avoid adding a large number of identical α values and biasing the average. The claim that this avoids overfitting the model to the data has been substantiated by many studies, and this is the variant used in the present thesis.

Now, ever since Minsky and Seymour [1969], we have known that the single-layer perceptron algorithm will never manage to correctly classify data that is not linearly separable. Indeed, this is true of all three classifiers presented in this section (perceptrons, MaxEnt and linear SVMs). In a two-dimensional space with two classes of data, a linearly separable data set is defined as a set where one can draw a straight line such that one class of data will be on one side of the line, and the other class on the other side. More generally, in an n-dimensional space, linearly separable data is data where we can draw an *n-1*-dimensional hyperplane that separates the two classes. In practice, however, given the very high-dimensional spaces we deal with in NLP problems (one dimension per feature), this tends not to be an issue, since the higher the ratio of dimensions to data-points, the more likely it is that the data will be linearly separable. Even if the data is not entirely linearly separable, we can still get reasonable results after applying sufficient training iterations. This explains why the NLP community can make do with a single-layer perceptron, rather than having to use more complex multilayer perceptron algorithms.

2.8.3 Log-linear or maximum entropy models

The maximum entropy (MaxEnt) algorithm was first defined by Jaynes [1957], and made popular in NLP contexts by Ratnaparkhi [1998]. The term "maximum entropy" indicates that we are trying to find the solution which maximises the information entropy for each feature, given the constraints. The constraints in this case are our training examples. The information entropy is a measure of the uncertainty as to which label to select: the more equally distributed are the label probabilities for a given feature, the higher the entropy. Stated more plainly, we assume, for each feature, that all labels are equally probable, except to the extent that hard evidence (in the form of training examples) proves the contrary. As can be seen from this definition, the maximum entropy algorithm is probabilistic by its very nature.

In this section we follow Ratnaparkhi's formal definition of the algorithm, but with some difference of notation to keep it consistent with the other sections in the present chapter. In the present thesis, we use the Apache OpenNLP[2] implementation of the maximum entropy algorithm, which is directly inspired by Ratnaparkhi's work.

Like the perceptron algorithm, the function ϕ defining the conditional probability $p(y|x)$ is defined in terms of a large matrix of weights $\alpha_{f,y}$ for each feature and label combination, but this time they are combined as a product of exponents:

$$p(y|x) = \frac{1}{Z(x)} \prod_{f \in \mathcal{F}} {\alpha_{f,y}}^{f(x)} \tag{2.10}$$

The term $Z(x)$ is simply a normalisation factor, used to ensure that we have a correctly defined probability distribution such that $\sum p(y|x) = 1$, and is defined as:

$$Z(x) = \sum_{y \in \mathcal{Y}} \prod_{f \in \mathcal{F}} {\alpha_{f,y}}^{f(x)} \tag{2.11}$$

[2] `http://opennlp.apache.org/`

This is also known as a log-linear model of the data, because when ${\alpha_{f,y}}^{f(x)}$ is replaced with $\log({\alpha_{f,y}}^{f(x)})$, we have the following linear equation:

$$p(y|x) = \frac{1}{Z(x)} \sum_{f \in \mathcal{F}} \log({\alpha_{f,y}}^{f(x)}) \tag{2.12}$$

Note in eq. (2.10) that, since in most cases $f(x)$ returns 0 or 1 (that is, for all boolean or string features), ${\alpha_{f,y}}^{f(x)}$ will respectively return either 1, in which case the total product is unchanged, or the weight $\alpha_{f,y}$.

Note also that this definition of conditional probability is arbitrary, and in order to attain any sort of accuracy, the weight α associated with each (f, y) pair has to be carefully estimated during training. Once we have the weights α, analysis for unknown contexts x' is very simple: we simply calculate all features $f(x')$ and then calculate $p(y|x')$ for all $y \in \mathcal{Y}$ as per eq. (2.10).

2.8.3.1 GIS algorithm for maximum entropy training

The most common algorithm for estimating these weights is the Generalised Iterative Scaling algorithm (GIS) [Darroch and Ratcliff, 1972], in which the total co-occurrence counts for each feature f with each label y in the training corpus (or more precisely, the sum of feature results) are used to constrain the weights α so that eq. (2.10) above converges towards the conditional probability actually observed in the training corpus. A detailed discussion of GIS and a proof of its convergence is beyond the scope of the present thesis. Interested readers are referred to Ratnaparkhi [1998]. Skipping the "why", we now concentrate on the "how", and present the algorithm itself.

In the definitions below, we use the symbol $\tilde{p}$ to represent the observed probability of a certain event in the training corpus $\mathcal{D}$, and p to represent the model's current probability of the same event.

The GIS algorithm requires that the feature results for any $x \in \mathcal{D}$ sum up to a constant C. In order to enforce this constraint, we define this C by calculating the sum of feature results for each feature vector x in the training corpus $\mathcal{D}$, and setting C to the maximum sum. We then force all feature vectors to meet this constraint by adding to each vector a "corrective" feature f_C such that $f_C(x) = C - \sum_{f \in \mathcal{F}} f(x)$.

GIS now defines the following iterative method for updating the weights α, where superscript numbers in parentheses indicate the iteration number:

$${\alpha_{f,y}}^{(0)} = 1 \tag{2.13}$$

$${\alpha_{f,y}}^{(n+1)} = {\alpha_{f,y}}^{(n)} \cdot \left(\frac{E_{\tilde{p}}(f, y)}{{E_p}^{(n)}(f, y)} \right)^{\frac{1}{C}} \tag{2.14}$$

where $E_{\tilde{p}}(f, y)$ is the observed expectation for each (feature, label) combination, defined as follows:

$$E_{\tilde{p}}(f, y) = \sum_x \tilde{p}(x, y) f(x) \tag{2.15}$$

and ${E_p}^{(n)}(f, y)$ is the model expectation for each (feature, label) combination, defined as follows:

$${E_p}^{(n)}(f, y) = \sum_x \tilde{p}(x) p^{(n)}(y|x) f(x) \tag{2.16}$$

In eq. (2.15), $\tilde{p}(x, y)$ refers to the observed probability of x with y, which is $1/|\mathcal{D}|$ when the linguistic context x appears with label y, and 0 otherwise. Thus, we are only including those training examples in which the associated label is y. In eq. (2.16), $\tilde{p}(x)$ refers to the observed probability of the linguistic context x, which is always $1/|\mathcal{D}|$. Since all non-zero terms in both equations are multiplied by the coefficient $1/|\mathcal{D}|$, these coefficients cancel out, and we can rewrite the equations as follows:

$$E_{\tilde{p}}(f, y)^* = \sum_x f(x) \text{ where } (x, y) \in \mathcal{D} \tag{2.17}$$

$$E_p{}^{(n)}(f, y)^* = \sum_x p^{(n)}(y|x) f(x) \tag{2.18}$$

The term $p^{(n)}(y|x)$ was already defined in eq. (2.10).

We are now ready to examine the GIS training algorithm itself, as shown in algorithm 2.5. We first calculate the constant C and add the corrective feature f_C to all feature vectors. After initialising all of the statistical model weights to 1, the observed expectation $E_{\tilde{p}}(f, y)^*$ is calculated as per eq. (2.17) on lines 7 to 12. We then begin our iterations. We first calculate the conditional probability $p(y|x)$ as per eq. (2.10) on lines 14 to 21. Next, we calculate the new model expectations $E_p{}^{(n)}(f, y)^*$ as per eq. (2.18) on lines 22 to 27. Finally, we're ready to update the statistical model weights as per eq. (2.14) on lines 28 to 32. When all iterations have completed, the statistical model is returned.

2.8.3.2 Additive smoothing

In eq. (2.10), we calculate the conditional probability for each label $y \in \mathcal{Y}$, given a context x. The weight α for a given feature f and label y will generally be > 1 for feature/label pairs with high co-occurrence rates, and < 1 for feature/label pairs with low co-occurrence rates. However, if a feature and label *never* co-occur in the training corpus, α necessarily remains at its initial value of 1, and has no effect on the final product. Thus, as was the case with perceptrons, a feature will penalise a relatively seldom co-occurring label more than a label with which it never co-occurs.

In order to remedy this situation, we can apply additive smoothing: whenever a feature occurs with a given label y', we assume it also occurs with all other labels $y \in \mathcal{Y}$ a very small number of times (e.g. 0.01). Thus, in the observed expectation matrix, the total co-occurrence sum for every feature will be guaranteed to be positive for every label, and none of the α weights in the statistical model will remain at their initial values of 1. This, however, can have adverse affects as well: the very small α for never occurring feature/label combinations can offset other strong indicators for the same label in a given context. In the present thesis, we have attempted additive smoothing for several training scenarios, but have always achieved better results without applying it.

2.8.3.3 Inverting numeric features

As was mentioned before, we tend to normalise all numeric features to [0,1].

Now, let us take a numeric feature, such as `length` giving the length l of a token, calculated as was described above as `length` $= 1$ if $l > 10$, else `length` $= l$ / 10. Let us assume that long words have a higher probability of being common nouns (label `NC`) and short words have a higher probability of being prepositions (label `P`).

```
Input: training corpus 𝒟, feature set ℱ, label set 𝒴, iterations k
Output: statistical model α
C ← 0 ;
foreach (featureVector x, label y) ∈ 𝒟 do
    if ∑ featureResult ∈ x > C then C ← ∑ featureResult ∈ x;
end
add corrective feature f_C based on C to each feature vector x ∈ 𝒟 ;
α ← |ℱ| · |𝒴| matrix initialised to 1 ∀α_{f,y};        // initialise statistical model
observedExp ← |ℱ| · |𝒴| matrix ;          // observed expectation of each (f,y)
foreach (featureVector x, label y) ∈ 𝒟 do
    foreach featureResult in x indexed by f do
        observedExp_{f,y} ← observedExp_{f,y} + featureResult;
    end
end
for i = 1 to k do
    Π ← |𝒟| · |𝒴| matrix;                          // matrix to contain p(y|x)
    foreach (featureVector x, label y'') ∈ 𝒟 indexed by d do
        Π_{d,y} ← 1  ∀y ∈ 𝒴 ;
        foreach featureResult in x indexed by f do
            Π_{d,y} ← Π_{d,y} · α_{f,y}^{featureResult}  ∀y ∈ 𝒴;          // calculate p(y|x)
        end
        Π_{d,y} ← Π_{d,y}/(∑ Π_{d,y'} ∀y' ∈ 𝒴) ∀y ∈ 𝒴;              // normalise by 1/Z(x)
    end
    modelExp ← |ℱ| · |𝒴| matrix;            // model expectation of each (f,y)
    foreach (featureVector x, label y') in 𝒟 indexed by d do
        foreach featureResult in x indexed by f do
            modelExp_{f,y} ← modelExp_{f,y} · Π_{d,y} · featureResult  ∀y ∈ 𝒴 ;
        end
    end
    foreach feature f ∈ ℱ do                    // update alpha weights in model
        foreach y ∈ 𝒴 do
            α_{f,y} ← α_{f,y} · (observedExp_{f,y}/modelExp_{f,y})^{1/C};
        end
    end
end
return α;
```

Algorithm 2.5: GIS algorithm for Maximum Entropy training

In eq. (2.10), this feature will have a high α for `NC` (say 10.0) and a low α for `P` (say 0.1). This means that when a word is over 10 characters long and `length`=1.0, the feature will multiply the conditional probability for `NC` by $10^{1.0} = 10$ and multiply it by $0.1^{1.0} = 0.1$ for `P`. However, when the word is only 1 character long and `length`=0.1, the feature will still favor the label `NC`, albeit to a lesser extent, multiplying the conditional probability by $10^{0.1} = 1.26$ for `NC` and by $0.1^{0.1} = 0.79$ for `P`. Thus, the feature is in itself incapable of giving a higher probability for `P` even when the word is short.

If we want to enable the feature to increase the probability for `P`, we need to include an equivalent inverted feature, calculated as 1-`length`+0.1. This will give 1.0 for a word that is only 1 character long, and 0.1 for a word that is over 10 characters long. Such a feature can now effectively increase the conditional probability of `P` at the expense of `NC`.

2.8.4 Linear SVMs

We will gloss over linear SVMs in far less detail than perceptrons and maximum entropy models, since the mathematical concepts behind them are far more complex than those outlined above, and far too complex for a cursory discussion. However, because they give the highest accuracy in certain of our experiments, we include them in our general discussion.

The basic idea behind SVMs [Vapnik, 1995], for a given $|\mathcal{F}|$-dimensional dataset (consisting of N feature vectors x), is to find an $(|\mathcal{F}|-1)$-dimensional hyperplane which separates the vectors into two classes in such a way that maximises the distance from the hyperplane to the closest vectors (the so-called "support vectors"). In order to allow for datasets which are "almost" linearly separable, SVMs typically allow certain vectors to be misclassified, but constrain the degree of non-classification through a so-called slack variable ξ. When training, this slack variable is indirectly controlled through the so-called "soft margin" parameter C. Without going into the gruesome details, a high value of C allows for less misclassified vectors, and can lead to over-fitting by overusing the information in the feature vectors, whereas a low value of C allows for more misclassified vectors, thus ignoring critical information in the feature vectors [Alpaydin, 2004, p. 224]. If $C{=}\infty$, we have a hard-margin SVM, allowing for no misclassified vectors at all.

The other parameter used when training linear SVMs is ϵ. This parameter defines a zone around the hyperplane (known as the ϵ-insensitive zone or insensitivity zone), in which classification errors are completely ignored. The wider the zone, the less closely we need to fit the hyperplane to the data.

Determining the correct values of C and ϵ is largely a matter of trial and error, for example, by attempting various powers of 2 from 2^{-4} to 2^{4} for C, and various powers of 10 from 10^{-4} to 10^{-1} for ϵ, and selecting the values giving the best accuracy for an evaluation corpus.

Once the best separating hyperplane has been found, unknown data is classified by identifying its location with respect to this hyperplane. We can then use the geometric distance between a given vector and the closest point in the hyperplane as a representation of the SVM's confidence in its decision: the greater the distance, the higher the probability that the vector is indeed in the correct classification. This allows us to use SVMs to construct a probability distribution.

Much modern research has concentrated on the "kernel" trick, in which the data set undergoes a non-linear transformation, by adding a dimension which is a non-linear function of the other dimensions, prior to attempting to find the separating hyperplane. This allows us to apply SVMs to data which is not linearly separable, as shown in the fig. 2.5 for finding a separating line in 2-class data which is not initially linearly separable. However, the geometric distance can no longer be interpreted as an accurate measure of confidence, since the non-linear transformation has distorted the space on which the vectors lie. We thus concentrate our research in the present thesis on linear SVMs only, which are the only ones capable of mapping $\mathcal{X} \rightarrow P(\mathcal{Y}|\mathcal{X})$. Joachims [2006] gives the full mathematical and algorithmic details for solving the linear SVM training problem in linear time. The implementation we use is a

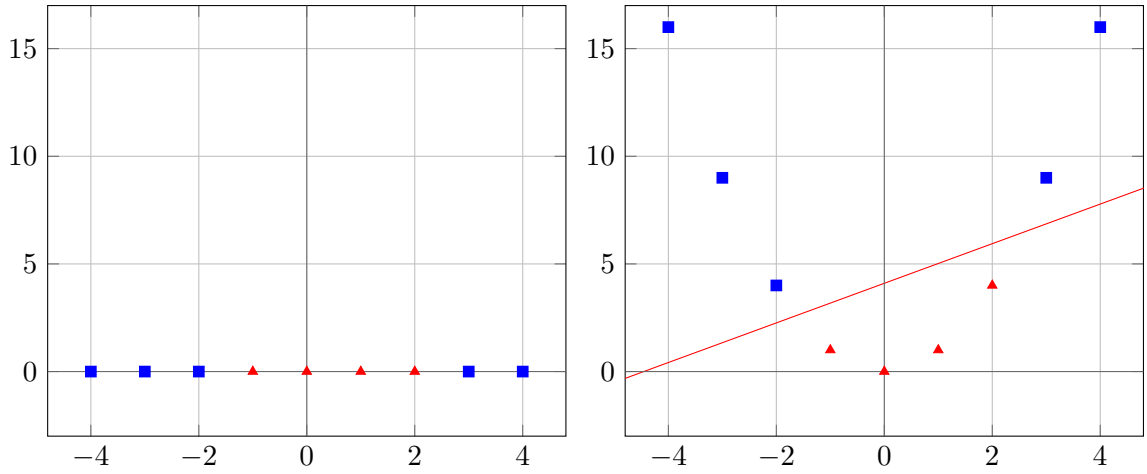


Figure 2.5: Applying the kernel trick to a 2-class SVM: $y \to x^2$

Java port of LibLinear[3] [Ho and Lin, 2012].

Like perceptrons and maximum entropy models, for a given feature f, SVMs penalise a relatively seldom co-occurring label more than one which never co-occurs with f.

2.8.5 Classifier comparison

Although no classifier comparison can be complete without measuring accuracy on a test data set, we nevertheless attempt a short comparison of the three classifiers presented in this chapter.

Classifier:	**perceptron**	**maxent**	**linearSVM**
Training time	fast	middle	slow
Algorithm complexity	simple	middle	complex
Training method	incremental	total	total
Probabilistic	non-trivial	by nature	non-trivial
Parameters	iterations, cutoff	iterations, cutoff	C, ϵ, cutoff
Parameter complexity	intuitive	intuitive	complex

Table 2.2: Classifier comparison

The comparison shown in table 5.2 concentrates on several points which are important to us from a practical perspective. Training time is important in an experimental framework, because a faster training time simplifies the testing of multiple competing configurations. Algorithm complexity attempts to capture the ease with which we can look into the black box to explain and resolve analysis anomalies. The interest of an incremental training method, where the model is updated one training example at a time, is critical in some cutting-edge global learning approaches, where we mix training examples with other criteria to adapt the model to a given application. By contrast, classifiers in which the model parameters are based on the totality of training examples cannot easily incorporate anything other than the

[3] `http://liblinear.bwaldvogel.de/`

training data itself in the training process. However, we do not explore these global learning possibilities in the present thesis. The probabilistic nature of classifiers is required for beam search methods: successfully mapping the classifier results to a probability distribution is critical to enable the beam search to increase accuracy. In section 5.4, we will thus compare the accuracy of the various classifiers not only in the deterministic (or greedy) scenario, but also with higher beam widths. Regarding parameters, the perceptron and maxent algorithms both involve two fairly intuitive parameters: the number of training iterations and the feature frequency cutoff. As we will see in chapter 5, these parameters behave in a fairly rational manner, with easily interpretable curves. On the other hand, there is no intuitive meaning to the C parameter in linear SVMs, forcing us to perform a grid search in order to find the best SVM parameters for each machine learning problem.

2.9 Supervised machine learning project examples

Having presented a formal framework for supervised machine learning and defined certain learning algorithms in detail, we now turn to some supervised machine learning projects in which I was involved since the start of my doctoral studies. We will discuss their general context, aims and results, and show how they instantiate the various formal concepts presented in this chapter.

2.9.1 Authorship attribution

In 2011, the NLP group of the CLLE-ERSS laboratory decided to participate in the shared task PAN (Plagiarism analysis, Authorship identification, and Near-duplicate detection). This collaborative project involved Basilio Calderone, Nabil Hathout, Franck Sajous, Ludovic Tanguy and myself, with the suggestions and help of the entire NLP group [Tanguy et al., 2011]. The task was divided into three authorship attribution subtasks, centered around small subsets of the Enron e-mail corpus[4]:

1. Authorship attribution: Indicate the author for each e-mail within a set of e-mails
2. Authorship attribution with unknown authors: Indicate the author for each e-mail within a set of e-mails, with the possibility of marking the author as "unknown"
3. Authorship identification: Within a set of e-mails, identify those belonging to a particular author

The goal was thus, in all cases, to recognise the "style" of a particular author as opposed to the styles of the others. In each case, the sub-task included a training set, a development set, and a test set provided shortly before the result submission deadline. A large majority of participants in this task used a linguistically "poor" approach: the only features used to characterise the e-mails were character trigram frequencies. In other words, if the e-mail started with the line "Hello Mom,", the feature set would include: {"`Hel`", "`ell`", "`llo`", "`lo_`", "`o_M`", "`_Mo`", "`Mom`", "`om,`"}. This has the advantage of being completely language neutral, and requiring no prior linguistic processing. Our team's hypothesis was that linguistically "rich" features can outperform a character trigram approach. These features included morpholigical features (e.g. word counts with different morphological prefixes/suffixes), syntactic

[4] `https://www.cs.cmu.edu/~enron/`

features (e.g. part-of-speech trigrams, average and maximum syntax tree depth), semantic features drawn from the Princeton Wordnet [Fellbaum, 1998], and a variety of ad-hoc features (e.g. stop-word usage, average sentence length).

Since each author in our team was responsible for generating a different set of features, I developed collaborative machine learning software, csvLearner[5], in which the features were read from a set of CSV files. In addition to simply reading and combining the features for each e-mail, this software makes it possible to normalise features (with respect to the max or mean value of a single feature or of an entire group of closely related features), to discretise features (converting a real-valued feature into a set of nominal categories), and to perform various evaluation tasks such as cross-validation.

In terms of the formal machine learning framework presented in the present chapter, our approach could be summarised as follows:

- linguistic context x: a single e-mail
- label y: an author's numeric identifier
- training set $\mathcal{D}$: a collection of e-mails, each tagged with an author identifier
- feature set $\mathcal{F}$: the various features described above, as well as character trigram frequencies and word frequencies. The vast majority of features were numeric.
- probabilistic classifier ϕ: maximum entropy

From a machine learning perspective, we validated the advantage of normalising a group of related features (e.g. word frequencies) with respect to the group max, instead of the individual feature max, so as to avoid losing information on relative frequencies, as can be seen in table 2.3.

Method	**Mean Accuracy**	**Standard deviation**
Discretisation	66.18%	2.75%
Normalisation (max)	67.08%	2.96%
Normalisation (mean)	67.21%	2.58%
Grouped normalisation (max)	69.74%	2.05%
Grouped normalisation (mean)	69.44%	2.17%

Table 2.3: Authorship attribution results

In the tasks involving unknown authors, the choice of a probabilistic classifier made possible a simple criterion for separating known and unknown authorship, in this case the classifier's confidence in its best guess. If the confidence was below a certain threshold, the author was marked as unknown. This simple method proved quite efficient: we ranked first in both authorship attribution tasks with unknown authors. Our overall 1st place ranking in authorship attribution tasks seems to validate our hypothesis about linguistically rich features, although subsequent tests indicate that the linguistically rich features tend to be more useful in small data sets, and that for very large datasets character trigrams tend to be sufficient.

[5] `https://github.com/urieli/csvLearner`

2.9.2 Jochre: OCR for Yiddish and Occitan

In 2009 I began developing software for the optical character recognition of Yiddish using supervised machine learning techniques: Jochre[6] (Java Optical CHaracter REcognition). This came about through a collaboration with the Yiddish Book Center, in Amherst, Massachusetts, which had collected 1.5 million Yiddish books throughout the world representing 18,000 unique titles, and had scanned 12,000 of them, published for the most part between 1870 and 1960 and freely downloadable online[7].

In 2013, Marianne Vergez-Couret and myself decided to adapt Jochre to Occitan, in order to perform a comparative OCR study of Yiddish and Occitan, both unnormalised languages with a variety of dialects and spelling conventions [Urieli and Vergez-Couret, 2013]. We were particularly interested in the reranking of OCR guesses by means of a lexicon.

Jochre performs OCR in three steps:

1. Segmentation: break up the image into paragraphs, rows, groups (representing words) and shapes (representing letters) using ad-hoc statistical methods

2. Letter recognition: guess the letter corresponding to each shape via supervised machine learning

3. Reranking: correct the guesses for each word with the help of a lexicon

In terms of the formal framework presented in the present chapter, our machine learning approach for letter recognition could be summarised as follows:

- linguistic context x: a single shape, and the history of letters assigned to the previous shapes in the current group

- label y: the letter or letters corresponding to the shape

- training set $\mathcal{D}$: 95 pages of Yiddish and 80 pages of Occitan, selected to display a variety of fonts (and dialects for Occitan), segmented by Jochre, and manually annotated with the correct letters using the JochreWeb interface

- feature set $\mathcal{F}$:

    - letter bigrams and trigrams, as well as a feature indicating if the shape is the last one in a group
    - a relative brightness grid for each shape, in which the shape is broken up into a 5×9 grid, and each section is assigned a value from 0 to 1 giving its total pixel darkness with respect to the darkest section (assigned a value of 1)
    - vertical size, vertical elongation, distance from the baseline
    - specialised binary features for Hebrew letters used to separate similar letters by structural differences at a higher level of abstraction than the brightness grid

- probabilistic classifier ϕ: maximum entropy

[6] `https://github.com/urieli/jochre`
[7] `http://www.yiddishbookcenter.org/books/search`

We constructed a lexicon of inflected forms for Yiddish from an XML version of Niborski and Vaisbrot [2002], provided by Harry Bochner, co-editor of the English translation of this dictionary [Beinfeld and Bochner, 2013]. The Occitan lexicon was derived from works already added to the BaTelÒc litarary corpus of Occitan [Bras, 2006]. Lexicon-based reranking was performed by multiplying the score for unknown words by a configurable unknown word coefficient in the range (0,1]. The main results are shown in table 2.4 for Yiddish and in table 2.5 for Occitan.

	Baseline features		**Inverse numeric features**		**Hebrew alphabet features**	
	Words	**Letters**	**Words**	**Letters**	**Words**	**Letters**
No lexicon	84.94%	95.61%	87.11%	96.31%	89.74%	97.09%
With lexicon	87.48%	96.28%	89.14%	96.84%	**91.20%**	**97.44%**

Table 2.4: Jochre Yiddish OCR results

	Full corpus		**Languedoc dialect corpus**		**Gascon dialect corpus**	
	Words	**Letters**	**Words**	**Letters**	**Words**	**Letters**
No lexicon	91.54%	97.53%	92.08%	97.64%	90.99%	97.41%
Gascon lexicon	92.72%	97.81%	93.07%	97.85%	**92.36%**	**97.78%**
Languedoc lexicon	92.83%	97.86%	**94.10%**	**98.15%**	91.53%	97.56%
Full lexicon	**93.13%**	**97.93%**	94.08%	98.13%	92.16%	97.71%

Table 2.5: Jochre Occitan OCR results

This study highlights, among other aspects, the usefulness of inverse numeric features given the preponderance of numeric features in OCR and similar signal processing type applications. Furthermore, we see the advantage of reranking the raw scores (letter probabilities) using information only available at a higher level of abstraction (word inclusion in a lexicon).

2.9.3 Talismane—Syntax analysis for French

The final supervised machine learning project in which I was involved is the syntax analyser Talismane, about which certain results have already been published with Ludovic Tanguy in Urieli and Tanguy [2013]. Since this project is covered in detail in the remaining chapters of this thesis, it will not be discussed in the present chapter.

2.10 Discussion

Our goal in this chapter was to gain a sufficient understanding of machine learning mechanisms so as to enable us to apply them correctly and productively in the parsing process. To this end, we presented a formal conceptual and symbolic framework for supervised machine learning that will be used throughout the remainder of this thesis, as illustrated by the case study of pos-tagging. Within this framework, the linguist needs a thorough understanding

of the algorithm selected to translate the original problem into a classification problem, of the meanings of the various labels, and of the exact nature of the linguistic context to be classified at each step of the task. Armed with this understanding, the linguist is now ready to define specific features that are both predictive with respect to the information needed to select a label, and generalisable with respect to other corpora in the target sub-language.

We presented various projects making heavy use of probabilistic classifiers, which map $\mathcal{X} \rightarrow P(\mathcal{Y}|\mathcal{X})$ rather than $\mathcal{X} \rightarrow \mathcal{Y}$, thus excluding the currently dominant current of kernel-based methods. The use of probabilities for the beam search and for classifier confidence measure is central to the rest of this thesis.

When comparing classifier types, we are not so much interested in overall classifier accuracy (although this is of course important), but also in their accurate estimation of probabilities for the beam search (see section 5.4), and in the ease with which we can peek into the statistical black box to understand anomalies and/or possibly manipulate the training method to suit our needs. Although we did not present this methodology within the present thesis, we found ourselves often examining the weights assigned to individual features in the perceptron or maxent models in order to understand unusual behavior in a given context.

Many articles in machine learning circles concentrate on tuning of machine learning algorithms via their parameters and other more complex manipulations. Our approach can instead be said to be *feature centered*, in that we concentrate less on the choice of algorithm in itself and the manipulations it undergoes than on the ease with which linguistic knowledge is incorporated into a robust system, by means of features and rules. In the next two chapters, we will thus explore the mechanisms for injecting linguistic knowledge more thoroughly. Chapter 3 describes Talismane's implementation of each of the four modules in enough detail to enable us to understand the information available at each step in the process and the type of decision being made. Chapter 4 gives details on where and how linguistic knowledge can actually be incorporated, and on the types of resources that are currently available for French.

The reality, however, is that a good baseline configuration with respect to the pure machine learning aspects is critical, and when selecting this baseline, the method providing significant improvements in accuracy, in both the greedy and beam search scenarios, will always be preferred to other methods. Experiments for finding the best baseline machine learning configuration are presented in chapter 5. It is only when the baseline provided by different methods shows no significant differences that we have the luxury to select a classifier based on other criteria.

Part II

Syntax analysis mechanism for French

Chapter 3

The Talismane syntax analyser - details and originality

In this chapter, we delve into the philosophy behind the Talismane syntax analyser (section 3.1), its general architecture (section 3.2), and its high-level implementation details. As mentioned in the introduction, Talismane was designed as a highly configurable tool that creates as many openings as possible for linguists to inject linguistic knowledge. Our main purpose is to show enough of the inner workings of these modules to give the linguist the necessary keys to designing useful features.

We thus begin by examining the problem definition for each of the four modules in section 3.3, showing how their specific task is converted into a classification problem within the formal framework presented in the previous chapter, and giving examples of typical ambiguities that need to be resolved.

We then move on to the mechanics that will allow us to help resolve these ambiguities, examining Talismane's rich feature configuration syntax in section 3.4, along with various specific aspects that make it simpler to define feature combinations, or to deal with features that return mutliple values for the same linguistic context.

Another innovative aspect of Talismane is the ability to define rules that override the statistical model in all four of the basic modules. The mechanism for applying these rules, as well as the syntax used to define them, are described in section 3.5.

We then look briefly into filters (section 3.6), which enable us to determine which portions of the raw text to parse, before comparing Talismane with similar projects in section 3.7.

3.1 Philosophy

TALISMANE stands for "*Traitement Automatique des Langues par Inférence Statistique Moyennant l'Annotation de Nombreux Exemples*" in French, or "*Tool for the Analysis of Language, Inferring Statistical Models from the Annotation of Numerous Examples*" in English.

It is an open source statistical syntax analyser for natural languages written in Java, developed within the framework of the present doctoral thesis. With over 85,000 lines of code, it covers a full range of functions needed for a successful NLP project, from machine learning algorithms, to a feature syntax compiler, to the flexible incorporation of external resources, to the training, analysis and evaluation of each task in the syntax analysis chain. Many

aspects of Talismane's behaviour can be tuned via the available configuration parameters, and indeed, one of the primary concerns was to include as many openings as possible for resources and configuration files defined by linguists. Furthermore, Talismane is based on an open, modular architecture, enabling a more advanced user to easily replace and/or extend the various modules, and, if required, to explore and modify the source code[1]. It is distributed under an open-source license in order to encourage its non-commercial redistribution and adaptation.

Talismane should be considered as a framework which could potentially be adapted to any natural language. The present thesis presents a default implementation of Talismane for French, but the overall architecture was designed to clearly separate language-neutral interfaces from language specific implementations. As long as a training corpus is available, it can theoretically be adapted to many different languages out-of-the-box—although some may require a completely new implementation for certain modules, such as Chinese tokenisation, which is radically different to its French counterpart.

The portability offered by Java enables Talismane to function on most operating systems, including Linux, Unix, MacOS and Windows.

Full documentation for the current version of Talismane is available at the Talismane home page[2], and the interested reader is encouraged to visit this page for details missing from the present discussion.

3.2 Architecture

Talismane transforms raw text, i.e. a stream of characters, into dependency parse trees. It is divided into four statistical modules:

- Sentence-boundary detection: marking the end of each sentence within the raw text
- Tokenisation: marking the boundaries between syntactic units (or tokens) in each sentence, which may or may not correspond to graphical word boundaries.
- POS-tagging: part-of-speech tagging, in which each token is assigned a single part-of-speech from the tagset.
- Dependency parsing: finding a governor for each token in the sentence, and assigning a label to the dependency arc between the governor and dependent from a closed set of dependency labels.

Talismane was designed to plug into a full processing chain without complicated adaptation. To this end, Talismane can either read from an input file or from the standard input stream, and can output to either an output file or to the standard output stream. Furthermore, Talismane can begin and end processing with any one of its four modules. There is a default input and output format for each of the modules (CoNNL-X format for the parser), but these can be overridden by configuration files to suit the needs of a particular processing chain.

Each of the last three modules can be configured to use a beam search. Furthermore, between modules, the system can be configured to be deterministic (or "greedy"), passing

[1] `https://github.com/urieli/talismane`
[2] `http://redac.univ-tlse2.fr/applications/talismane.html`

only the best solution to the next module in the chain, or non-deterministic, where the entire final beam is propagated on to the next module in the chain, allowing a higher-level module to resolve ambiguities left open by a lower-level module. Experiments with beam propagation are described in section 5.5, page 147.

3.3 Problem definition for Talismane's modules

This section presents the abstract problem definition for each of the four modules in Talismane. More concrete details, such as the actual training corpus and the baseline feature set used for the French implementation will be presented in the chapter 4.

3.3.1 Sentence boundary detection

The Talismane sentence boundary detector takes a stream of raw unannotated text as input, and breaks it up into sentences as output. Restating the problem as a binary classification problem, the sentence detector's role is to decide, for each sentence boundary candidate (characters in the subset {`.`, `?`, `!`, `"`, `)`}), whether it is really a boundary or not.

In sentence detection, the main sources of ambiguity are:

1. Strong punctuation marks (`.`, `?`, `!`) followed by double quotes or parentheses, where the actual sentence boundary can be pushed beyond the period to the next punctuation mark:
   a) No sentence break: He said "leave me alone[.]" Then...
   b) Sentence break: I met him[.] "Leave me alone," he said.
   c) Possible features: spaces between strong and weak punctuation marks.
2. Abbreviations followed by a period and a space, especially when the following word is capitalised:
   a) No sentence break: "Mr[.] Smith is home."
   b) Sentence break: "I bought apples, oranges, etc[.] Then, I..."
   c) Possible features: Is the following word capitalised? Is the preceding word on a list of known abbreviations? If so, is it typically followed by capitalised words (e.g. "Mr.")? If the word following the period is capitalised, is it known in the lexicon as something other than a proper noun (e.g. "Then" is known, "Smith" is not)?
3. Strong punctuation inserted inside a sentence
   a) No sentence break: "I saw Smith together with - can you believe it[?] - Jones."
   b) Sentence break: "I saw Smith together with Jones - can you believe it[?] Jones said...
   c) Possible features: Is the strong punctuation mark followed by other punctuation? Is the following word capitalised?
4. Numbered lists inside a sentence

a) No sentence break: "The prizes were awarded to 1[.] Smith for outstanding work, 2[.] Jones for..."

b) Sentence break: "It is obvious that 2 and 2 make 4[.]"

c) Possible features: is the sentence preceding the period a complete sentence? Are there several related numbers in the same block of text? Is the number directly preceded by weak punctuation (e.g. a comma)?

The decision is taken using a supervised machine learning approach. In terms of the formal definition given in chapter 2, sentence boundary detection can be defined as follows:

- linguistic context x: a single boundary character, with access to the n characters to the right and left of this boundary character
- label y: is the context x a `true` or `false` sentence boundary
- training set $\mathcal{D}$: a corpus of text, where each true sentence break is followed by a paragraph break

Most of the features described above are possible to encode within this context, although some may require the use of external resources in the form of lists for known abbreviations, or a lexicon to distinguish proper nouns from other words. In the last example, asking whether the sentence preceding the number is a complete sentence is clearly not possible, since we have no access to syntactic information at this point in the process. Similarly, trying to find related numbers in the same block of text could be difficult, since it may require an arbitrarily large number n of characters to the right or left of the current character. The actual baseline feature set used by the sentence boundary detector is defined in section 4.4.2, page 116.

In Talismane's design, sentence boundary detection is final: the decision taken cannot be overridden by higher level modules. Any other design would introduce considerable complexity, since we would have to analyse in parallel several potentially overlapping sentence candidates and find a reliable algorithm for comparing the analyses and choosing the best one. This means final decisions are taken with very little information: the n characters to the right and left of a sentence boundary candidate. Luckily, highly ambiguous cases are very rare in edited written text. The case of unedited text such as blogs or open discussion pages is much more problematic, since authors are not necessarily careful about their spaces and capitalisation. Indeed, most of the sentence detection errors in our own tests are due to uncapitalised words after sentence boundaries. This illustrates our strong dependence on the training corpus sub-language. We could of course attempt to ignore all capitalisation in our features, but this is likely to considerably lower results for edited text.

3.3.2 Tokenisation

The Talismane tokeniser takes a sentence as input, and transforms it into a sequence of tokens (or syntactic units) as output. The vast majority of tokens correspond to graphical word boundaries, but certain polylexical units (or compound words) group several graphical words into a single token. The tokeniser combines a pattern-based approach with a statistical approach. The patterns are language specific, and ensure that only areas where any doubt exists concerning the tokenisation are actually tested statistically.

Typical examples of tokeniser ambiguity in French include:

1. *bien que*

   a) Joined = Subordinating conjunction: "*Tu sais,* bien_que *je ne travaille pas le dimanche, je ferai une exception.*" ("You know, *even though* I don't work on Sundays, I'll make an exception.")

   b) Separate = Adverb + subordinating conjunction introducing a direct object: "*Tu sais* bien *que je ne travaille pas le dimanche.*" ("You know *well* that I don't work on Sundays.")

   c) Possible features: is *bien que* the first word in the sentence, and, if not, does it follow a punctuation mark? Is the preceding verb on a list typically modified by *bien* (e.g. *savoir*, *aimer*, ...)? Does it sub-categorise *que* as a direct object?

2. *quitte à*

   a) Joined = Preposition: "Quitte_à *étudier, autant s'amuser.*" ("*If you must* study, at least have fun.")

   b) Separate = Verb + preposition: "*Elle me* quitte *à cause de ma thèse.*" ("She's *leaving* me because of my thesis.")

   c) Possible features: is *quitte à* the first word in the sentence, and, if not, does it follow a punctuation mark? What is the part of speech of the word directly preceding it (since clitics such as *je* or *me* are strong indicators for a verb)?

3. *de la*

   a) Joined = partitive determiner: "*Je mange* de_la *tarte.*" ("I'm eating *some* pie")

   b) Separate = preposition + determiner: "*Je parle* de *la chemise.*" ("I'm speaking *about* the shirt")

   c) Possible features: Is the preceding verb transitive? Does the preceding verb typically govern an object with the preposition *de*? What is the part-of-speech of the previous word (e.g. *"avec de la"* always implies a partitive determiner)?

In almost all of these ambiguities, one variant is far more common than the other. For *bien que* and *quitte à*, it is rare to find cases where the tokens are separate. The opposite is true for *de la*, were the compound form is much rarer. This makes it difficult to disambiguate without very strong signals. Also, almost all of the features discussed above require information which is not available when tokenising: the part-of-speech of the preceding word, or being able to recognise whether the preceding word is a verb and, if so, knowing its subcategorisation frames. This information is available to Talismane only after the pos-tagging phase. We can attempt to inject a certain amount of knowledge by basing our features on a lexicon, in which we can look up the parts-of-speech associated with a given word form. If we are lucky enough to have a lexicon with reliable subcategorisation frames, even more information can be injected—otherwise, we may have to rely on the small subset of subcategorisation frames which happen to occur in our training corpus. The actual baseline feature set used by the tokeniser is described in section 4.4.3, page 117.

Example (2b) above is particularly interesting in that two potential compound words overlap: *quitte à* and *à cause de*, of which the first is separate and the second is joined. We would like a system that only allows three possibilities for this configuration: *"quitte_à cause*

de", *"quitte à_cause_de"* and *"quitte à cause de"*, and does not allow *"quitte_à_cause_de"*, unless the latter were a recognised compound in its own right. Furthermore, trying to decide between the compounds *quitte à* and *à cause de* illustrates the importance of tokeniser confidence measures: it may well be that both are considered more likely to be compound than otherwise, because of the overwhelming statistics in favor of compounding when contextual factors are ignored. In this situation, we want to select the compound with the higher confidence measure, and reject the other one. We will see in the discussion of the techncial mechanism below to what extent these goals are achieved.

The tokeniser begins by breaking the sentence up into atomic tokens. An **atomic token** is defined as a contiguous character string which is either a single separator, or contains no separators. The list of separators considered include all punctuation marks as well as the various whitespace characters: the space bar, the tab character, the newline, etc. Thus, a text such as the following: "*A-t-elle mangé de l'autruche ?*" ("Has she eaten any ostrich?") will be separated into the following atomic tokens:

`[A][-][t][-][elle][ ][mangé][ ][de][ ][l][']][autruche][ ][?]`

The expected output would be:

`[A][-t-elle] [mangé] [de l'][autruche] [?]`

Note that this approach is not language-neutral: it cannot yet handle a language in which a single graphical word needs to be broken up into two or more tokens. This is the case, for example, in Hebrew, where coordinating conjunctions, some prepositions, and definite articles are directly prefixed to the following word. However, it could fairly easily be made to apply to such languages if potential prefixes where all marked as atomic tokens. On the other hand, languages such as Chinese, were word boundaries are not marked by a white space, may well need a completely different approach.

Restating the problem as a binary classification problem, the tokeniser's role is to determine, for the interval between any two atomic tokens, whether it is separating or non-separating.

By default, all separators will be considered to be separated from both the previous and next atomic tokens. Punctuation marks can be configured to override this decision: for example, in French, we can configure the system so that certain punctuation marks (e.g. the dash in "*vice-président*") are connected by default to both the previous and next atomic tokens, and others (e.g. the apostrophe in "*l'autruche*") are connected by default to the previous atomic token, but separated from the next one.

Finally, we manually configure a set of patterns, by means of regular expressions, to indicate sequences of atomic tokens that should be submitted to further testing by supervised machine learning. For example, we could define the pattern "`(.+)-t-(elle|elles|il|ils|on)`", indicating that an atomic token sequence such as "*A-t-elle*" may need to override the defaults, in order to separate the initial dash from the atomic token preceding it. Similarly, a pattern such as "`de l'`" would indicate that, in the atomic token sequence "*de l'autruche*", we need to test the space, in order to decide whether it separates the two atomic tokens "*de*" and "*l'*" (preposition + determiner) or not (partitive determiner). The list of patterns is added as a configuration file, and needs to cover all of the cases marked as compounds in the training corpus, and can be extended to cover other potential compounds through the pattern grouping mechanisms discussed below.

In terms of the formal definition given in chapter 2, tokenisation can be defined as follows:

- linguistic context x: the first pair of atomic tokens matched by a given pattern, with

access to the pattern in question, the full sequence of atomic tokens matched by the pattern, and the full sequence of atomic tokens forming the sentence

- label y: is the atomic token pair connected or separate? Label set: `join` and `separate`
- training set $\mathcal{D}$: a corpus of text, in which each compound word has been indicated via some sort of mark-up

Note that the tokeniser doesn't make any assumptions about the nature of compound words, which is a field of research unto itself. It is entirely up to the corpus annotators to decide which words are compound, and Talismane will recognise these decisions as long as they are reflected in the pattern configuration file. However, Talismane does not currently support split compounds, where another token is inserted in the midst of a compound token. For example, the French verbal compound "*tenir compte de*" ("to take into account") cannot be marked as a compound within Talismane, because it is split in the negative expression "*Je ne tiens pas compte de...*". Indeed, the shift-reduce algorithm for transition parsing, presented in section 1.2.3 cannot natively handle split compounds even if we somehow accommodated their representation in the tokeniser output. We could potentially handle such cases in post-processing, but this would have to occur after the parsing phase is complete, and the output format would have to handle such cases correctly as well (CoNNL-X, for example, has no annotation for split compounds).

Since patterns may occur quite rarely in the training corpus, Talismane allows us to manually group patterns together in the pattern configuration file. The goal is to create groups for patterns which are likely to have similar compounding behavior based on their context. For example, we might decide to group together compounds which are guaranteed to be joined when found at the start of a sentence, such as *bien que* and *alors que*. We can also cater for compounds missing from the training corpus, by including them in the same group as a more common compound. Two levels of grouping are allowed: the pattern *name*, which is used for patterns which are expected to behave identically, such as `de l'` and `de la`, and the pattern *group*, allowing a higher level grouping of patterns with similar behavior, such as all patterns playing the roles of subordinating conjunctions (`alors que`, `bien que`, `jusqu'à ce que`, etc.). When designing tokeniser features, we can then incorporate into the feature the pattern group, the pattern name, or the actual word forms matched by the pattern, thus allowing us to create features with varying granularities, in the hope of capturing statistical regularities in the least granular features (those referring to the pattern group).

3.3.2.1 Tokenisation mechanism

While a token pattern can include any number of atomic tokens, the decision taken for the interval between the first pair of atomic tokens determines the decisions to be taken for all subsequent pairs. For example, let us take the pattern `bien que`, illustrated by the two sentences in example 1 of the previous section.

This pattern is formed of the atomic tokens `[bien][_][que]`. If we decide that the first atomic token `[bien]` is connected to the second one `[_]`, as in sentence (a) above, it follows that the second and third atomic tokens are connected as well. Similarly if the first two are separated, as in sentence (b) above, the remaining atomic tokens are separated.

This allows us to perform a single statistical test per pattern, rather than one statistical test per interval. The Talismane pattern definition syntax allows us to define portions of

the pattern to which this decision will not be applied, but which are required to give the pattern its correct context. For example, in the case of the "`de l'`" described above, the final apostrophe should be excluded from the decision process, because, even if we decide that the *de* should be separated from the *l'* in a particular case, the apostrophe should still follow the default for apostrophes and be connected to the preceding *l*. We indicate this using the curly brackets: "`de l{'}`".

Testing each pattern as a whole, rather than testing each individual interval within the pattern, allows us to avoid constructing non-existent compound words (especially deeper down in the beam). However, we still have to deal with the case where a given atomic token is contained in a number of overlapping patterns. Let us assume our French training corpus considers the subordinating conjunctions "`bien que`" and "`si bien que`" as compound words. Then any appearance of the sequence "*si bien que*" can be tokenised as follows: "*si_bien_que*", "*si bien_que*", or "*si bien que*". The current mechanism stops whenever it hits an atomic token at the start of a pattern, in order to test it further. If the test result is the same as the default decision for the first interval in the pattern (in this case `separate`), the system makes a decision for the first atomic token pair only, and then continues testing the remaining tokens. This allows it to test "`bien que`" as a separate pattern, even if it determined that "`si bien que`" was separate. If, however, the decision is not the default for the first interval in the pattern, the system applies the decision to the entire pattern, and skips past the last token in the pattern for further testing. This means that if "`si bien que`" was assumed to be joined, we won't attempt to separate *bien* from *que* using the "`bien que`" pattern. This method means different solutions in the beam could each involve a different number of tokenisation decisions. In order to make them directly comparable, the score for each tokenisation solution is taken to be the geometric mean of the probabilities for each component decision.

If we now return to example (2b) from the previous section, we can see that we have met our criteria for patterns which overlap with 2 or more atomic tokens, but not for patterns which overlap with only one atomic token, as in the case of *"quitte à cause de"*. This is because we are testing the intervals between atomic tokens, and in this case, the two patterns share no intervals: the first pattern stops with the interval "`[_][à]`", and the second pattern starts with the interval "`[à][_]`". We plan to extend our system to cover such cases in the future.

The case is somewhat more complicated when two different patterns start on the same atomic token. Take for example the patterns "`plutôt que`" (coordinating conjunction) and "`plutôt que de`" (preposition introducing an infinitive). When we hit the atomic token *plutôt*, both patterns will be tested, and each pattern will get a probability for each label, `join` and `separate`. We need to consider the following possibilities: "*plutôt que de*", "*plutôt_que de*", and "*plutôt_que_de*". This is calculated as follows. Let P_1, P_2, ..., P_n be the patterns starting on a given token, ordered from shortest to longest. Let S_1, S_2, ..., S_n be their probabilities for the label `separate`, and J_1, J_2, ..., J_n be their probabilities for the label `join`. We need to consider the case π_0 where all atomic tokens are separate, and the cases π_i for $i = 1$ to 10 where P_i is joined and the remaining atomic tokens are separate. The probability for each case π is calculated by multiplying the individual probabilities compatible

with π:

$$\pi_i = \prod_{j=1}^{i} J_j \prod_{k=i+1}^{n} S_k \tag{3.1}$$

For π_0, this will simply multiply all of the probabilities for `separate`, whereas for the others, it will multiply a combination of `join` and `separate` probabilities compatible with a given tokenisation scheme. We transform these values into a probability distribution by dividing by the total sum.

3.3.3 Pos-tagging

The Talismane pos-tagger takes a sequence of tokens (forming a sentence) as input, and adds a part-of-speech tag to each of these tokens as output, along with some additional sub-specified information (lemma, morpho-syntactic details). It combines a statistical approach based on the use of a probabilistic classifier with a linguistic approach via the possibility of symbolic rules used to override the classifier. If beam propagation is applied (see section 5.5), the pos-tagger could receive multiple possible tokenisation solutions from the tokeniser, possibly with a different number of tokens. This would result in a different number of pos-tagging decisions being applied to each pos-tagging solution, one per token. In order to make these different pos-tagging solutions directly comparable as we apply a beam search, we compare solutions having reached the same character index within the sentence, and use the geometric mean of probabilities for individual pos-tagging decisions as the basis for comparison.

Talismane's architecture makes it fairly straightforward to replace the tagset. This involves several steps:

1. defining the new tagset in a file with a specific format, indicating which tags belong to open classes and which tags to closed classes.

2. mapping the morpho-syntactic categories found in the training corpus to tags in the tagset, using a file in a specific format. Note that this could be a many-to-many mapping.

3. mapping the morpho-syntactic categories found in the lexicon to tags in the tagset, using a file in a specific format. Note that this is could be a many-to-many mapping.

4. rewriting pos-tagger features and/or rules to refer to the new tagset, if any of them referred to specific tags

5. retraining the pos-tagger's statistical model

6. rewriting parser features and/or rules to refer to the new tagset, if any of them referred to specific tags

7. retraining the parser's statistical model

After pos-tagging, the pos-tagger assigns lemmas and detailed morpho-syntactic information. In the current version, this information is sub-specified, meaning it is only provided when it is found in the lexicon. Talismane's lexicon interface is described in more detail in section 4.3.2.

Since pos-tagging was used as the case study for chapter 2, we will not dwell any further on this module. The reader is referred to that chapter for a more formal definition of pos-tagging, and to section 4.4.4, page 119, for more details about the baseline feature set used by the pos-tagger.

3.3.4 Parsing

The Talismane syntax parser is a transition-based statistical dependency parser, in the tradition of the parser described by Nivre et al. [2007b], already presented in detail in section 1.2.3.

The parser takes a sequence of pos-tagged tokens (with additional sub-specified lemmas and morpho-syntactic information) as input, and generates a dependency tree as output, or a dependency forest if any nodes were left unconnected.

To recapitulate, the parser begins by transforming the sequence of pos-tagged tokens into an initial parse configuration (all tokens except for the artificial *root* on the buffer), and then applies a series of shift-reduce transitions until it reaches a terminal configuration, being defined as a configuration whose buffer is empty. A parse configuration was already defined in section 1.2.3 as follows:

- σ: a stack, or ordered sequence of pos-tagged tokens which have been partially processed
- β: a buffer, or ordered sequence of pos-tagged tokens which have not yet been processed
- Δ: a set of dependency arcs of the form *label(governor, dependent)* that have already been added
- τ: a sequence of transitions allowing us to reach the current state from an initial state

In terms of the formal machine learning framework described in chapter 2, parsing can be defined as follows:

- linguistic context x: a single parse configuration
- label y: the transition to apply to the current parse configuration, presented in table 1.9, and based on the dependency labels presented in table 1.2, table 1.3, and table 1.4
- training set $\mathcal{D}$: a corpus of text, typically in the CoNNL-X format presented in section 1.1.5

The features are thus defined against a parse configuration, and can use information from the partial dependency structures already constructed, in addition to the sequence of pos-tagged tokens underlying the configuration. At each step in the algorithm, we need to determine whether a dependency exists or not between the pos-tagged token found at top of the stack σ_0 and the one found at the top of the buffer β_0. Features are thus centered around these two tokens, as well as their existing governors and dependents, and the tokens surrounding them and between them. The full set of baseline features for the parser can be found in section 4.4.5, page 121.

```
   Input: pos-tagged sentence S, feature set F, probabilistic classifier
   Output: terminal parse configuration C
 1 C ← initial parse configuration for S;
 2 while buffer in C is not empty do
 3     featureVector ← empty list ;
 4     foreach feature in F do
 5         featureResult ← apply feature to (C);
 6         add featureResult to featureVector;
 7     end
 8     // get a probability distribution for all transitions, ordered by
           decreasing probability
 9     transitionsWithProbs ← classifier.predict(featureVector);
10     foreach (transition, probability) in transitionsWithProbs do
11         if transition valid for C then
12             C ←apply transition to C ;
13             break out of for each loop ;
14         end
15     end
16 end
17 return C;
```

Algorithm 3.1: Basic transition based parsing algorithm

3.3.4.1 Transition-based parsing algorithm

Following on from the supervised machine learning definitions given in chapter 2, and the formal definition for parsing above, we are now ready to define the basic shift-reduce parsing algorithm.

Algorithm 3.1 is very similar to the pos-tagging analysis algorithm (algorithm 2.2, page 55). The only notable difference is that, since transitions have pre-conditions, not all transitions are valid for a given parse configuration, for purely mechanical reasons. For example, in the arc-eager transition system presented in table 1.9, we cannot perform a left-arc transition if it would result in a circular dependency. We thus loop through all transitions until we reach the first valid transition on line 11. In the greedy version of the algorithm, we apply this transition to the configuration and continue, until the buffer is empty.

3.3.4.2 Measuring parser confidence

In the transition-based parsing algorithm presented above, we would like to measure the parser's confidence in each of the dependencies it has detected. This confidence measure enables us to identify high and low confidence dependencies for various reasons:

- get a rough estimate of parsing quality (on the assumption that confidence is inversely correlated to errors). This correlation is examined in section 5.3, page 142.

- get a rough estimate of corpus distance (on the assumption that average parser confidence is higher for corpora similar to the training corpus)

- filter out low-confidence dependencies when automatically constructing linguistic resources (e.g. resources injected into semi-supervised approaches). This approach is examined in section section 7.3.3, page 201.
- select dependencies most likely to be erroneous, to be submitted to the user for validation in active learning systems. This is one of our future perspectives, and has not been explored further in the present study.

Although the parser gives us a direct estimate of its confidence in each transition applied (via the probability distribution for transitions), there are two transitions, `shift` and `reduce`, which do not generate dependency arcs. We would like to somehow include the probabilities for these transitions in the dependency arc confidence measures. In Talismane, we have decided to assign a confidence to each dependency as the geometric mean of the cumulative product of individual transition probabilities since the last dependency was added[3]. In other words, whenever a `left-arc` or `right-arc` transition is applied, we take the product of probabilities for all transitions since the previous `left-arc` or `right-arc`, and assign its geometric mean as the confidence measure of the dependency arc generated.

3.3.4.3 Applying a beam search to parsing

If instead of a greedy algorithm, a beam search is applied to parsing, comparing incremental parsing solutions during parsing is not trivial, since different solutions may involve a different number of transitions. Moreover, if beam propagation has been applied from tokenisation onwards, the parser could receive as input pos-tagging solutions containing a different number of tokens. Various solutions can be imagined for comparing incremental parses, and these are detailed in the following section. The simplest solution is to compare parses one transition at a time, and this is the solution used for illustration purposes in the present section.

The beam search parsing algorithm, shown in algorithm 3.2, is again very similar to the pos-tagging beam search algorithm shown in algorithm 2.3, page 57. Rather than taking the best transition and applying it to the current configuration, we apply all valid transitions to the current configuration and add them to a heap on line 18. This heap is either the next heap (non-terminal configurations), or the final heap (terminal configurations). The heap automatically orders the configurations by decreasing score, where the score is the product of individual transition probabilities. In the next iteration, we take the top k configurations on the next heap, and apply the same analysis to each of these configurations. We continue processing until the next heap is empty, meaning that the top k configurations on the previous beam were all terminal. In the Talismane version (not shown here), we maintain multiple heaps, and add the next solution to the heap corresponding to its comparison index, which can be defined in various ways (e.g. the remaining buffer size). This method is described in more detail in the following section.

3.3.4.4 Incremental parse comparison strategies

As mentioned in the previous section, comparing incremental parse configurations in a beam is not a trivial task, since different parse solutions may apply a different number of transitions before reaching the terminal solution. The most common solution [Sagae and Lavie, 2006]

[3]Note that we could have use the cumulative product directly rather than the geometric mean - however, in testing, the correlation between confidence and correctness is much higher if we use the geometric mean

```
Input: pos-tagged sentence S, feature set F, probabilistic classifier, beam width k
Output: terminal parse configuration C
beam ← empty heap;                                   // ordered by decreasing score
C_0 ← initial parse configuration for S;
add C_0 to beam;                                     // prime the initial beam
finalBeam ← empty heap;                              // ordered by decreasing score
while beam not empty do
    i ← 0;
    nextBeam ← empty heap;                           // create next beam
    while beam is not empty AND i<k do
        C ← pop configuration from top of beam;      // next most likely analysis
        featureVector ← empty list ;
        foreach feature ∈ F do
            featureResult ← apply feature to (C);
            add featureResult to featureVector;
        end
        transitionsWithProbs ← classifier.classify(featureVector);
        foreach (transition, probability) ∈ transitionsWithProbs do
            if transition valid for C then
                C' ←apply transition to C ;
                if C' is terminal then
                    add C' to finalBeam;             // add analysis to final beam
                else
                    add C' to nextBeam;              // add analysis to next beam
                end
            end
        end
        i ← i + 1;
    end
    beam ← nextBeam;
end
return C on top of finalBeam;
```

Algorithm 3.2: Transition-based parsing algorithm with beam search

is to compare parses one transition at a time. Sagae and Lavie add an additional condition to stop as soon as they have k terminal solutions, although this would advantage shorter solution paths. Comparing parses by the number of transitions is a fairly arbitrary choice, dictated by its simplicity. In Talismane, we decided to construct multiple comparison heaps, and compare solutions using various comparison indexes. The simplest comparison index is the number of transitions, which gives the same solution as described above. Another option is to compare solutions with the same number of atomic tokens remaining in the buffer (on the assumption that we use a transition system where the buffer never grows in size). If, after applying a transition, the heap corresponding to the next solution's buffer size is already being processed (e.g. after applying a `reduce` transition, which only affects the stack), we create a new separate heap for handling the resulting solutions. This heap will only be tackled when

the top k solutions on the existing heap have been processed. All terminal parses (with an empty buffer) are assigned the same comparison index (the maximum possible integer value), thus ensuring that they will all be compared to each other when parsing has completed. Even when comparing by the number of transitions applied, we still allow all parses on the top k of any given heap to continue parsing until their buffer is empty, rather than automatically comparing the first k parses to complete. Thus, we avoid in all cases privileging shorter paths.

We tested four different parse comparison indexes:

- Transition count: this is identical to the majority approach since we compare parses having applied the same number of transitions, except that we process all heaps, rather than stopping as soon as k paths reach a terminal configuration.

- Buffer size: compare parses with the same number of items remaining on the buffer.

- Buffer and stack size: compare parses with the same number of items remaining on the buffer and stack.

- Dependency count: compare parses having constructed the same number of dependencies.

Table 3.1 shows the results for the various parse comparison strategies, in terms of labelled accuracy for all of our evaluation corpora combined. Although the differences are fairly small, the standard strategy of comparing parses having undergone the same number of transitions is clearly not optimal. The buffer size strategy seems to give the best accuracy across the board (except for beam 2, where dependency count has a slight edge). In terms of statistical significance (see section 5.1.2), the difference between the standard transition count strategy and our buffer size strategy is significant in beams 1, 2 and 5 (p-value < 0.02), but insignificant for beams 10 and 20. Indeed, there is a greater difference in accuracy for lower beams than for higher beams. This is probably because in the higher beams, regardless of the comparison strategy, the correct dependency has a higher chance of being kept in one of the various comparison heaps and therefore making it to the terminal heap, whereas in the lower beams, it may get discarded.

	Beam width				
Strategy	**1**	**2**	**5**	**10**	**20**
Transition count	87.91	88.56	88.95	89.04	89.08
Buffer size	88.05	88.63	88.98	89.05	89.09
Buffer & stack size	87.96	88.58	88.95	89.03	89.07
Dependency count	87.99	88.65	88.96	89.03	89.05

Table 3.1: Incremental parse comparison strategies, labelled accuracy at different beams

We also calculated the accuracy when stopping as soon as k parses have reached a terminal state, where k is the beam width [Sagae and Lavie, 2006]. This approach considerably lowers scores, especially at higher beams. In view of these results, we use buffer size as the incremental parse comparison strategy for the remainder of this thesis.

3.4 Formally defining features and rules

In order to open Talismane up to computational linguists, we need to find a way to allow them to define complex features and rules to their heart's content. In this section, we present the feature definition syntax used to define both features and rules in declarative configuration files. Although each module contains its own low-level feature functions, the syntax used for specifying function parameters and combining these functions into features is common to all four modules.

Indeed, the features to be defined often need to combine information in various ways. For example, in parsing, it is clear that certain pos-tags combinations are more likely to create certain types of dependencies. For example, if the top of stack σ_0 is a determiner with pos-tag `DET`, and the head of buffer β_0 is a common noun with pos-tag `NC`, the dependency *det(*β_0*,* σ_0*)* resulting from the transition *left-arc*$_{det}$ is very probable. Thus, an obvious choice for a feature is a combination of the two pos-tags. To this end, we define the concatenation operator ||. Now, given that the parser has a built-in function `PosTag(X)`, giving the pos-tag of a given element in the parse configuration, we can define this feature as:

```
PosTag(Stack[0]) || PosTag(Buffer(0))
```

Similarly, we can imagine a feature that tests whether the word on the top of stack is a verb, and already has a direct object assigned to it, so as to avoid assigning two direct objects to the same transitive verb. However, verbs in our tagset have several different tags, one per verb mood: `V` for indicative verbs, `VINF` for infinitive verbs, `VIMP` for imperative verbs, etc. All of these moods can govern a direct object. We thus need a logical "or" operator | to test for the various verb tags, a logical "and" operator & to combine these with the direct object test criterion, and an equality operator == to check the number of existing direct objects. Now, knowing that the parser has a built-in function `DepCountIf(governor, condition)`, returning the number of dependents of a given governor satisfying a given condition, and another function `DepLabel()`, returning the dependency label governing a given token, we can write this feature as:

```
PosTag(Stack[0])=="V" | PosTag(Stack[0])=="VINF" |
PosTag(Stack[0])=="VIMP" & (DepCountIf(Stack[0],
DepLabel()=="obj")==1)
```

This could be restated in English as: the pos-tag of σ_0 is an indicative verb, an infinitive verb, or an imperative verb and it has exactly 1 dependent with the direct object label. If this boolean feature returns a result of *true*, it should make it very unlikely for the system to create the dependency *obj(*σ_0*,* β_0*)* resulting from the transition *right-arc*$_{obj}$, since the verb at σ_0 already has a direct object.

The syntax thus supports certain operators listed in table 3.2, completing the operators already presented above, and certain generic functions, some of which are listed in table 3.3, enabling the linguist to define constructs such as "*if* condition *then* return result A *else* return result B".

Within each Talismane module, these generic functions and operators are used to combine a set functions specific to the module's internal mechanism, such as `DepCountIf` described above for the parser. We will introduce these little by little, as required to describe specific features used by our experiments. A full list of feature functions is given in the Talismane

user manual[4].

Operator	Result	Description
+	number	addition
-	number	subtraction
*	number	multiplication
/	number	division
%	number	modulus (remainder after division)
==	boolean	numeric, string or boolean equality
!=	boolean	numeric, string or boolean inequality
<	boolean	less than operator
>	boolean	greater than operator
<=	boolean	less than or equal to
>=	boolean	greater than or equal to
&	boolean	boolean AND
|	boolean	boolean OR
||	string	merges two or more string features by concatenating their results and adding a | in between.
(...)	n/a	grouping parenthesis
"..."	string	encloses a string literal

Table 3.2: Talismane feature syntax: operators

One generic function to note is the `OnlyTrue` function. In a standard boolean function, a *false* result will be considered as a valid result in the statistical model, and can be used to make certain decisions more probable. If the `OnlyTrue` function is applied, a *false* result is transformed to a *null*, which is simply discarded, and has no bearing on the statistical model. This allows us to include features where the value *false* wouldn't add much value and would uselessly burden training. For example, in most corpora, the vast majority of tokens don't contain a number. Thus, knowing that a token doesn't contain a number doesn't really help determine its part-of-speech. The pos-tagger defines a `Regex` function, testing whether the token matches a certain regex. This function uses the Java regex syntax[5], where `\d` represents a digit, `\D` represents a non-digit, `*` indicates 0 or more occurrences of the previous item, and `+` indicates 1 or more occurrences of the previous item. We would write our feature something like this: `OnlyTrue(Regex("\D*\d+\D*"))`, in order to exclude the class of words which don't contain numbers from the model.

Now, many systems similar to Talismane provide a much simpler feature set. However, it is one of our hypotheses that results for specific phenomena can be improved by means of complex targeted features. A series of experiments applying complex language-specific features to correctly annotate specific phenomena is given in chapter 6.

3.4.1 Using named and parametrised features

It is sometimes useful to re-use a particular feature as a component of many other features, either through result concatenation, or through logical combinations. To this end, features

[4] `http://urieli.github.io/talismane/`

[5] `http://docs.oracle.com/javase/tutorial/essential/regex/`

Function	**Type**	**Description**
ExternalResource (string name, string keyElement1, string keyElement2, ...)	string	Returns the category indicated in the named external resource for the set of key elements provided.
IfThenElse (boolean condition, any thenResult, any elseResult)	any	If condition evaluates to *true* then return one result else return another result.
IndexRange (integer from, integer to)	integer	Creates n separate features, one per index in the range going from "from" to "to".
Inverse (number)	number	Inverts a normalised double feature (whose values go from 0 to 1), replacing result x with 1-x. If the result is < 0, returns 0.
IsNull (any)	boolean	Returns *true* if a feature returns *null*, *false* otherwise.
Normalise (number feature, number minValue, number maxValue)	number	Changes a numeric feature to a value from 0 to 1, where any value $\leq$ minValue is set to 0, and any value $\geq$ maxValue is set to 1, and all other values are set to a proportional value between 0 and 1.
Not (boolean)	boolean	Performs a boolean NOT: replaces *true* with *false* and vice-versa
NullIf (boolean condition, any feature)	any	If the condition returns *true*, return *null*, else return the result of the feature provided.
OnlyTrue (boolean)	boolean	If the wrapped boolean feature returns *false*, will convert it to *null*. Useful to keep the feature vectors sparse, so that only *true* values return a result.
ToString (any)	string	Converts a non-string feature to a string.

Table 3.3: Talismane feature syntax: a subset of generic functions

in Talismane can be named, allowing them to be reused in any feature definition, and also simplifying the interpretation of any output files referring to features, as the name will be used instead of the full feature definition. In the feature configuration file, this is done by giving the features a unique name, followed by a tab, followed by the feature itself. Furthermore, parameters can be added to a named feature, making it possible to pass specific values as arguments to the feature. For example, we may want to re-use the feature testing whether or not a token is a verb through the various verbal mood pos-tags. We also want to be able to test this feature on any token in the configuration—the top of stack, the head of buffer, etc. To this end, we could redefine our feature from the previous section as follows:

```
IsVerb(X)          PosTag(X)=="V" | PosTag(X)=="VINF" | PosTag(X)=="VIMP"

VerbHasObject      IsVerb(Stack[0]) & (DepCountIf(Stack[0],
                   DepLabel()=="obj")==1)
```

We can then reuse this `IsVerb` feature, e.g. `IsVerb(Buffer[0])` to test if the head of buffer is a verb.

3.4.2 Defining feature groups to simplify combination

In NLP applications, we often try to provide useful feature combinations, since the robust numeric classifiers presented in this thesis cannot automatically select combinations with high information content. For example, when parsing, we may wish to define a full set of basic features describing some aspect of the parse configuration, such as the word form, lemma or pos-tag of the word at position β_1, just after the head of buffer. This on its own may not help us take any decision with respect to a dependency between the current σ_0 and β_0, but may well be highly informative when combined with the word forms and/or postags of σ_0 and β_0. For example, let us assume we have the phrase *"une pomme rouge et mûre"*. We reach a point in parsing with σ_0=*rouge*, β_0=*et*, and β_1=*mûre*, and are trying to decide whether *rouge* is the first conjunct of *et*. Knowing that β_1 is an adjective doesn't necessarily help us make any decision on its own, but knowing that both σ_0 and β_1 are adjectives is very important (to differentiate from *"une pomme rouge et une orange"*). Thus, we may want to systematically combine all of the basic features with (A) the pos-tags of σ_0 and β_0, and (B) the lemmas of σ_0 and β_0.

To this end, the basic features can be grouped together, by adding an additional feature group name separated by tabs from the feature name and the feature code. The following feature definitions use the parser's built-in functions `WordForm`, `Lemma`, and `PosTag`, and define a set of basic features as belonging to the `BasicFeatures` group:

```
WordFormBuffer1()  BasicFeatures     WordForm(Buffer[1])
LemmaBuffer1()     BasicFeatures     Lemma(Buffer[1])
PosTagBuffer1()    BasicFeatures     PosTag(Buffer[0])
```

We can systematically combine this basic features group with the σ_0 and β_0 pos-tag pair and lemma pair, as follows:

```
BasicFeatures_P        PosTag(Stack[0]) || PosTag(Buffer[0]) || BasicFeatures
BasicFeatures_L        Lemma(Stack[0]] || Lemma(Buffer[0]) || BasicFeatures
```

3.4.3 Features returning multiple results

Sometimes a feature may need to return multiple results for a given linguistic context. For example, we may want a feature that returns each of the different pos-tags assigned to a given word form in the lexicon.

String collection features are a special kind of feature, which evaluates to a collection of strings at runtime (instead of evaluating to a single result). They can be used anywhere a regular string feature is expected. However, in order to avoid generating cross-product of all instances of a given collection feature enclosed in the same parent feature, each collection feature is evaluated prior to the parent feature, and a single value is then inserted at a time to the parent feature. Any parent feature which refers to a string collection feature in its definition will be converted at runtime into n separate features, one per string-collection result.

3.5 He who laughs last: bypassing the model with rules

One of the primary innovations of Talismane is to allow the user to bypass the statistical model for each of the modules through the definition of **rules**. It is all fine and dandy to allow the model to take its decisions based on a statistical analysis of the training data, but sometimes the robust statistical model makes a stupid error for a really obvious case, and the computational linguist can't help but wanting to have the final word: this is where rules come in handy. Rules are applied when Talismane analyses new data, allowing the system to override the statistical model and automatically assign or exclude a certain classification to a certain phenomenon.

Rules thus add a symbolic dimension to the statistical approach. This makes it possible for Talismane to adapt to certain specific cases which are either under-represented or not represented at all in the training corpus. Indeed, the most typical use for rules is error correction: the user reviews Talismane's analysis, and discovers that certain rare phenomena are misclassified. Rather than trying to write new features and retraining the model, which is unlikely to succeed if the phenomenon is sufficiently rare, the user can add specific rules to a configuration file, and re-analyse. These rules are applied during text analysis, and can be configured differently for each analysis. Unlike post-processing corrections, rules are directly integrated into the analysis process, and affect contextual features (e.g. n-gram features) for subsequent items. Any boolean feature can be transformed into a rule: if the boolean feature returns `true` for a given context, the rule is applied, and the statistical model's decision is overturned. Rules are associated with either a positive classification (i.e. the system must assign a given classification if the feature returns `true`) or a negative classification (i.e. the system cannot assign a given classification if the feature returns `true`).

Rules can be applied to all four modules. Sentence detector rules are assimilated to filters and covered in section 3.6, and tokeniser rules are applied via regular expressions that automatically tokenise an entire matching sequence as a single token (e.g. e-mail addresses or website URLs).

Pos-tagger rules are more complex, and are configured as follows:

```
[postag|!postag]        [boolean feature]
```

Negative rules, or *constraints*, are typically used around closed classes. The following constraint, for example, does not allow the pos-tagger to classify a word as a preposition if it isn't listed as such in the lexicon, so that new prepositions cannot be "invented":

```
!P          Not(LexiconPosTag("P"))
```

Alternatively, the following constraint ensures that a word which is listed only under closed classes in the lexicon (such as *"le"* in French, listed as a determiner and object clitic) should never be classified as a common noun:

```
!NC         HasClosedClassesOnly()
```

Positive rules are typically used for very specific cases that are under-represented in the training corpus, but which the linguist feels should always result in a certain classification. For example, we can imagine a rule telling Talismane to classify as an adjective any cardinal

number, when preceded by a determiner and followed by a token which, according to the lexicon, can be classified as a noun. This rule makes use to several built-in functions specific to the pos-tagger: `History(`n`)` returning the pos-tagged token at position n with respect to the current token, where n is negative, `Offset(`n`)`, returning the token at position n with respect to the current token, where n is positive or negative, `Word(string...)`, returning `true` if the current token matches any of the words on the list, and `LexiconPosTag(token, postag)`, returning true if the token in question is listed as having the pos-tag in question in Talismane's lexicon.

```
ADJ          PosTag(History(-1)) == "DET" & Word("deux", "trois", "quatre",
             "cinq", "six", "sept", "huit", "neuf", "dix") &
             LexiconPosTag(Offset(1), "NC")
```

The English interpretation of this rule would be: assign the pos-tag "adjective" if the previous word was tagged as a determiner, the current word is either "deux", "trois", etc., and the next word is listed as a common noun in the lexicon. Parser rules are similar to pos-tagger rules, and are configured as follows:

```
[transitionCode|!transitionCode]   [boolean feature]
```

For example, if the user notices that Talismane is erroneously assigning multiple left-hand subjects to verbs, the following parser rule can prevent this from happening:

```
!LeftArc[suj]      DepCountIf(Buffer[0], DepLabel=="suj")>0
```

An English interpretation of this rule is: do not attach the top of stack as a subject of the the head of buffer if the head of buffer already has at least one dependent with the label "subject". A more complex version of the same rule can be used to allow for valid double-subject sentences such as *"La dame a-t-elle trouvé son chien ?"*, ensuring that if two subjects are assigned, one of them is a subject clitic (`CLS`) and the other one is not ():

```
!LeftArc[suj]      (DepCountIf(Buffer[0], DepLabel=="suj" &
                   PosTag=="CLS")==1 & PosTag(Stack[0])=="CLS") |
                   (DepCountIf(Buffer[0], DepLabel=="suj" &
                   PosTag!="CLS")==1 & PosTag(Stack[0])!="CLS")

!RightArc[suj]     (DepCountIf(Stack[0], DepLabel=="suj" &
                   PosTag=="CLS")==1 & PosTag(Buffer[0])=="CLS") |
                   (DepCountIf(Stack[0], DepLabel=="suj" &
                   PosTag!="CLS")==1 & PosTag(Buffer[0])!="CLS")
```

An English interpretation of the first rule above would be: do not attach the top of stack as a subject of the head of buffer, if either the head of buffer already has a clitic subject and the top of stack is also a clitic, or the head of buffer already has a non-clitic subject and the top of stack is also a non-clitic. The two rules in the above example could be replaced by a single rule if we could replace `LeftArc` and `RightArc` by `Arc`, and `Buffer[0]` by `Governor` (in the case of a `LeftArc`) or `Dependent` (in the case of a `RightArc`), and and `Stack[0]` by `Dependent` (in the case of a `LeftArc`) or `Governor` (in the case of a `RightArc`). It is planned to add such functionality in the future to ease rule writing. Regardless, rules, and the interactions between them, are fairly cumbersome to maintain, so that the ease with which they can

be added to the system is offset by lower long-term maintainability. We present the baseline rules included in the default French implementation in section 4.5, page 125, and experiments using rules to tackle specific phenomena in section 6.5, page section 6.5.

3.6 Filtering the raw text for analysis

Sometimes, the raw input text cannot be analysed as is. For example, in XML, it may only be necessary to parse text contained between certain tags, and to skip certain tags inside that text. Certain tags might mark sentence breaks, and others are simply formatting tags within a sentence. Talismane filters are used to prepare the raw input text for the first step in the linguistic processing chain (usually sentence detection). They allow us to indicate sections in input text that need to be parsed and other sections that should be skipped or replaced. Furthermore, they allow us to include certain markers from the input directly in the output, without processing.

Filter type	**Regular expression**	**Task**
SKIP	`<skip>.*</skip>`	Skip the XML tag `<skip>` and its contents, so that it is excluded from Talismane's output.
INCLUDE	`<b>(.*)</b>`	Include the text inside the XML tag `<b>`, but not the tag itself
START	`<text>`	Start processing just after the XML tag `<text>`
STOP	`</text>`	Stop processing just before the XML tag `</text>`
OUTPUT	`<mark>(.*)</mark>`	Mark the text inside the XML tag `<mark>` for output along with Talismane's analysis, excluding the marker itself
SKIP, SEN-TENCE_BREAK	`<p>`	Mark the XML tag `<p>` as a sentence break, and skip it.
REPLACE	`é→é`	Replace any occurrence of `é` by `é`

Table 3.4: Examples of Talismane filters

Filters in Talismane are defined using regular expressions. Examples of typical filters are shown in table 3.4. Talismane has been designed so that, after the filters have been applied, we keep a trace of a token's original position in the raw text (column number + row number), and can include this information in Talismane's output. This can be critical in applications, such as candidate term extraction, in which the user needs to review the context of a given phrase before taking a decision for a particular candidate term.

Note that `SKIP` and `INCLUDE` can be replaced by `START` and `STOP`, which is especially advantageous when the portion of text to be skipped is large, and might go beyond Talismane's configurable input buffer size.

3.7 Comparison to similar projects

Talismane differs from the parser in Nivre et al. [2007b], also used in the benchmarking study for French by Candito et al. [2010b], in the following ways:

- it extends parsing from a purely deterministic system to a beam-search approach, with different startegies for comparing incremental parses
- it places parsing organically within the context of full syntax analysis, from raw text to parse trees. Specifically, the final pos-tagging beam, retaining the most ambiguous cases from tokenising and pos-tagging, can be propagated to the parser, thus allowing the parser to resolve ambiguities left open by the lower levels of abstraction.
- it introduces a very rich syntax for feature description, allowing the definition of ad-hoc features specific to a given language and/or phenomenon
- it introduces symbolic rules as a means for overriding the statistical model's decisions
- it provides a measure of the parser's confidence in the dependencies constructed as optional output
- it maintains traceability of each token's column/row position in the raw text, even after applying filters to parse annotated text (e.g. XML)
- it incorporates both standard (local) transition-based parsing and global learning parsing as training/analysis options. Global learning is not covered in the present thesis.

Other systems extending transition-based parsing to non-deterministic case via a beam search are described by Sagae and Lavie [2006], Johansson and Nugues [2006, 2007], Zhang and Clark [2009], Zhang and Nivre [2011, 2012], Bohnet and Nivre [2012]. With respect to the beam-search, Talismane differs from these systems mainly in that the beam search crosses several levels of analysis (tokenising, pos-tagging, parsing), and in the method via which the system determines which parses to compare while running the algorithm: these studies maintain a single heap and privilege shortest path solutions, whereas Talismane constructs multiple comparison heaps, allowing path-length neutral comparison. Talismane is also the first system to apply these methods to French.

Zhang and Clark [2008] extended both the graph-based and the transition-based systems using a beam search, and apply an identical perceptron-based learning algorithm to both systems. Finally, they create a highly competitive combined parser which again uses perceptron methods to select the best parse from both parsers' beams. Within the present thesis, we seek instead to find ways to add information to a transition-based parser at different points in the algorithm, so as to allow it to approach the accuracy of graph-based parsers while maintaining linear processing time.

Talismane is similar in some ways to the Zpar system[6] described in Zhang and Nivre [2011], which also supports beam search and global learning, as well as supporting tokenisation (for Chinese) and pos-tagging. However, Talismane places more emphasis on language-specific features via its rich feature definition syntax, incorporates symbolic rules, and integrates the

[6] `http://www.sourceforge.net/projects/zpar`

various modules more tightly by enabling the propagation of the solution beam from one module to the next.

Bohnet and Nivre [2012] integrate pos-tagging and parsing even more tightly in their mate-tools system[7], by performing them in a single step, and report improved state-of-the-art for all languages. At first this seems counter-intuitive, since pos-tagging typically provides 97%+ accuracy, while parsing accuracy hovers around 90%. Thus, performing pos-tagging ahead of time gives us more information when parsing: we know the pos-tags of all tokens to the left and the right of the current σ_0 and β_0. However, their positive results indicate that this is not necessarily a valid assumption. In Talismane, we accommodate this idea somewhat by propagating the pos-tagger beam to the parser, thus allowing the parser to decide between ambiguities left open by the pos-tagger. However, their tighter integration has the advantage of not relying on a wider beam to consider different solutions, and would need to be explored further.

3.8 Discussion

In this chapter, we set out to describe the mechanism behind the four Talismane modules in sufficient detail to enable a linguist to inject linguistic knowledge in a way that is useful to each module's decision-making process. We described each module in terms of the formal supervised machine learning framework presented in chapter 2. We then described the actual configuration mechanism used to inject this knowledge, in the form of features, rules and filters.

In the following chapter, we will examine the specific sources of linguistic knowledge available for French, and show how to incorporate this knowledge into the system via the baseline features for each module. We will then be ready to begin describing our experiments, primarily aiming to prove that this expressive configuration mechanism actually serves a purpose: that specific features and rules can improve results for specifically targeted phenomena.

[7] `http://code.google.com/p/mate-tools/`

Chapter 4

Incorporating linguistic knowledge

In the previous chapters, we covered our dependency parsing annotation scheme, the supervised machine algorithms needed to produce this scheme, and the specific implementation of these algorithms within Talismane. We now turn to a critical question in NLP from a linguist's perspective: how do we incorporate linguistic knowledge into these algorithms to help them in their decision-making process?

There are several levels in which this knowledge is incorporated:

- Training corpora, defining the scope of basic material from which the system will attempt to learn useful generalisations. These are presented in section 4.1, along with the specific training corpora used by Talismane for its French implementation.
- Evaluation corpora, allowing us to judge the extent to which a model built from a particular training corpus and configuration is generalisable to other genres and/or domains. They are presented in section 4.2, along with the specific evaluation corpora used in our experiments.
- External resources, containing more or less comprehensive knowledge regarding a particular aspect of a language. We discuss these in section 4.3, describing the general types of external resources that can be used, the manner in which they are incorporated, and the specific resources actually used in our experiments.
- Features, which are the linguist's attempt at capturing the most informative and generalisable aspects of a linguistic context with respect to the decision to be made. In section 4.4, we present baseline features for each Talismane module, based on the expressive feature definition syntax described in the previous chapter.
- Rules, enabling the linguist to react to mistakes made by the statistical tool by overriding its decisions in specific cases. Rules are one aspect of Talismane which is not typically available in other statistical systems. The baseline rules used to override the statistical model's decisions for Talimane's default implementation are presented in section 4.5.

4.1 Training corpora

The first source of linguistic knowledge for any supervised machine learning system is the corpus on which the system is trained. Choosing the training corpus to be as representative

as possible of the type of language we wish to analyse is critical, and involves many linguistic criteria. A technical corpus, for example, is likely to be limited in both is vocabulary and in its acceptable syntactic structures. An edited journalistic corpus from a national newspaper concentrating on politics and economy will again have a limited vocabulary, and may be entirely devoid of constructs more common in less formal writing, such as the second person singular. Having selected a corpus, a large amount of linguistic knowledge is then injected during the annotation process itself: at a high-level, via the choice of annotation scheme, and at a lower level, via the annotation guide which gives criteria for annotating difficult cases.

4.1.1 Errare humanum est: Annotation reliability

Training corpora are never perfect, but the degree of their reliability is difficult to measure. One way of attempting to measure reliability is by comparing annotations for identical sections by different annotators.

There are several reasons behind differing annotations:

- Human error: by far the most common reason for disagreement in annotations is simple human error. The majority of corpora nowadays are annotated by asking people to correct the automatic annotations produced by a machine. Many errors are introduced by simply overlooking an error made by the machine in the first place.

- Incomplete annotation guide: a large number of specific cases can only be consistently annotated by human annotators if the annotation guide gives proper guidance with easy-to-follow criteria. This is true even for cases where no semantic ambiguity is possible, but where the correct choice is not clear, e.g. the governor for *pourquoi* in "*C'est la raison pourquoi je vous écris*" ("It's the reason why I'm writing you"). Is the governor *est*, *raison* or *écris* ? There is no inherent ambiguity in the linguistic construct itself, in that nothing semantic in the construct would make us choose the main verb in one case and the subordinate verb in another. And yet, unless the annotation guide gives clear instructions for this construct, annotators are at best likely to annotate it consistently in their own annotations but inconsistently to the others, and at worse will annotate it at random. Another obvious case is titles, which represent proper nouns, but contain other parts of speech, e.g. "*Il a lu La Peste de Camus*" ("He read La Peste by Camus"). Should anything in the title be marked as `NPP`? Again, there is no inherent ambiguity to the construct, but unless the guide says otherwise, one annotator may annotate *La/DET Peste/NC* and another *La/NPP Peste/NPP*, without either one being clearly right or wrong.

- Annotation class granularity: in some cases, the categories from which we need to choose are either too specific, too general or too vague to allow for a clear choice. For example, the category `ET` (foreign word) clearly overlaps with other categories, depending on the role played by the foreign word in the sentence, and puts into question the usefulness of this category in the first place. An interesting case in parsing is the border between prepositional arguments (`de_obj`, `a_obj`, `p_obj`) and adjuncts (`mod`) which, as highlighted in Fabre et al. [2008], is a continuum rather than a clear-cut dichotomy. If we take *"voter pour moi"*, *"acheter pour moi"*, *"venir pour me voir"*, *"mourir pour des idées"* and *"dormir pour oublier"*, which are arguments and which are adjuncts?

- Undecidable ambiguity. Finally, there are some rare cases of undecidable ambiguity. A sentence such as "I saw the man with the telescope" may be undecidable out of context, but is almost never ambiguous in context, depending on the semantic levels to which the annotators have access. Sometimes however, especially in unedited text with spelling and grammar mistakes, it may be very difficult to elucidate the author's semantic intention for specific points.

In view of these possibilities for mismatches between two humans annotating the same corpus, it would be naive to assume a computer system could do better at imitating a human annotation than two humans among themselves. This brings us to the measurement of inter-annotator agreement, which, given an identical corpus that has been annotated by two or more annotators without consultation among themselves, provides an indication of the level of agreement between the annotators, and therefore of the inherent difficulty of the annotation task. These statistics, highly useful as a quality indicator for an annotated corpus, nevertheless cannot tell us whether the annotation task's difficulty is due to the level of expertise required to correctly annotate, the quality and comprehensiveness of the annotation guide, the inherent ambiguity within the task which is not resolvable by the annotation classes provided, or the annotator's personality.

The most common measurement used for inter-annotator agreement of classification tasks is Cohen's Kappa [Carletta, 1996], which attempts to take into account the likelihood of two annotators agreeing by random chance. In this statistic, neither annotation is assumed to be a reference: we are simply comparing the level of agreement between them. This is the statistic we use in the present study.

In the case of identification tasks, e.g. identifying all of the named entities in a given text, the average f-score is used instead. In this case, we alternatively assume each annotator is a reference (let's call him annotator A), and calculate annotator B's precision (what portion of annotator B's entities correspond to those identified by annotator A) and recall (what portion of annotator A's entities were found by annotator B) with respect to the reference. We then take the average f-score of both annotators. This method can also be used for syntax consituency annotation, where each annotator identifies the beginning and end of phrases within the text.

4.1.2 French Treebank

The French Treebank [Abeillé et al., 2003], hereafter FTB, is an annotated journalistic corpus comprised of extracts from the newspaper *Le Monde* of 1986 to 1993, containing 21,562 sentences and 664,904 tokens including punctuation (605,277 when compound words are merged into a single token). Annotations include morpho-syntactic categories, lemmas, and phrase consituent structures. The treebank was subsequently enriched [Abeillé and Barrier, 2004] with annotations for the verbal argument function of phrases (e.g. subject, direct object), covering about half of the corpus for the version received in 2009 (10,097 sentences, 318,778 tokens, 289,901 when compounds are merged). The corpus is accompanied by three detailed annotation guides [Abeillé and Clément, 2006, Abeillé et al., 2004, Abeillé, 2004], giving instructions to the correctors of the initial automatic annotation.

```xml
<SENT nb="20">
  <NP fct="SUJ">
    <w cat="N" lemma="m." mph="ms" subcat="C">M.</w>
    <w cat="N" lemma="Teulade" mph="ms" subcat="P">Teulade</w>
  </NP>
  <VN>
    <w cat="V" lemma="pouvoir" mph="P3s" subcat="">peut</w>
  </VN>
  <w cat="PONCT" lemma="," subcat="W">,</w>
  <w compound="yes" cat="ADV" lemma="à juste titre">
    <w catint="P">à</w>
    <w catint="A">juste</w>
    <w catint="N">titre</w>
  </w>
  <w cat="PONCT" lemma="," subcat="W">,</w>
  <VPinf fct="OBJ">
    <VN>
      <w cat="V" lemma="considérer" mph="W" subcat="">considérer</w>
    </VN>
    <Ssub fct="OBJ">
      <w cat="C" lemma="que" subcat="S">que</w>
      <w cat="PONCT" lemma="\&quot;" subcat="W">\&quot;</w>
      <NP fct="SUJ">
        <w cat="D" lemma="le" mph="fs" subcat="def">la</w>
        <w cat="N" lemma="crédibilité" mph="fs" subcat="C">crédibilité</w>
        <PP>
          <w cat="P" lemma="de">du</w>
          <NP>
            <w cat="D" lemma="le" mph="ms" subcat="def"></w>
            <w cat="N" lemma="système" mph="ms" subcat="C">système</w>
            <AP>
              <w cat="A" lemma="conventionnel" mph="ms" subcat="qual">conventionnel</w>
            </AP>
          </NP>
        </PP>
      </NP>
      <VN>
        <w cat="V" lemma="être" mph="P3s" subcat="">est</w>
      </VN>
      <w compound="yes" cat="ADV" lemma="en jeu">
        <w catint="P">en</w>
        <w catint="N">jeu</w>
      </w>
      <w cat="PONCT" lemma="\&quot;" subcat="W">\&quot;</w>
    </Ssub>
  </VPInf>
  <w cat="PONCT" lemma="." subcat="S">.</w>
</SENT>
```

code 1: XML sample from the French Treebank

Listing 1 shows an XML sample from the FTB for the following sentence: *"M. Teulade*

peut, à juste titre, considérer que « la crédibilité du système conventionnel est en jeu ».” (“Mr. Teulade can consider, and rightly so, that the conventional system’s credibility is at stake”). Each word is assigned a grammatical category, sub-category, and additional morpho-syntactic details (tense, mood, person, gender, number, possessor number), as well as a lemma. The exception is compound word components, to which only the main grammatical category is assigned. Empty tokens are inserted after all agglutinated forms, as in the case of *“du”* above, representing a combination of the preposition *de* and the determiner *le*.

The corpus has several weaknesses as a training corpus for a general purpose French syntax analyser. First of all, it has no published statistics on inter-annotator agreement. Thus, we have no clear measurement of the corpus’ internal consistency or quality.

In terms of corpus content, the specialised nature of this journalistic corpus (politics, economy, finance) is clearly outlined in table 4.1, showing the 30 most frequent common noun lemmas. Indeed, none of the publications on the French Treebank indicate which criteria were used to select the extracts in the first place. The corpus is thus representative of well-edited journalistic French corresponding to these domains, but not easily transposable to other genres or domains.

Furthermore, the corpus contains a very high proportion of compound words covering all grammatical categories. All in all, 14% of the tokens in the corpus are contained in a compound word. The annotation guide’s criteria for retaining compound words seem insufficient for making a consistent decision in many cases. From a total of 34,476 compound words, the largest category is common nouns (10,013), followed by adverbs (5,999), prepositions (5,515), proper nouns (5,039) and cardinal determiners (3,843). The closed categories such as prepositions are not problematic. Most compound adverbs are fairly generic as well, and could apply to any text genre (e.g., *d’abord, d’ailleurs, en effet, en revanche, sans doute*). However, certain adverbs are far more questionable: *à l’eau, à l’étranger, au gouvernement, dans l’air, dans le besoin, sur le terrain, en développement, purement et simplement.* In view of such a list, it would be difficult to define a criterion (other than the number of constituent words, or the corpus-dependent criterion of frequency) for the exclusion of any prepositional phrase acting as an adverbial.

The case of compound common nouns is even more problematic. The ten most common compounds are: *chiffre d’affaires, premier ministre, taux d’intérêt, pouvoirs publics, directeur général, secrétaire général, conseil d’administration, projet de loi, conseil des ministres, main-d’oeuvre.* They are all highly specific to political or economic French, and tend to decompose both syntactically and, to a great extent, semantically. Their status as compounds is however clear if we use criteria such as the difficulty of insertion (*directeur antérieur* général*) and lexical substitution (*principal* ministre*). In the corpus, there are no occurrences as non-compounds, although one could of course imagine a non-compound case such as “Il est le premier ministre à avoir reconnu cette erreur”. However, the compound nouns in the corpus also contain over 2,000 hapax (e.g. *articles de toilette, bateaux de pêche, champ de pétrole, crise économique et financière, hélicoptère de combat, jeunes femmes, langues étrangères, nouveaux pays industrialisés, restaurants gastronomiques*). The line between compound words and simple colocations becomes fuzzy, and it is difficult to imagine a hard and fast criterion which would allow us to distinguish *“articles de toilette”* or *“champ de pétrole”* as compound words from *“articles de correspondance”*, *“articles de rangement”*, *“champ de vision”* or *“champ de compétence”*, all of which exist as non-compound phrases in the corpus.

In the case of many compounds such as nouns and adverbs, the difficulty of defining clear criteria for compound identification makes it unrealistic to expect a syntax analyser to recog-

Lemma	**Translation**	**Count**
%	%	3554
franc	French Franc	1490
M.	Mr.	1209
entreprise	company	808
groupe	group	779
pays	country	779
marché	market	588
monsieur	Mr.	537
gouvernement	government	535
société	company/society	535
prix	price	529
président	president	493
emploi	employment/job	457
dollar	Dollar	407
état	state	382
production	production	369
économie	economy	344
accord	agreement	342
salarié	employee	330
travail	work	324
capital	capital	323
activité	activity	319
taux	rate	316
baisse	decrease	314
banque	bank	309
secteur	sector	303
fin	end	303
politique	policy	292
action	stock/action	290
mesure	measurement	284

Table 4.1: Top 30 common nouns in the French Treebank

nise these compounds automatically. In order to simplify evaluation, as well as generalising analysis to other domains, I have applied the following modifications to the FTB:

- Compound words: only closed category compounds and a short list of generic adverbial expressions and others without syntactic compositionality have been retained as compounds. In order to avoid losing information, compound words that were not retained are simply marked by changing the currently redundant attribute `compound="yes"` to `compound="no"`, while retaining the compound components in all cases.

- Compound components: Full morpho-syntactic information has been automatically assigned to all compound components using the LeFFF (see section 4.3.3), and manually corrected when any ambiguity existed. This concerned top-level part-of-speech tags for 5,000 components with no grammatical information at all, and grammatical sub-categories as well as additional morpho-syntactic characteristics for all 94,000 components.

In keeping with the FTB distribution license, a compound-modified FTB corpus has been sent to Anne Abeillé's team, and can be requested from them.

Because of the impossibility of handling split compounds in Talismane (see section 3.3.2, 80), we decided to analyse the list of split compounds within the FTB. These represent only 76 cases for all compound categories combined (0.22%). The most typical cases are adverbs with intensifiers (e.g. *à **plus** long terme*, *en **grande** partie*), prepositions with inserted adverbs (e.g. *à cause **notamment** de*, *par rapport **non seulement** à*), common nouns with adjectival modifiers (e.g. *ministre **tchécoslovaque** de l'intérieur*) and various modified verbal expressions (e.g. *ne met **pas forcément** en cause*, *va **d'ailleurs** bon train*). There is also one interesting case of a compound clitic split by punctuation, in the sentence: *"Un public que **l'« on** ne changera pas par décret », comme le dit. . . "*. Of these, since we have not retained compounds for nouns, verbs and most prepositional adverbials, we are only concerned with the closed category cases: prepositions (15 cases) and the single split clitic. These prepositional compounds are all syntactically compositional as prepositional phrases, and we don't foresee a problem handling them as a sequence of separate components.

Regarding the grammatical categories and sub-categories, Talismane supports the use of a configuration file to map the categories, sub-categories and additional morpho-syntactic details in the FTB training corpus to the the tagset being used. This file can be found in the Talismane source code.

Despite the various issues outlined in this section, the FTB is the only syntactically annotated corpus of French of sufficient size to be useful for supervised machine learning tasks, and has, either directly or through a derivative work, been used to train all of the Talismane modules in the default French implementation. When selecting the baseline features, we divide the FTB into training (80%), development (10%) and test (10%) corpora for tuning and evaluating the pos-tagger. For the sentence detector and tokeniser, because of the sparsity of phenomena being tested, we systematically use 10-fold cross validation.

4.1.3 French Treebank converted to dependencies

The FTB's annotations are directly usable by a phrase constituent structure parser, but not by a dependency parser such as Talismane. Candito et al. [2010a] thus transformed the FTB into dependency structures, hereafter FTBDep. The latter corpus is based on the

1	M.	m.	N	NC	g=m|n=s|s=c	3	suj
2	Teulade	Teulade	N	NPP	g=m|n=s|s=p	1	mod
3	peut	pouvoir	V	V	m=ind|n=s|p=3|t=pst	0	root
4	,	,	PONCT	PONCT	s=w	3	ponct
5	à_juste_titre	à_juste_titre	ADV	ADV	_	3	mod
6	,	,	PONCT	PONCT	s=w	3	ponct
7	considérer	considérer	V	VINF	m=inf	3	obj
8	que	que	C	CS	s=s	7	obj
9	"	"	PONCT	PONCT	s=w	15	ponct
10	la	le	D	DET	g=f|n=s|s=def	11	det
11	crédibilité	crédibilité	N	NC	g=f|n=s|s=c	15	suj
12	du	de	P+D	P+D	s=def	11	dep
13	système	système	N	NC	g=m|n=s|s=c	12	obj
14	conventionnel	conventionnel	A	ADJ	g=m|n=s|s=qual	13	mod
15	est	être	V	V	m=ind|n=s|p=3|t=pst	8	obj
16	en_jeu	en_jeu	ADV	ADV	_	15	mod
17	"	"	PONCT	PONCT	s=w	15	ponct
18	.	.	PONCT	PONCT	s=s	3	ponct

Table 4.2: Example sentence from the FTBDep corpus

portion of the FTB that was annotated with verbal argument functions. The authors give the total size as 12,531 sentences (339,522 tokens with all compounds merged). Since FTBDep is specifically designed as a training corpus, it has several advantages, such as a strict versioning scheme, allowing publications to ensure they are referring to the identical training material. The authors used heuristic rules to decompose compounds with syntactic regularity, and to determine the head for each phrase. Since the original corpus was in constituency tree structure, the conversion to a dependency structure resulted in a fully projective corpus with no crossed dependency arcs. A small portion of the corpus was then manually annotated for non-projective structures, including a corresponding annotation guide [Candito et al., 2011b]. When comparing the automatic and manual annotations, the authors found a difference of 1.22% in the UAS (unlabeled attachment score), giving an indicator of the percentage of non-projective structures in French. The final corpus is available upon request, and is provided in CoNNL-X format. An example of FTBDep annotation for the sentence from listing 1 is given in table 4.2. The columns correspond to those given in section 1.1.5, page 35.

This thesis uses FTBDep version 6 exactly as is for parser training/evaluation. We kept the same training, evaluation and test corpora divisions to keep our results directly comparable to similar studies.

4.2 Evaluation corpora

The specialised nature of the FTB and FTBDep training corpora in terms of both genre (edited journalistic text) and domain (political, financial, economic) cast into doubt the ability of any parser trained on these corpora to handle out-of-domain or out-of-genre text. In order to test parser performance on such texts, several smaller evaluation corpora are used in this

Corpus	Annotations A vs. B	Annotation A vs. ref	Annotation B vs. ref
FrWiki	83.96	91.59	88.64
Europarl	90.14	94.20	92.26
EstRépu	90.45	94.22	93.72

Table 4.3: Sequoia corpus interannotator agreement (f-score average) from Candito et al. [2012]

thesis.

4.2.1 Sequoia

The Sequoia corpus [Candito et al., 2012] is a freely available French corpus annotated for syntax, and consisting of 3,204 sentences (69,246 tokens) taken from the French Europarl, the regional newspaper *l'Est Républicain* (*EstRépu* below), the French Wikipedia (*FrWiki* below), and translated documents from the European Medicines Agency (*EMEA* below), separated into a dev and test corpus for out-of-domain corpus evaluation. It is available in both constituency and dependency formats, and follows the FTB annotation guidelines except for compound words. It uses the same simplified tagset from Crabbé and Candito [2008] as the present thesis. The dependency version was generated automatically from the constituency version using the same automatic converter as FTBDep, and is therefore entirely projective by default. However, the authors used lexical cues to identify and correctly annotate certain extraction phenomena centered around wh-words (*que*, *dont*, etc.) and *en* [Candito and Seddah, 2012]. This allowed them to identify a small number of non-projective phenomena: a total of 42 for 69,238 tokens, or about 0.06% of all dependency arcs.

The Sequoia corpus comes with inter-annotator agreement scores, indicated as an average f-score for the identified constituency structures, and reproduced in table 4.3. These include agreement scores between the two annotators, and between each annotator and a reference created by merging and validating the two annotations. The overall inter-annotator agreement seems to place a fairly low upper bound on parser accuracy: no higher than about 90.5% if we take the A vs. B scores as an indicator, since the reference corpus was constructed by merging the two annotations, and presumably only reviewing those areas where the two annotators differ, thus encouraging a higher overall agreement. The relatively equivalent scores between each annotator and the reference imply that each annotator generated a different set of errors.

4.2.2 Wikipedia.fr discussion pages

The FrWikiDisc corpus is a freely available corpus annotated for tokens, pos-tags and syntactic dependencies within the context of the present thesis, using the Brat annotation tool [Stenetorp et al., 2012]. The corpus was taken from the discussion pages of certain articles on the French Wikipedia, selected because we felt the topics would engender considerable debate: the Iraq War, the Algerian War, and genetically modified organisms. On these discussion pages, users communicate on suggested or past modifications to the related encyclopedia article, often in a heated and fairly informal manner. Within these pages, we manually excluded any extracts copied directly from the encyclopedic pages, in order to unify the corpus genre to discussion only. From casual observation, the contributors seem to have a fairly high level of education, and alternate long and complex arguments with shorter exclamatory

Test	**Cohen's Kappa**
Annotator A vs. Annotator B	0.864
Annotator A vs. Original	0.789
Annotator B vs. Original	0.835
Annotator A vs. Reference	0.971
Annotator B vs. Reference	0.880

Table 4.4: FrWikiDisc corpus inter-annotator agreement (Cohen's kappa)

remarks. The corpus consists of approximately 300 sentences and 10,000 tokens, for which two annotators corrected Talismane's original automatic annotation. A third of the corpus was annotated by each annotator separately, and a third by both annotators to estimate inter-annotator agreement. For this last third, the annotations for the two annotators were merged and corrected after consultation in order to generate a reference.

Table 4.4 shows inter-annotator agreement for this last third of the corpus in terms of Cohen's kappa. As can be seen, the A vs. B agreement is on par with agreement scores for Sequoia. It is assumed that higher scores would be achieved if the same two annotators continued to work on other corpora. It can also be seen that the merged corrected corpus uses far more annotations from annotator A than from annotator B. However, the inter-annotator agreement for A vs. B is considerably higher than each annotator's individual agreement with the original Talismane annotations, indicating some degree of consistency in interpreting the annotation guide.

The annotation guide itself is based on Candito et al. [2011b], and has been extended to cover any ambiguous cases encountered in the corpus. A copy of the extended guide can be downloaded from the Talismane home page[1]. We included all of the manual dependency labels from table 1.4, and, unlike the original FTBDep and Sequoia, we annotated all non-projective dependency arcs as well, made simpler by the use of a dependency model directly, rather than converting a constituency model to a dependency model. We chose not to annotate governors for punctuation (which are generally arbitrary), with the exception of coordinating commas.

Table 4.5 shows the differences in certain characteristics between the FrWikiDisc corpus and the FTBDep training and development corpus. On average, sentences are longer in FrWikiDisc, and are far less equal in length: standard deviation is almost doubled. With respect to syntax tree depth (dependency distance in hops from the root token to a given token in the sentence), FrWikiDisc is considerably deeper both in terms of the mean for all tokens (excluding punctuation), and of the mean of the maximum depth for each sentence. As can be expected, the % of unknown words is significantly higher in the wiki discussion pages than in the FTBDep development corpus. Here, unknown words refer to words that are not contained in the FTBDep training corpus. This holds true for the lexicon of distinct raw inflected forms (including punctuation), for the full list of raw tokens, for the lexicon of distinct alphanumeric forms converted to lowercase (and excluding punctuation), and for the full list of alphanumeric tokens converted to lowercase. There are significant differences in the proportions for certain pos-tags (ignoring pos-tags with less than 100 occurrences in the smallest corpus), which give some indication of the differences in style between the two corpora: in FrWikiDisc, there are almost three times as many subject and object clitics for a roughly equivalent proportion of

[1] `http://redac.univ-tlse2.fr/applications/talismane.html`

	FrWikiDisc	**FTBDep dev**	**FTBDep train**
Sentence count	301	1235	9873
Sentence length in words (mean)	34.41	30.56	29.14
Sentence length in words (deviation)	28.50	16.10	16.57
Average syntax depth	5.06	3.86	3.78
Max syntax depth (sentence mean)	7.10	5.59	5.45
Data for inflected forms (incl. punctuation)			
Lexicon size	2371	7220	24090
% unknown	28.76	22.59	—
Occurrence count	10055	36508	277833
% unknown	11.06	5.19	—
Data for forms converted to lowercase (excl. punctuation)			
Lexicon size	2207	6902	22462
% unknown	26.37	21.67	—
Occurrence count	8633	31473	238963
% unknown	11.37	5.58	—
Selected part-of-speech data			
% common noun	16.50	22.17	21.95
% proper noun	3.78	4.09	3.98
% adverb	7.68	4.29	4.70
% verb	6.94	5.53	5.55
% subject clitic	3.40	1.23	1.16
% object clitic	1.22	0.49	0.40
% subordinating conjunction	1.93	1.00	0.90
% coordinating conjunction	2.92	2.14	2.24
% relative pronoun	1.46	0.93	1.01

Table 4.5: FrWikiDisc corpus characteristics

verbs, probably indicative of a more personal, conversational style. There are also far more conjunctions and relative pronouns, indicative of a certain degree of syntactic complexity. The subordinating conjunctions, however, are due to a preponderance of constructions indicating personal opinion, e.g. *J'ajoute que...*, *Je vous indique que...*, *J'avoue que...*, etc. There are far more adverbs: typically connective adverbs at sentence start (e.g. *ensuite, incidemment*), emphatic adverbs (e.g. *complètement*) or negative adverbs. The lower percentage of common nouns is probably simply a question of mathematics, since they take up the space left when other parts-of-speech have been removed.

Regarding non-projectivity, 30 non-projective structures were identified in the corpus, amounting to about 0.3% of all arcs. This is proportionally 5 times higher than the non-projective arcs annotated in the Sequoia corpus. The break up is shown in table 4.6. Comparative constructions account for 8 of the 30, *en* clitic extraction for 6, and relative pronouns dependent on the object or 2nd verb in complex verbal structures (called "wh-object" above) for 4. The remaining 12 cases are what we would call "voluntary" non-projective structures, where the non-projectivity could have easily been avoided by reordering the sentence phrases, or by re-introducing an elliptical argument in coordination. Of these, only the *en*

Description	Example	Count
comparative	*. . . plus nombreuse que . . .*	8
en-clitic	*. . . en font partie*	6
wh-object	*ce que je veux exprimer*	4
voluntary	*. . . en refaisant une modif à 16:39 que je réannule*	12

Table 4.6: FrWikiDisc corpus non-projective arc types

Label	Description	Count
aff_moyen	"middle" reflexive construction	0
arg_comp	comparative argument	8
arg_cons	consecutive argument	1
mod_cleft	cleft relative	6
mod_loc	locative adjuncts	23
p_obj_loc	locative verbal arguments	20
p_obj_agt	passive or causative agent	32
suj_impers	the impersonal *il*	65

Table 4.7: FrWikiDisc corpus manual dependency label count

clitics and wh-objects would have been annotated in Sequoia, accounting for only 1/3rd of all non-projective structures in FrWikiDisc, and explaining the proportional difference in non-projectivity between the two corpora.

Finally, we look at the manual dependency labels defined in table 1.4. Table 4.7 gives counts for each manual dependency label (out of 8661 dependencies, since punctuation was not annotated with a governor). The only manual label to exceed 0.5% is `suj_impers`, the impersonal *il*.

4.2.3 Unannotated corpora

In addition to annotated evaluation corpora, we will often make use of much larger unannotated corpora within the present thesis. This will allow us to run experiments which analyse a given corpus with and without a certain modification (e.g. a new feature), and review the differences between the two analyses. For this type of experiment, we have selected five unannotated 1 million word extracts from the following corpora:

- Est Républicain: the regional newspaper Est Républicain from the year 2003, available on the CNRTL website[2]
- Leximedia: a collection of newspaper articles concerning the 2007 French presidential campaign, from the French national newspapers Le Monde, Libération and Le Figaro, prepared by the CLLE-ERSS laboratory [3]
- Frantext: French literary texts from the 20th century[4]

[2] `http://www.cnrtl.fr/corpus/estrepublicain/`
[3] `http://redac.univ-tlse2.fr/applications/leximedia2007.html`
[4] `http://www.frantext.fr`

- Revues.org: a collection of scientific articles in the human and social sciences[5]
- Wikipedia.fr: a dump from the 2008 French Wikipedia, cleaned and reformatted into simple text format by the CLLE-ERSS laboratory[6]

In some cases, we will also use full versions of these unannotated corpora to construct linguistic resources.

4.3 External Resources

In addition to training and evaluation corpora, statistical NLP systems often make use of external resources to attempt to complement the linguistic coverage of the necessarily limited training material. In our cases, all external resources used are lexical, in that they provide various characteristics of specific inflected forms or lemmas. After a general discussion of how to incorporate external resources into the system, we will describe certain resources used in our experiments.

4.3.1 Generalising features using external resources

We begin this section with an example: let us assume the training corpus contained the tokens "*mangez*" and "*mangiez*" but not "*mangeons*". Many syntactic characteristics are shared by all three forms: they are all typically transitive verbs governing a direct object, the direct object typically belongs to the semantic class "food", etc. Considering both occurrences as instances of the lemma *manger* thus allows the system to consider them as an equivalence class, making generalisations with more statistical weight than when counting the occurrences separately, as long as the individual occurrences really display similar behaviour. Moreover, these generalisations can now be applied to the token "*mangeons*" as well, although it is unknown in the training corpus, simply because it shares the same lemma.

Of course, the system would only know that "*mangez*", "*mangiez*" and "*mangeons*" share the lemma *manger* if it has a lexicon available giving lemmas for all inflected forms (or else some sort of reliable heuristic, such as stemming, for deducing the lemma forms). Thus, generalisations are often incorporated into features via the use of external resources such as lexicons or ontologies. A semantic ontology could, for example, allow the system to group "*pomme*", "*pain*" and "*salade*" into a single a semantic class, *food*. With such a resource, we are now able to recognise the fact that the direct object of a word whose lemma is *manger* is most often a word whose semantic class is *food*, again generalising to any token found in our ontology but absent from our training corpus.

As was discussed in chapter 2, the feature vector is the sole representation of a linguistic context available to the machine learning system. The expert designing the system (in our case a linguist) selects those features which he feels might be useful to selecting the correct label, and, critically, which he feels will generalise well to unknown contexts.

As will be seen in the list of baseline features below, many of our baseline features group together various bits of information, such as the word form of the token on top of the stack concatenated to the word form of the token on top of the buffer. In order to incorporate generalisations from external resources such as those described above, the linguist can construct

[5] `http://www.revues.org/`

[6] `http://redac.univ-tlse2.fr/corpus/wikipedia.html`

features which, instead of using the word form, use the lemma or semantic class. Whether these features should replace or simply complement the word-form features is a matter of trial and error. However, given the fact that the training corpus necessarily contains only a very small sample of possible linguistic constructs, the use of more comprehensive external resources is critical to attaining better accuracy on evaluation corpora.

4.3.2 Talismane's definition of a lexicon

The relatively small size of the FTB (when compared to the Penn Treebank for example) implies a heavier reliance on lexical resources. Talismane defines a default interface for lexicons, around which a wide variety of built-in features are based. This interface defines various characteristics for each inflected form, as follows:

- word form: the actual word form encountered in corpora for words corrsponding to this entry, e.g. *volait*
- lemma: the "dictionary entry header" corresponding to this entry, being the infinitive for verbs, the singular for nouns, and the masculine singular for adjectives, e.g. *voler*.
- lemma complement: free text to distinguish homographic lemmas (e.g. *voler* "to fly" and *voler* "to steal")
- category: the main part-of-speech in the lexicon, used to map this entry to a pos-tag from the tagset
- subcategory: finer part-of-speech in the lexicon, sometimes used to map this entry to a pos-tag from the tagset
- gender: grammatical gender (e.g. feminine/masculine), often needed to identify agreement between nouns and adjectives or past participles
- number: grammatical number (e.g. singular/plural), often needed to identify agreement between nouns and verbs or adjectives
- tense: verb tense (often including mood)
- person: grammatical person (e.g. 1st, 2nd, 3rd), useful for identifying the precise relationship between clitics and verbs
- possessor number: the grammatical number of the possessor (e.g. singular for French *mes*, plural for *notre*)
- morphology: a combined representation of gender, number, tense, person and possessor number, sometimes used to map this entry to a pos-tag from the tagset
- status: allows us to distinguish various levels of frequency or acceptance, enabling Talismane to automatically select the most likely lemma in ambiguous cases (e.g. the status of *suis* can be used to indicate that it is more likely to belong to the lemma *être* than *suivre*)

This information can be incorporated into features for any pos-tagged token, by mapping the token's word form and pos-tag to a set of lexicon entries with a corresponding category, sub-category and morphology, and, if more than one entry is returned, using the entry's status to select the most likely entry. It can thus be used by the pos-tagger for any token already analysed, and by the parser for any token in the parse configuration's stack or buffer.

Furthermore, we can use this information in cases where the system has not yet guessed the pos-tag, e.g. in order to return a list of possible pos-tags or lemmas for a given word form. Such information can be incorporated into features for the tokeniser, and also for any token that has not yet been analysed by the pos-tagger.

4.3.3 LeFFF

The main glossary used by Talismane for its default French implementation is LeFFF [Sagot et al., 2006, Sagot, 2010]. This lexicon contains 404,483 distinct inflected forms, each of which can correspond to several entries, one per associated lemma, for a total of 625,720 entries. The lexicon was constructed semi-automatically, by scanning large corpora for possible additions. It includes all of the information required by the Talismane lexicon interface except for status, as well as subcategorisation frames, which we currently do not use. We added a manual status to the small subset of LeFFF entries that include ambiguous lemmas for the same inflected form and main grammatical category, based on our subjective judgement of the more probable lemma.

We present experiments limiting Talismane's output based on closed class word forms found in the lexicon in section 6.5.1. We present another experiment where we attempt to replace or augment LeFFF with a crowd-sourced glossary in section 7.2.

4.4 Baseline features

We now turn to a description of the baseline features used by Talismane in its four modules. By "baseline", we mean the default core features giving us an acceptable accuracy score, that we will later attempt to improve through various mechanisms.

Overall, our baseline features describe the linguistic context generically, leaving out features that gather additional information when certain specific conditions are met. They were constructed through linguistic intuition and a review of features used by other studies [Candito et al., 2010b, Zhang and Nivre, 2011], and were tested and tuned on the FTB/FTBDep development corpus only. The baseline features include both "atomic" features, providing a single piece of information about the linguistic context, and combinatory features, which combine two or more atomic features as a simple concatenated string. The need to combine features by hand may seem contradictory to the spirit of robust classifiers, which are supposed to handle a myriad of non-independent features and automatically extract the most pertinent patterns and information. However, empirical tests have shown that scores for models with combined features far outweigh those for models with atomic features only: in the parser, for example, using atomic features only gives a labelled accuracy of 74.15%, whereas with combinations we attain 87.78%. Also, since the number of possible combinations is an astronomical figure—that is, $n!$ for n atomic features, it is up to the linguist to identify and test the most informative combinations. Automatic feature selection is a field of study in and of itself, and out of scope for the present thesis. Nevertheless, we are not aware of any methods for automatically selecting the most informative *combinations*—only of methods for

identifying a small set of features which are most informative out of a much larger set . To our knowledge, none of these methods scale up to considering $n!$ possible feature combinations in reasonable time, with n typically between 10 and 20, and each atomic feature in the set of n typically resulting in thousands of separate feature values, each of which is considered as a distinct entity with distinct information content by the model trainers.

Many studies do not give an exhaustive list of features and feature combinations used, therefore making it difficult to compare results. We therefore list the full feature set for every module: this may seem a needless amount of detail, but it should allow our results to be fully reproducible by other teams if so desired.

In terms of methodology, the baseline features were selected by beginning with a very small set of minimalistic features with obvious informative relevance for the task at hand, such as the bigram and suffix features for pos-tagging. We then added additional features one small subset at a time, where each subset contained a set of closely inter-related features, and tested the accuracy change for each subset. If the accuracy increased the subset was retained, and if it decreased or remained unchanged, the subset was either rejected or broken up into smaller subsets and retested.

4.4.1 Cutoff

The baseline feature sets in the following sections result in a huge amount of very specific information being gathered for each linguistic context. Take for example the parser feature combining the lemmas of the top of stack σ_0 and head of buffer β_0. Every different σ_0 and β_0 pair will result in a different string feature result, converted by our training algorithms into a separate numeric feature with its own weight per category. By Zipf's law, the vast majority of these pairs will appear only once in the training corpus, and are very unlikely to provide useful generalisations.

One very simple method for generalising the statistical model constructed during training is by applying a **cutoff** k [Ratnaparkhi, 1998]: a feature f is only included in training if it appears with a non-zero result in at least k training examples. By only including features which appear at least k times in the training corpus $\mathcal{D}$, we hope to only keep those features which represent linguistic regularity, and leave out those features which are highly specific to the training corpus. This has the added practical advantage of vastly reducing the feature space, resulting in quicker training and smaller models. In terms of the parser, for example, moving from a cutoff of 1 to a cutoff of 2, for the FTBDep training corpus and our set of baseline features, reduces the size of the feature set from 12,753,697 to 3,193,551.

There is, unfortunately, no accepted method for selecting the best value of k other than trial and error with a good evaluation corpus. In our case, the cutoff varies from 1 to 10 depending on the module and the classifier. These cutoff tests are presented in section 5.2, page 132.

4.4.2 Sentence detector baseline features

Recall from section 3.3.1, page 79, that the sentence detector tests each character representing sentence boundary candidate, to decide whether or not it is an actual boundary. We define the following baseline features for the sentence detector:

- x (string): the character being tested.

- x? (boolean): is the candidate strong punctuation (“.”, “?” or “!”)?
- xA (boolean): is the next letter a capital letter? Capital letters typically indicate a new sentence.
- (x) (boolean): is the candidate inside parentheses? It is unlikely for a sentence to end inside parentheses.
- $A.$ (boolean): is the candidate a period immediately following a capital letter? This could indicate initials in a person’s name, rather than a sentence boundary.
- nx (string): what are the n characters preceding the candidate (where n goes from 1 to 4)
- xn (string): what are the n characters following the candidate (where n goes from 1 to 4)
- Tx (string): what are the n atomic tokens preceding the candidate (where n goes from 1 to 3)
- xT (string): what are the n atomic tokens following the candidate (where n goes from 1 to 3)
- *surroundings* (string): a string constructed of the n atomic tokens to the right and left of the candidate, replacing alphabetic tokens by “word”, alphabetic tokens starting with a capital letter by “Word”, capitalised words with a length of 1 by “W”, capitalised words with a length of 2 by “Wo”, numbers by “1”, and retaining white space and punctuation (where n goes from 1 to 3). This is an attempt at capturing various patterns indicating that a sentence either should or shouldn’t end—for example, the specific punctuation sequence when quotes and strong punctuation are combined, the use of numbers to indicate lists within a sentence, or the use of initials.

Because of the relatively small number of sentence boundary candidates in the FTB training corpus, 10-fold cross validation was used to tune the baseline feature set.

4.4.3 Tokeniser baseline features

Recall from section 3.3.2, page 80, that the tokeniser is based on a set of patterns corresponding to possible compound words. For each sequence of atomic tokens matched by a pattern, it tests the interval between the first two atomic tokens to decide if it is joined or separate, and applies the result to all other intervals in the sequence. It has access to the tokens inside the matched sequence and surrounding it, as well as to the pattern that was matched.

We define the following “intrinsic” atomic features around the pattern itself:

- π_w: the full word form matched by the pattern. In other words, regardless of context, is this word form likely to be joined or separate?
- π_p: the pattern name. Again, regardless of context, is this pattern typically joined or separate?
- π_g: the pattern’s group name. Regardless of context, is this pattern group typically joined or separate?

We define the following "contextual" atomic one-token features, directly readable from the sentence itself, which are each concatenated with π_p above:

- W_{-1}: the word form of the previous token.
- W_1: the word form of the next token.
- $1st$: is the pattern the first word in the sentence?
- $1stOrPunct$ is the pattern either the first word, or does it immediately follow punctuation?
- $1st|\pi_w$: $1st$ concatenated to π_w.

Furthermore, we define the following atomic "lexical" features, relying on an external lexical resource, and concerning the matched sequence's immediate context:

- P_{-1}: each pos-tag associated in the lexicon with the previous token.
- P_1: each pos-tag associated in the lexicon with the next token.
- L_{-1}: the lemma corresponding to each pos-tag associated in the lexicon with the previous token.
- L_1: the lemma corresponding to each pos-tag associated in the lexicon with the next token.
- U_{-1}: is the previous token unknown in the lexicon.
- U_1: is the next token unknown in the lexicon.
- A_{-1}: a concatenation of all of the pos-tags associated with the previous token in the lexicon.
- A_1: a concatenation of all of the pos-tags associated with the next token in the lexicon.

In terms of combinations, we then generate all possible two-token features based on the five basic feature types above, W, P, L, U and A, for the token position combinations (-2,-1) and (1,2). Each of these two-token features is concatenated with π_p above. For example:

- $W_{-2}W_{-1}$: the word form of the token at a distance of 2 to the left of the current pattern match, combined with the word form of the previous token.
- $W_{-2}P_{-1}$: the word form of the token at a distance of 2 to the left of the current pattern match, combined with each pos-tag associated in the lexicon with the previous token.
- ...
- W_1W_2: W_1 combined with the word form of the token 2 to the right of the current pattern match.
- W_1P_2: W_1 combined with each pos-tag associated in the lexicon with the token 2 to the right of the current pattern match.

- ...

Note that we initially planned to concatenate all features with the pattern group π_g as well, in order to be able to generalise to patterns with no or insufficient attestations in the training corpus, thus enabling the classifier to find useful statistical generalisations at different levels of granularity. However, this systematically lowered accuracy. This is unfortunate, since it means that missing patterns have no contextual features to help decide if they are joined or separate. The cause may be that our current pattern groups are too coarse (grouping together all patterns of a given part-of-speech), and that finer groups are required. Further tests are necessary to see if finer grouping can be made to yield better results.

As can be seen, the features make heavy use of the lexicon, since relying on the training corpus alone would limit us to using the word forms only, leading to considerable data dispersion and unlikely to generalise well. We do not have any features looking for compound forms in lexicons: we concentrate on lexical information for single-word forms only, looking at the words to the right and left of the pattern match. Information about the words at a distance of 2 from the current pattern match is only used when concatenated to the word at a distance of 1: it is assumed that, for example, using the potential pos-tags of the word at distance 2 without knowing anything about the word at distance 1 is likely to create misleading generalisations.

Because of the sparsity of tokeniser pattern matches in the training corpus, 10-fold cross validation was used to tune the baseline feature set. When using a MaxEnt classifier and a cutoff of 3, this resulted in a mean accuracy of 92.06% for pattern matches (standard deviation 0.67%), and 99.75% for all other tokens (standard deviation 0.02%).

4.4.4 Pos-tagger baseline features

Recall from section 3.3.3, page 85, that the pos-tagger needs to assign a pos-tag to a single token at a time, with access to the token itself (and all other tokens in the sentence), and the pos-tags assigned to all previous tokens.

The pos-tagger baseline feature set contains the following "intrinsic" atomic one-token features, directly readable from the token's word form or its position in the sentence:

- W_{-1}, W_0, W_1: the word form of the previous, current, and next token.
- Sfx_{n}: the n-letter suffix of the current token, where n goes from 2 to 5.
- $Pref_{\mathrm{n}}$: the n-letter prefix of the current token, where n goes from 2 to 5. Although it may seem strange from a morphological perspective that prefixes in French give a reliable indication of the correct pos-tag, since the suffix is clearly far more informative for pos-tag choice than the prefix, empirical test results show that this improves accuracy.
- *First*: whether the current token is the first in the sentence.
- *Last*: whether the current token is the last in the sentence.
- various regex features: whether the current token contains a space, a period, a hyphen, whether it begins with a capital letter (followed by a lowercase letter), whether it is all in capital letters, whether it is a number, and whether it ends with a period.
- two compound word features: the first word in a compound, the last word in a compound

- *First*$|W_0$: the current word form combined with the fact that it is the first token in the sentence.
- *First*$|P_0$: the current token's possible pos-tags, combined with the fact that it is the first token in the sentence.

We also make use of the following atomic "lexical" features, relying on an external lexical resource. Note a critical difference between backward-looking lexical features (where index < 0), which can base the lexical information on the single pos-tag already assigned, and forward-looking lexical features (where index ≥ 0), which rely on the lexicon alone to provide information for all possible pos-tags associated with this word form.

- P_{-1}, P_0, P_1: the pos-tag assigned to the previous token, and the pos-tags associated in the lexicon with the current and next token. Note that P_{-1} translates the standard bigram feature (predicting the current pos-tag based on the previous pos-tag). P_0 is a feature which attempts to capture the extent to which the lexicon is an accurate reflection of the training corpus: how likely it is that a token has a given pos-tag, if it is listed in the lexicon as having this pos-tag.
- L_{-1}, L_0, L_1: the previous token's lemma, and the lemmas corresponding to each pos-tag associated in the lexicon with the current and next token.
- A_0, A_1: all of the pos-tags associated in the lexicon with the current (or next) token, concatenated together.
- U_0, U_1: whether the current (or next) token is unknown in the lexicon.

We now combine the atomic features above into the following two-token features:

- two-token features with (-2,-1):
  - $P_{-2}P_{-1}$: the previous two tokens' pos-tags. This translates the standard trigram feature: predicting the current pos-tag based on the previous two pos-tags.
  - $P_{-2}L_{-1}$, $L_{-2}L_{-1}$, $L_{-2}P_{-1}$: variants on the previous feature replacing one or both of the pos-tags with the lemma.
- two-token features with (-1,0): $P_{-1}W_0$, $L_{-1}W_0$, $P_{-1}U_0$ and $L_{-1}U_0$.
- two-token features with (0,1): W_0/U_0 with $W_1/P_1/L_1/A_1/U_1$.

Finally, we make use of the following three-token features:

- three-token features with (-2,-1,0): $P_{-2}P_{-1}W_0$, $P_{-2}L_{-1}W_0$, $L_{-2}L_{-1}W_0$ and $L_{-2}P_{-1}W_0$.
- three-token features with (-1,0,1): P_{-1}/L_{-1} with W_0 and $W_1/P_1/L_1$.
- three-token features with (0,1,2): $W_0W_1W_2$, $W_0A_1A_2$, $W_0A_1W_2$, and $W_0W_1A_2$.

This feature set generates of course a huge number of hapax features, with very little likelihood of being generalisable. This data dispersion is dealt with by applying a cutoff (see section 4.4.1). When using a MaxEnt classifier with 100 iterations and a cutoff of 7, these features give an overall accuracy of 97.59% on the FTB development corpus (10% of the FTB corpus). If we use atomic features only, we have an overall accuracy of 97.03% for the same configuration.

4.4.5 Parser baseline features

Recall from section 3.3.4, page 86, that at each step of parsing, the parser is presented with a parse configuration containing a stack of partially processed tokens and a buffer of unprocessed tokens, and needs to decide whether a dependency exists between the top of stack σ_0 and the head of buffer β_0.

We begin with a set of basic features describing σ_0 and β_0 themselves from various angles, in the hope that there are some useful statistical generalisations to be made based on this isolated information:

- POS(σ_0): σ_0's pos-tag all on its own (on the assumption that certain pos-tags are more likely to generate certain dependencies, regardless of the context).
- LEM(σ_0): σ_0's lemma all on its own. In this feature and all other features referring to lemmas, we use the lemma if it was found in the lexicon, otherwise we use the word form.
- POS|LEM(σ_0): POS(σ_0) and LEM(σ_0) combined.
- POS(β_0), LEM(β_0), POS|LEM(β_0): as above but for β_0.
- POS($\sigma_0|\beta_0$): the postags of σ_0 and β_0 combined. In other words, what dependencies (if any) are most likely for this pos-tag pair.
- LEM($\sigma_0|\beta_0$): the lemmas of σ_0 and β_0 combined.
- POS|LEM(σ_0)POS|LEM(β_0): the pos-tags and lemmas of σ_0 and β_0 combined.
- POS|LEM(σ_0)POS(β_0): the pos-tag and lemma of σ_0 with the pos-tag of β_0.
- POS(σ_0)POS|LEM(β_0): the pos-tag of σ_0 with the pos-tag and lemma β_0.
- LEX($\sigma_0|\beta_0$): if either σ_0 or β_0 has a closed class pos-tag, then use the lemma, otherwise use the pos-tag. In other words, lexicalise any closed classes. Note that this pair is only used if one of the two has a closed class pos-tag, since otherwise it would duplicate information in POS($\sigma_0|\beta_0$).
- LEXVERB($\sigma_0|\beta_0$): if either σ_0 or β_0 has a closed class pos-tag or is a verb, then use the lemma, otherwise use the pos-tag. In other words, lexicalise closed classes and verbs. Note that this pair is only used if one of the two has a closed class pos-tag or is a verb.

To provide useful information about the potential dependency between σ_0 and β_0, other baseline parser features described in this section are used both on their own, and in combination with the three σ_0/β_0 pair features POS($\sigma_0|\beta_0$), LEX($\sigma_0|\beta_0$) and LEXVERB($\sigma_0|\beta_0$). The intuition behind this can be seen by taking a feature such as LEM(β_1), the lemma of the token on the buffer just after the head of buffer. Take the following example:

Example 4.1 *Il veut protéger l'industrie de l'automobile de la libre concurrence.*
(He wants to protect the automobile industry from free competition.)

Now, assume we reach a configuration where σ_0=*protéger*, β_0=*de*$_2$ (the 2nd occurrence of *de*) and β_1=*concurrence*. The feature LEM(β_1) now returns "concurrence". On its own, this information tells us next to nothing about the potential dependency between σ_0 and β_0. If we combine it with POS($\sigma_0|\beta_0$), we now have the feature result (V, P, concurrence), which might be of help. Combining it with LEX($\sigma_0|\beta_0$) gives us (V, de, concurrence). Finally, combining it with LEXVERB($\sigma_0|\beta_0$) gives us (protéger, de, concurrence), which definitely seems useful in the present case for helping us to predict the dependency *de_obj(protéger, de)*, but only if it is attested a sufficient number of times in the training corpus.

We now turn to features gathering some additional information about σ_0 and β_0 themselves, used on their own and in combination with the three σ_0/β_0 pairs described above:

- GEN($\sigma_0|\beta_0$): grammatical gender for both σ_0 and β_0 (assuming this information is only useful together, when it indicates agreement or lack thereof between nouns and adjectives/past participles).

- NUM($\sigma_0|\beta_0$): grammatical number (e.g. singular/plural) for both σ_0 and β_0 (assuming this information is only useful together, when it indicates agreement or lack thereof between nouns and verbs/adjectives).

- TENSE($\sigma_0|\beta_0$): the verb tense of σ_0 and β_0 (e.g. for recognising auxiliary verbs)

- PER($\sigma_0|\beta_0$): the grammatical person (1st, 2nd, 3rd) of σ_0 and β_0 (e.g. to differentiate reflexive clitics from direct object clitics).

- MORPH($\sigma_0|\beta_0$): the full morphological details string of σ_0 and β_0, including gender, number, tense, person, and possessor number.

- DISTANCE(σ_0, β_0): the distance between σ_0 and β_0, treated as a string rather than a number, with any distance > 6 marked as "long". This gives us a total of 7 nominal classes. Only used in combination with the three σ_0/β_0 pair features (not on its own). The intuition here is that certain dependencies (e.g. object of a preposition) will always be short-range, whereas others (e.g. coordination between verbs) can be long-range.

We now turn to some contextual information from the partial dependency tree already constructed, relating to the governor (or head) of σ_0 (HEAD), σ_0 and β_0's left-most dependents (LDEP), and σ_0's right-most dependent (RDEP). The intuition here is that adding another dependency is often determined by the current outermost dependency. For example, when trying to determine if a noun is the antecedent of a relative phrase, it is useful to know that the current leftmost dependent of a verb is a relative pronoun. As presented in chapter 1, the richness of transition-based parsers is based on such features, which are only available in systems which tackle parsing sequentially, and therefore cannot be used in graph-based parsers. Again, these features are used on their own and in combination with the three σ_0/β_0 pairs.

- DEP(σ_0): the dependency label governing σ_0.

- POS(HEAD(σ_0)), LEM(HEAD(σ_0)), POS|LEM(HEAD(σ_0)), TENSE(HEAD(σ_0)), DEP(HEAD(σ_0)): the pos-tag, lemma, pos-tag and lemma combined, tense, and governing dependency label for the governor of σ_0.

- the same five features for LDEP(σ_0).
- the same five features for RDEP(σ_0).
- the same five features for LDEP(β_0). Note that in the arc-eager transition system, RDEP(β_0) can never exist.
- VALENCY(σ_0), VALENCY(β_0): the current number of dependents for σ_0 (or β_0), treated as a string rather than a number. Only used in combination with the three σ_0/β_0 pair features (not on its own).

We now look a bit further down in the buffer. The intuition here is that items to the right of the current head of buffer are often critical to determining the head of buffer's governor. This is the case, for example, when trying to determine the first conjunct of a coordinating conjunction (see section 1.1.3.2). It is also the case in prepositional phrase attachment, where knowing the object of a preposition helps us determine whether it is governed by the previous noun or verb.

- POS(β_1), LEM(β_1), POS|LEM(β_1): the pos-tag, lemma and both combined for the 1st token after the head of buffer.
- POS(β_1)POS(β_2): the pos-tag of the 1st and 2nd tokens after the head of buffer
- POS|LEM(β_1)POS(β_2), POS|LEM(β_1)POS|LEM(β_2): other combinations with the first two tokens after the head of buffer
- POS(β_1)POS(β_2)POS(β_3): the pos-tags of the first three tokens after the head of buffer
- POS|LEM(β_1)POS(β_2)POS(β_3), POS|LEM(β_1)POS|LEM(β_2)POS(β_3): other combinations with the first three tokens after the head of buffer

Similarly, we look a bit deeper into the stack. Our intuition is that we sometimes need to confront the current top of stack with the items higher up in the stack before deciding on the most likely governor/dependent for the head of buffer. In example 4.1, assume we reach a configuration where σ_1=*protéger*, σ_0=*industrie*, β_0=*de*$_2$ and β_1=*concurrence*. We have to decide whether to construct the phrase *protéger de concurrence* ("protect from competition") or *industrie de concurrence* ("the competition industry"). Knowing that σ_1 is a verb which tends to govern an object with *de* can help us favor the first option, thus selecting the `reduce` transition, since *industrie* has no more dependents. Similarly, in the configuration where σ_2=*protéger*, σ_1=*industrie*, σ_0=*automobile*, β_0=*de*$_2$ and β_1=*concurrence*, we may need to look as far as σ_2 to select the correct `reduce` transition. The following features are only added in combination with the three σ_0/β_0 pairs (not on their own):

- POS(σ_1), LEM(σ_1), POS|LEM(σ_1): the pos-tag, lemma and both combined for token just beneath the current top of stack.
- POS(σ_2)POS(σ_1): the pos-tag of the 1st and 2nd tokens beneath the top of stack.
- POS|LEM(σ_2)POS(σ_1), POS|LEM(σ_2)POS|LEM(σ_1), POS(σ_2)POS|LEM(σ_1): other combinations with the first two tokens beneath the top of stack.

We also include features that look further down the stack and buffer together. In example 4.1, assume we reach two configurations, both with σ_1=*protéger*, σ_0=*industrie*. In the first configuration, β_0=*de*$_1$ and β_1=*automobile*, and in the second configuration β_0=*de*$_2$ and β_1=*concurrence*. A feature combining σ_1 and β_1 allows us to look at the verb *protéger* and the prepositional object together, when deciding whether or not *industrie* governs *de*. With sufficient semantic knowledge (or sufficient occurrences in the corpus), such a feature would allow us to decide that in the first case *industrie* governs *de*, but in the second case it doesn't. The following features are only added in combination with the three σ_0/β_0 pairs (not on their own):

- POS(σ_1)POS(β_1): the pos-tag of the 1st token beneath the top of stack and the 1st token after the head of buffer.
- POS|LEM(σ_1)POS(β_1), POS|LEM(σ_1)POS|LEM(β_1), POS(σ_1)POS|LEM(β_1): other combinations with the same two tokens.

In some cases, the tokens immediately to the left or right of a token in the sentence can help decide whether it is dependent on a token farther away. This is particularly true of punctuation—a comma, for example, can signal a break in syntactic continuity. We thus look at the tokens immediately to the left of σ_0 and β_0 in the sentence, noted LSEQ$_{\mathrm{n}}$ where n is the distance from the σ_0, the tokens immediately to the right of σ_0, noted RSEQ$_{\mathrm{n}}$.

- POS(LSEQ$_1$(σ_0)), LEM(LSEQ$_1$(σ_0)): the pos-tag and the lemma of the token immediately to the left of σ_0.
- POS(RSEQ$_1$(σ_0)), LEM(RSEQ$_1$(σ_0)): the pos-tag and the lemma of the token immediately to the left of σ_0, except when this token is β_0.
- POS(LSEQ$_1$(β_0)), LEM(LSEQ$_1$(β_0)): the pos-tag and the lemma of the token immediately to the left of β_0, except when this token is σ_0.
- POS(RSEQ$_1$(β_0)), LEM(RSEQ$_1$(β_0)): the pos-tag and the lemma of the token immediately to the right of β_0.
- POS(LSEQ$_1$(σ_0))POS(RSEQ$_1$(σ_0)): the pos-tags of the tokens immediately to the left and right of σ_0.
- POS(LSEQ$_2$(σ_0))POS(LSEQ$_1$(σ_0)): the pos-tags of the two tokens immediately to the left of σ_0.
- POS(RSEQ$_1$(σ_0))POS(RSEQ$_2$(σ_0)): the pos-tags of the two tokens immediately to the right of σ_0.
- POS(LSEQ$_2$(β_0))POS(LSEQ$_1$(β_0)): the pos-tags of the two tokens immediately to the left of β_0.

In a system using a MaxEnt classifier with 100 training iterations and a cutoff of 7, and using the gold standard pos-tags as input to the parser, these baseline features give an overall labelled accuracy of 87.78% on the FTBDep development corpus, and an unlabelled accuracy of 90.04%. Using atomic features only, we have a much lower labelled accuracy of 74.15% and unlabelled accuracy of 77.85%. We compare the results attained by various configuration options in the next chapter.

4.5 Baseline rules

In section 3.5, page 95, we presented the concept behind rules, which override the statistical model's decisions, either imposing or prohibiting the choice of a certain category. There are very few baseline rules used by Talismane, as rules are typically constructed after error analysis to cover cases which are under-represented in the training corpus, as presented in section 6.5, page 184.

4.5.1 Tokeniser

Tokeniser rules are used to force Talismane to group a certain sequence of atomic tokens together as a single compound word. The tokeniser contains various rules for handling numbers, recognising sequences of numbers (written in either the decimal system or in full letters). Thus, both *"74 543,67"* and *"trente-deux mille cinq cent quatre-vingt treize"* are tokenised as a single token.

There is also a special rule for recognising e-mail and web-site addresses.

In addition to marking compound words, the tokeniser rules are used to replace the original word forms by equivalence classes likely to follow the same usage patterns, and the features will only see the equivalence class label, rather than the original word form.

For example, in the case of numbers, we use the following labels:

- "deux": any number written in full letters.
- "9.99": any decimal number or number in scientific notation.
- "31": any whole number from 1 to 31 (to capture dates in a single class).
- "1999": any whole number from 1000 to 2999 (to capture years in a single class).
- "1999-2000": any pair of numbers from 1000 to 2999 with a dash between them (to capture year ranges in a single class).
- "999": any other whole number.

Similarly, e-mail and web-site addresses are replaced by a generic placeholder, all "pretty" quotation marks (e.g. guillemets) are replaced with standard quotation marks (for consistency with the FTB), and all bullets or dashes at sentence start are skipped.

4.5.2 Pos-tagger

The pos-tagger comes with three types of baseline rules:

- Closed classes: for each closed class pos-tag (e.g. prepositions, conjunctions, pronouns, etc.), only allow the pos-tagger to assign this pos-tag if it exists in the lexicon. This prevents us, for example, from inventing new prepositions.
- Open classes: do not assign an open class pos-tag (e.g. common noun, adjective, etc.) to a token if it is only listed with closed classes in the lexicon. This prevents us, for example, from assigning a tag such as "common noun" to the token "*le*".

- Ad-hoc: rules to assign the proper noun pos-tag (`NPP`) to e-mail and web-site addresses already recognised by the tokeniser rules.

Regarding the used of closed-class rules, we present an experiment testing their usefulness in section 6.5.1, page 185.

4.6 Discussion

In this chapter we covered the methods via which linguistic knowledge is incorporated into statistical syntax analysis, and more specifically into Talismane for the the analysis of French. We presented available training and evaluation corpora, as well as various available resources, before turning to a list of baseline features and rules used by the four Talismane modules. This concludes the four contextual chapters for this thesis, in which the general framework for robust statistical French syntax analysis was presented. We are now ready to turn to a series of experiments, starting with a general evaluation of Talismane using different configuration parameters for the same set of features, and then moving on to specific experiments attempting to improve on this baseline for various specific phenomena.

Part III

Experiments

Chapter 5

Evaluating Talismane

In the first two parts of this thesis, we presented the framework within which our experiments in robust statistical syntax analysis are performed. We now turn to the experiments themselves. Our first concern in section 5.1 is to define our general evaluation methodology within this thesis. We then turn to a series of experiments whose purpose is to vary the system's parameters in order to find the best baseline configuration from a purely algorithmic perspective, using the set of baseline features defined in chapter 4:

- In section 5.2 we run a comparison of the various classifiers presented in section 2.8, page 59, after selecting the best parameters for each classifier type.
- In section 5.3, we attempt to prove the usefulness of classifier confidence: is it correlated to guessing correctly?
- In section 5.4 we look at the improvements provided by a beam search with various beam widths, for the different Talismane modules.
- In section 5.5, we try to ascertain whether modules at a higher level of abstraction can correct the errors of the previous modules, if the full beam is propagated from one level to the next.

The main purpose of the chapter is to fix a baseline syntax analysis configuration, on which further experiments will attempt to improve, as well as to gain certain insights into what changes occur with different configurations, and how different corpora react to these changes.

5.1 Evaluation methodology

It has become common practice in NLP to present evaluation in terms of overall accuracy before and after applying a certain modification. While this statistic is useful for comparison purposes, it hides a vast quantity of interesting details. Not all applications are interested in correctly annotating the same linguistic phenomena. For example, some, such as terminology extraction, may only be interested in short-range phenomena, while others, such as question answering, may need the information provided by long-range dependencies.

Therefore, depending on the experiment, we may look into some additional statistics. Sometimes this will be cumulative accuracy by attachment distance, where we view the relative decrease in accuracy as we take longer distance dependencies into account. In other cases, we may concentrate on specific phenomena only: a certain word, or a certain dependency label, in which case precision, recall and f-score become meaningful (see section 2.7, page 58 for details on these metrics). In highly specific cases, the overall accuracy may not tell us very much. Instead, we state equivalent results in terms of the proportional reduction in the remaining error count for a specific type of relation or a specific word.

Furthermore, for almost all experiments, we compare results for the FTBDep dev and test corpora with results for other evaluation corpora, in order to ascertain to what extent the method is generalisable.

Another point of interest is comparing the degree of agreement between two methods. Two methods may both have an overall accuracy of 87%, but can differ in up to 26% of their guesses. If methods tend to guess wrong at different places, we will call these methods "orthogonal": there is an opportunity of somehow combining their analyses in order to attain a better score. Measuring agreement between two methods can be done using the same kappa statistic presented in section 4.1.1, page 102 on inter-annotator agreement: instead of comparing the annotations by two annotators, we compare the annotations by two methods. A high kappa indicates the methods are parallel in their informative content: a low kappa indicates they are orthogonal. However, we more often simply look at the number of new corrections as opposed to the number of new errors—which are at the basis of the statistical significance measures presented in section 5.1.2.

Many experiments isolate a single module (e.g. the parser), feeding it as input the so-called "gold standard" annotations from the evaluation corpus, instead of guessing these annotations using the previous module. In the case of the parser, for example, we judge its performance on the assumption that it receives as input the exact set of pos-tags annotated in the evaluation corpus. A more realistic test combines several modules in a chain, guessing the pos-tags and passing the guessed pos-tags to the parser for evaluation. We attempt this in several experiments, but not systematically, since it complicates the evaluation procedure.

Now, general statistics such as accuracy, based on the assumption the annotated corpora represent a perfect "gold standard", have to be treated with caution. The corpora's internal consistency, as reflected by the available inter-annotator agreement statistics (Cohen's kappa for classification tasks or average f-score for identification tasks), is rarely above 90%. This leads us to consider more "qualitative" evaluation methods, reviewing and judging specific differences in annotation between the baseline (control) method and the experimental method, or often, when a reference "gold" annotation exists, as a three-way comparison between the reference annotation, the baseline method and the experimental method. To this end, we examine the first n differences meeting certain criteria in either annotated or unannotated corpora. This is particularly useful when we aim at improving annotation for a rare phenomenon, with a small number of attestations in annotated corpora. For example, if we add a feature to help a pos-tagger distinguish the word *que* as a negative adverb or subordinating conjunction, we can apply Talismane to an unannotated corpus with and without the feature, and view the first 100 differences, in order to see if the feature was useful from a more qualitative perspective, and, in cases where it was not useful, to try to ascertain the reasons.

Still, even this qualitative evaluation makes the assumption that syntax analysis is an end in and of itself, which is rarely the case: in most cases, it is simply a link in a chain for some specific application. Measuring improvement for specific applications is, however, outside the

scope of this thesis, and is left as a future perspective.

In terms of technology, all tests in this thesis were run on an Intel Xeon E3-1245 V2 machine, with a 3.4GHz clock speed, 4 cores, 8 threads, and 8 Mb cache, running the Ubuntu 12.04.2 LTS 64-bit operating system.

Throughout this chapter, we use the FTBDep training, dev and test corpora for all modules, so as to allow for consistent results when testing such aspects as beam propagation. This means results are necessarily lower for pos-tagging than those presented in the previous chapter, given smaller size of the training corpus.

5.1.1 Parse evaluation metrics

Dependency parses are typically evaluated using two overall metrics: the labeled attachment score (LAS) is the accuracy of all arcs having the correct governor and the correct label. The unlabeled attachment score (UAS) is the accuracy of all arcs having the correct governor, regardless of the label. The syntax annotation guides provide no guidelines for annotating punctuation, except in the case of coordinating commas. This results in selecting fairly arbitrary governors for punctuation, rendering its evaluation meaningless. In all of our experiments, we thus consider the LAS and UAS excluding punctuation.

5.1.2 Statistical significance

Although results may seem to improve over the baseline with a given method, we would like to ensure that the improvement is due to the method itself, rather than random chance. To this end, statistical significance tests are applied. In these tests, the so-called "null hypothesis" states that the new method had no effect on the results, when compared to the baseline. Typically, the significance tests will result in an abstract value, which can then be interpreted as a probability that the null hypothesis is true, i.e. that any changes could have occurred by random chance alone. This probability is known as the p-value.

	Alternative right label	Alternative wrong label	Baseline totals
Baseline right label	a	b	$a+b$
Baseline wrong label	c	d	$c+d$
Alternative totals	$a+c$	$b+d$	$a+b+c+d=n$

Table 5.1: Baseline method vs. alternative method contingency table

In this thesis, since we have restated all of our tasks as classification problems, we can use McNemar's test [McNemar, 1947] which is applicable to dichotomous data: the dichotomy in our case can be stated as the yes/no question: did method M assign the correct label y to linguisitic context x. Assume we are comparing a certain alternative method to the baseline method. We show the initial contingency table in table 5.1. This table is constructed by

comparing the annotation of each linguistic context by each method to the gold annotation. Under McNemar's test, we approximate the χ^2 statistic as follows:

$$\chi^2 \approx \frac{(b-c)^2}{b+c} \tag{5.1}$$

The intuition behind this measurement is that only new corrections and new errors are useful to comparing the two methods. The unaffected mass of items which are correct or incorrect in both methods have no bearing on the actual improvement provided by the alternative method. By convention, we state that our results are statistically significant if the null hypothesis probability is less than 5%, typically stated as p-value ≤ 0.05, which corresponds to $\chi^2 \geq 3.841$. For small test samples where $b + c < 25$ (e.g. when testing a very specific phenomenon), since McNemar's test no longer accurately approximates a χ^2 distribution, we switch to the binomial test, which gives an exact, rather than an approximate, p-value.

5.2 Evaluating classifiers and classifier parameters

Our first set of tests aims at comparing the various classifiers with different parameter values for parsing and pos-tagging, in order to see to what extent the classifier choice affects our final results. For each module, we first attempt to select the best parameters for each classifier type (perceptron, MaxEnt, linear SVM), and then compare the various classifiers in their best configuration.

5.2.1 Evaluating classifiers for parsing

We first evaluate the various classifiers on the isolated parsing task, using the gold standard pos-tags as input. Each classifier has its own set of parameters, but one parameter is shared for all three classifier types: the cutoff (see section 4.4.1, page 116), determining how many times a feature must appear in the training corpus to be included when training the model. Ignoring for now the accuracy attained by different values of cutoff, changes to this parameter have considerable influence on model size and training time, as reflected by the number of parsing features to be considered for different cutoffs in the FTBDep training corpus, shown below:

Cutoff	Feature count
1	12,753,697
2	3,193,551
3	1,758,253
4	1,223,314
5	939,264
7	644,556
10	442,408
12	367,306

5.2.1.1 Tuning perceptron parameters for parsing

For Perceptron classifiers (section 2.8.2, page 61), we have two main parameters: the number of training iterations i, and the cutoff. An additional parameter, the tolerance τ, allows us

to stop training if the perceptron model stabilizes at a certain accuracy, so as to avoid a bias towards the weights of the final iterations (if they are all identical). In our case, we selected a tolerance of 1e-5, which resulted in all iterations being performed. For information, at a tolerance of 1e-4, the trainer breaks out after approximately 50 iterations, so the results are similar to those for the 50 iteration test with $\tau = 1e - 5$. We measured LAS, UAS, training time and analysis time for iteration values in {50, 75, 100, 150, 200} and cutoff values in {1, 3, 5, 7, 10, 12}, where a cutoff of 1 implies no cutoff (all features retained).

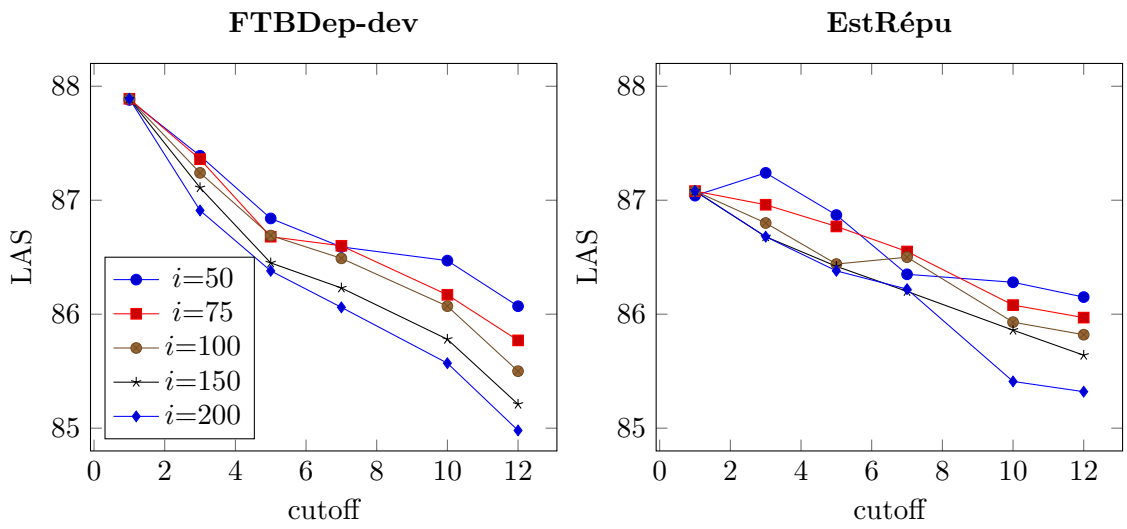


Figure 5.1: Evaluation corpora LAS for a perceptron classifier using different values for iterations i and cutoff

The results for the FTBDep development and EstRépu corpora are shown in fig. 5.1, using the baseline features and the gold standard pos-tags as input. All corpora except for EstRépu follow the same general pattern as FTBDep development. UAS scores follow the LAS curves fairly closely, but are about 2.2% higher. There are two surprising results here: first of all, *cutoff* seems to systematically reduce accuracy scores—whereas for the other classifiers below it will be a critical parameter. The only exception among the evaluation corpora is the EstRépu corpus, with a slight peak at a cutoff of 3. This implies that perceptrons are somehow able to "naturally" avoid over-fitting, selecting each feature to the extent that it is generalisable. Secondly, scores diminish almost systematically as the number of *training iterations* increases. This implies that the perceptron algorithm is able to do most of its work in the early iterations—later iterations give a bias to the final weights in the averaging mechanism, and lower the overall score considerably.

Still, it seems unlikely that this curve would continue ever upwards for fewer and fewer iterations. We therefore ran a second test to observe its behavior in the low iterations and low cutoff zones, with iteration values in {10, 20, 30, 40, 50} and cutoff values in {1, 2, 3, 4, 5}. The results are shown in fig. 5.2 for the same two corpora. Again, curves for all other corpora followed that of the FTBDep development corpus. All corpora except for EstRépu displayed a significant loss ($\approx 0.5\%$) at a cutoff of 2. This confirms an ideal cutoff of 1, and a stabilisation in the accuracy scores at iteration 20. This is very positive from the perspective of training time, since it implies the perceptron model trains very quickly to its maximum

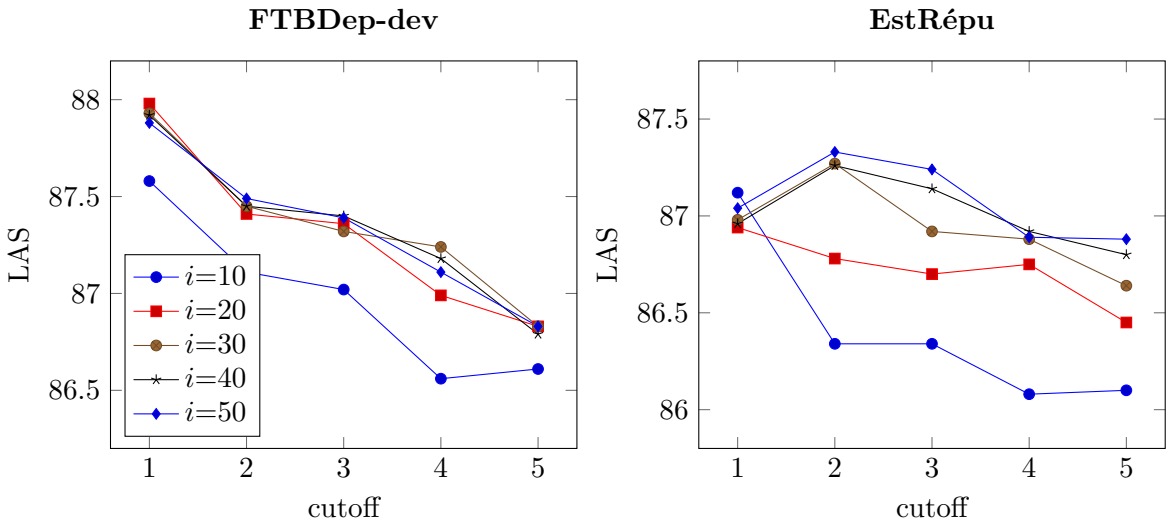


Figure 5.2: Evaluation corpora LAS for a perceptron classifier using lower values for iterations i and cutoff

accuracy model.

5.2.1.2 Tuning MaxEnt parameters for parsing

For MaxEnt classifiers (section 2.8.3, page 64), we have two main parameters: the number of training iterations i, and the cutoff. We measured LAS, UAS, training time and analysis time for iteration values in {50, 75, 100, 150, 200} and cutoff values in {1, 3, 5, 7, 10, 12}, where a cutoff of 1 implies no cutoff (all features retained). The results for the various evaluation corpora are shown in fig. A.1, page 218, using the baseline features and the gold standard pos-tags as input. UAS curves are not shown: they follow the LAS curves fairly closely, but are approximately 2.3% higher.

The usefulness of *cutoff* as a means of avoiding over-fitting is clear: in all cases, a cutoff of 3 is far superior to no cutoff at all (i.e. cutoff=1). This is a conspicuous difference between MaxEnt and perceptrons, where cutoff was not useful. Other than this, a perfect value for cutoff isn't clearly visible. In the journalistic corpora closest to the training corpus, ftbDep-dev and EstRépu, a cutoff of 7 gives the peak LAS. As we move to more distant corpora, in the case of FrWiki (encyclopedic) and Europarl (parliamentary proceedings), we seem to get slightly better results with cutoffs of 10 and 12, especially after more training iterations (150/200). The higher cutoff seems logical, since at a more distant corpus, only the features representing considerable linguistic regularity are likely to be useful. This seems to be confirmed by the EMEA-dev corpus (official medicine evaluation reports), with a clear peak at a cutoff of 10 for a high number of iterations. However, the FrWikiDisc corpus (informal, unedited discussion pages), which we would assume to be quite distant from ftbDep, shows a surprising peak at a lower cutoff of 5, followed by fairly flat statistics at higher cutoffs. One perspective for post-thesis work would be to try to predict the ideal cutoff value based on corpus characteristics (e.g. function word usage, sentence length, etc.).

Regarding the number of *training iterations*, it is clear that more iterations do not lead to over-fitting. The 50 iteration scores are systematically well below the others. The scores from

100 to 200 iterations are generally quite close to each other, tending to vary more for the more distant corpora, especially at higher cutoffs: it would seem that more training iterations are required to attain the best weights when more generic features are used (i.e. higher cutoff). Overall, the best (iterations, cutoff) pairs seem to be (100, 7) for journalistic corpora, and (150, 10) for more distant corpora.

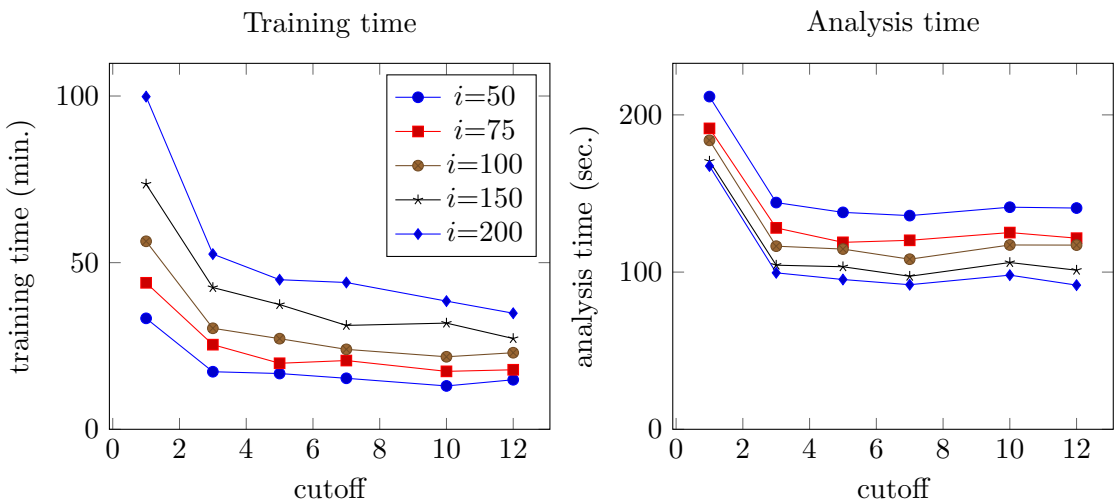


Figure 5.3: Training time and analysis time (FTBDep-dev) for a MaxEnt classifier using different values for iterations i and cutoff

Figure 5.3 shows *model training time* (in minutes) and *analysis time* (in seconds) for FTBDep development corpus, for the same set of parameters. As was seen above, more training iterations rarely lower the score: their main disadvantage is longer training time. This disadvantage affects the team building the model only, as it takes longer to test minor model updates. Surpisingly, more training iterations result in faster analysis—analysis time after 200 iterations is approximately 85% of analysis time after 100 iterations. This is another major advantage for higher iterations, since our definition of a robust syntax analyser included the possibility of analysing large corpora in reasonable time. It is difficult to imagine a logical reason for this unexpected speed-up. In our tests, we will thus use 100 or 150 training iterations, but production models can easily use 200 or more.

5.2.1.3 Tuning linear SVM parameters for parsing

In the case of linear SVMs, we have three main parameters to tune: the soft margin parameter C and the insensitivity zone width parameter ϵ (see section 2.8.4, page 68), and the feature cutoff parameter which applies to all of the classifiers (see section 4.4.1, page 116).

Figure A.2 on page 219 shows the LAS results of a grid search between values of C in the set $\{2^{-3}, 2^{-2}, \ldots, 2^{3}\}$ and values of ϵ in the set $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. The results concern all of our evaluation corpora, with the baseline features, a cutoff of 7 and gold standard pos-tags. Again, the UAS curves are not shown: they follow the LAS curves closely, but are about 2.2% higher. The curves all follow the same general shape, first rising to a peak and then falling. The peaks are almost always obtained with an insensitivity zone $\epsilon = 0.01$ or 0.001. Since the former is much quicker to train, we retain this value for further testing. Note

also that the curves for wider values of ϵ are far flatter than the curves at $\epsilon = 0.01$ or 0.001. Thus, selecting a narrow insensitivity zone requires us to carefully select the correct value of the soft margin parameter C. In our case, a value of $C = 2^{-2}$ gives the maximum LAS in almost all cases, except for the Europarl corpus, where $C = 2^{-1}$ is slightly higher.

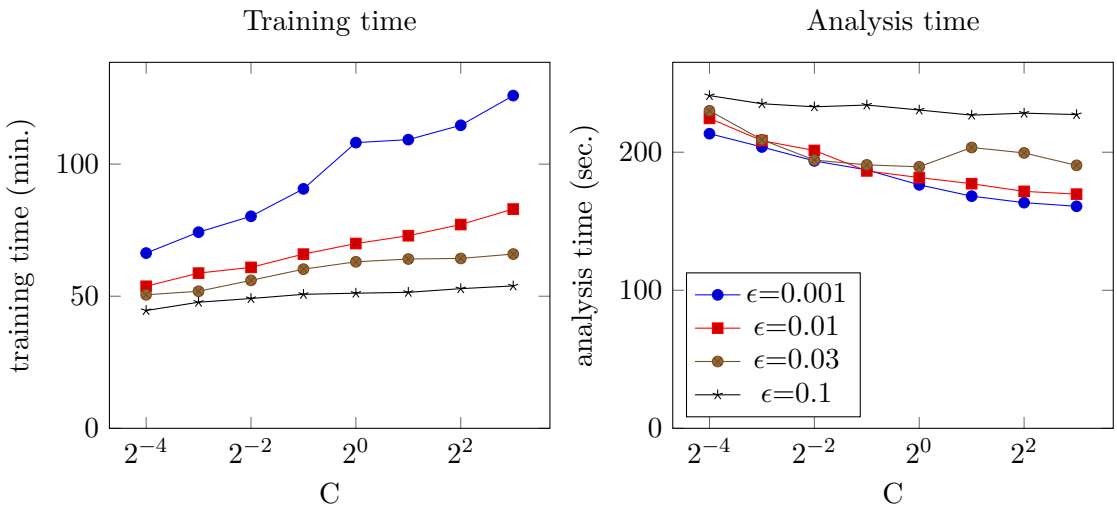


Figure 5.4: Training time and analysis time (FTBDep-dev) for a linear SVM using different values for C and ϵ

Figure 5.4 shows model training time (in minutes) and analysis time (in seconds) for FTBDep development corpus, for the same set of C and ϵ parameters. The training time increases with C and is higher for narrow insensitivity zones (ϵ). Analysis time on the contrary decreases with C and is lower for narrow insensitivity zones, although the difference between $\epsilon = 0.01$ and $\epsilon = 0.001$ is slight. Since there is no significant advantage to using a narrower insensitivity zone, we use $\epsilon = 0.01$ and $C \in \{2^{-3}, 2^{-2}, 2^{-1}\}$ as a basis for tests on cutoff.

The results for cutoff tests are shown in fig. A.3, page 220. Here we tested the FTBDep development corpus, with a fixed value of $\epsilon = 0.01$, $C \in \{2^{-3}, 2^{-2}, 2^{-1}\}$ and cutoffs $\in \{3, 5, 7, 10, 12\}$. The curves are not easily interpretable, since no single cutoff seems to work for all corpora. Surprisingly, unlike MaxEnt, the ideal cutoff value does not seem related to the assumed corpus distance: although there is a definite preference for cutoff 5 in the journalistic EstRépu corpus, the very distant FrWikiDisc and EMEA-dev corpora also benefit from lower cutoffs. A cutoff of 5 with $C = 2^{-2}$ seems to provide fairly good results throughout, and it is the value retained for further comparison.

5.2.1.4 Comparing the best configurations for parsing

We have thus selected four configurations for comparison: the perceptron classifier with 20 iterations and a cutoff of 1 (Perceptron), the MaxEnt classifier with 100 iterations and a cutoff of 7 (MaxEnt1), the MaxEnt classifier with 150 iterations and a cutoff of 10 (MaxEnt2), and the linear SVM classifier with a cutoff of 5, $C = 0.25$ and $\epsilon = 0.01$ (LinearSVM).

Figure 5.5 shows a comparison of LAS for the four configurations retained. Again, UAS follows LAS curve shapes closely, but is approximately 2.2% higher. As can be seen, the linear SVM model generally beats the two MaxEnt models by 1%, and the perceptron model by an

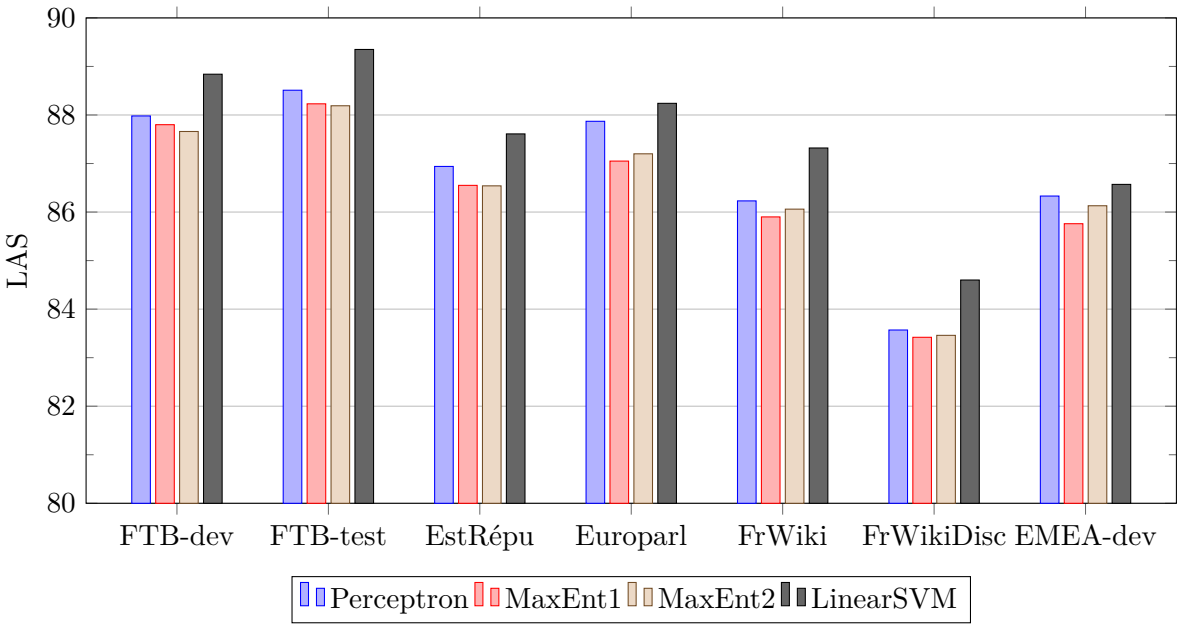


Figure 5.5: Parsing classifier comparison: LAS by corpus

average of 0.7%. Table 5.2 gives more exact comparison statistics. In terms of the basic setup, the main advantage to the MaxEnt models is a fairly short training time, small model size, and quick analysis time for small corpora. The perceptron model has quick training time as well and slightly better accuracy, but is very slow in analysis, and the model size is about 20 times bigger. The linear SVM model is very slow to train (over an hour) and has a very large model size—about 50 times bigger than the MaxEnt model. However, it performs analysis fairly quickly, especially for larger corpora, since the actual parsing time is faster than any of the other models.

Note that the model size can affect usability of the parser, in terms of RAM requirements. Whereas a parser with a MaxEnt model can easily run on a small laptop with only 1 Gb of available RAM, the LinearSVM model requires at least 16Gb of available RAM, and is therefore reserved for powerhouse servers.

In terms of statistical significance, measured over the results for all evaluation corpora combined, the differences between the two MaxEnt models are not significant. Between all other models, the differences are very significant (p-value < 0.001).

In conclusion, unless model size or memory are an issue, the linear SVM model should be preferred, as it is both more accurate and faster for larger corpora. When a very large number of training sessions are required (to compare different training configurations), perceptron or MaxEnt models can be used, however, when comparing different feature sets, the different cutoff values mean not all features will be handled the same.

5.2.2 Evaluating classifers for pos-tagging

As mentioned at the start of the chapter, in order to keep the pos-tagging and parsing results consistent, we perform all pos-tagger training and evaluation on the FTBDep corpus, although far more pos-tag data is available in the original FTB corpus.

Classifier	Perceptron	MaxEnt1	MaxEnt2	LinearSVM
Cutoff	1	7	10	5
Other parameters	$i = 20$	$i = 100$	$i = 150$	$C = 0.25$ $\epsilon = 0.01$
LAS (mean)	86.78	86.39	86.46	**87.50**
FTBDep-dev	87.98	87.80	87.66	**88.84**
FTBDep-test	88.51	88.23	88.19	**89.35**
EstRépu	86.94	86.55	86.54	**87.61**
Europarl	87.87	87.05	87.20	**88.24**
FrWiki	86.23	85.90	86.06	**87.32**
FrWikiDisc	83.57	83.42	83.46	**84.60**
EMEA-dev	86.33	85.76	86.13	**86.57**
UAS (mean)	89.20	88.94	89.05	**89.95**
FTBDep-dev	90.13	90.04	89.97	**91.03**
FTBDep-test	90.63	90.47	90.40	**91.55**
EstRépu	89.91	89.74	89.69	**90.50**
Europarl	90.68	90.11	90.29	**91.05**
FrWiki	88.24	88.13	88.36	**89.43**
FrWikiDisc	86.74	86.46	86.63	**87.83**
EMEA-dev	88.10	87.66	88.00	**88.25**
Training time	**17m13s**	23m59s	31m52s	1h08m10s
Analysis time	4m46s	1m48s	**1m46s**	4m19s
Setup time	1m29s	9s	**9s**	2m45s
Parsing time	3m17s	1m39s	1m37s	**1m34s**
Model size	243 Mb	15 Mb	**11 Mb**	491 Mb

Table 5.2: Comparison of the best classifier configurations for parsing

In terms of feature counts for various cutoffs (see section 4.4.1, page 116), the FTBDep training corpus gives the following counts:

Cutoff	**Feature count**
1	3,314,151
2	1,185,342
3	628,964
4	391,212
5	276,032
7	184,608
10	126,697
12	106,632

5.2.2.1 Tuning perceptron parameters for pos-tagging

As was the case for parsing, we have two main parameters to adjust: the training iterations and the cutoff. We ran a test with iteration values in {10, 20, 30, . . . , 70} and cutoff values in {1, 2, 3, 4, 5}. The results are shown in fig. A.5, page 222. As was the case for the parser,

applying a cutoff to perceptron training systematically lowers the scores. On the other hand, unlike parsing, the pos-tagger seems to attain better results with more training iterations, with 70 iterations generally providing maximal results, except in the one case of the out-of-domain EMEA-dev corpus, where 20 iterations give a max result, and 70 about 0.2% lower. We thus select a cutoff of 1 and 70 iterations for the perceptron trainer.

5.2.2.2 Tuning MaxEnt parameters for pos-tagging

As was the case for parsing, we measured accuracy for iteration values in {50, 75, 100, 150, 200} and cutoff values in {1, 3, 5, 7, 10, 12}, for all evaluation corpora. The results are shown in fig. A.4, page 221. Whereas the pos-tagger results confirm the need for as many iterations as possible, in terms of cutoff, we see a net difference between the FTBDep development corpus and all other corpora. The development corpus reaches a maximum at a cutoff of 3, and then tapers off as the cutoff increases. All other corpora show a preference for higher cutoffs: 5 in the case of the fairly similar EstRépu journalistic corpus, and 10 or 12 for the most distant corpora, FrWikiDisc (informal discussions) and EMEA-dev (technical medicine reports). Thus, as expected, the ideal cutoff value varies with corpus distance. Although almost all of the curves have inexplicable troughs at certain cutoffs, we select a cutoff of 10 with 200 iterations as most promising for the widest variety of corpora.

5.2.2.3 Tuning linear SVM parameters for pos-tagging

In the case of linear SVMs, as was the case for parsing, we have three main parameters to tune: the soft margin parameter C and the insensitivity zone width parameter ϵ (see section 2.8.4, page 68), and the cutoff.

Figure A.6 on page 223 shows the accuracy results of a grid search between values of C in the set $\{2^{-4}, 2^{-3}, \ldots, 2^{3}\}$ and values of ϵ in the set $\{10^{-3}, 10^{-2}, 10^{-1.5}, 10^{-1}\}$. The results concern all of our evaluation corpora, with the baseline features and a cutoff of 7. It is difficult to draw any definite conclusions from these results: the curves for $\epsilon = 10^{-3}$, 10^{-2} and $10^{-1.5}$ are all fairly close, without any obvious winner. Similarly, the maximum value for C typically lies between 2^{-3} and 2^{1}, but the curves are so completely different that it is not easy to choose the best value for C.

We now turn to cutoff, retaining $\epsilon = 10^{-2}$, and testing C in the set $\{2^{-3}, 2^{-2}, 2^{-1}, 2^{-0}, 2^{1}\}$ and cutoff in the set {1, 3, 5, 7, 10, 12}, in the hope that a clear winner will emerge. The results are shown in fig. A.7 (page 224). While the graphs show less seemingly random variation than the previous ones, we still have varying behaviour depending on the corpus, with most graphs (except for the technical corpus EMEA-dev) showing a preference for lower cutoffs. Generally, we seem to get near optimal results with $\epsilon = 10^{-2}$, $C = 2^{-1}$ and a cutoff of 3, and this is the model retained for comparison.

5.2.2.4 Comparing the best configurations for pos-tagging

We have thus selected three configurations for final comparison: the perceptron classifier with 70 iterations and a cutoff of 1 (Perceptron), the MaxEnt classifier with 200 iterations and a cutoff of 10 (MaxEnt, and the linear SVM classifier with a cutoff of 3, $C = 0.5$ and $\epsilon = 0.01$ (LinearSVM).

Figure 5.6 shows a comparison of accuracy for all evaluation corpora for the three configurations retained. The models are very close to each other, with different models getting the

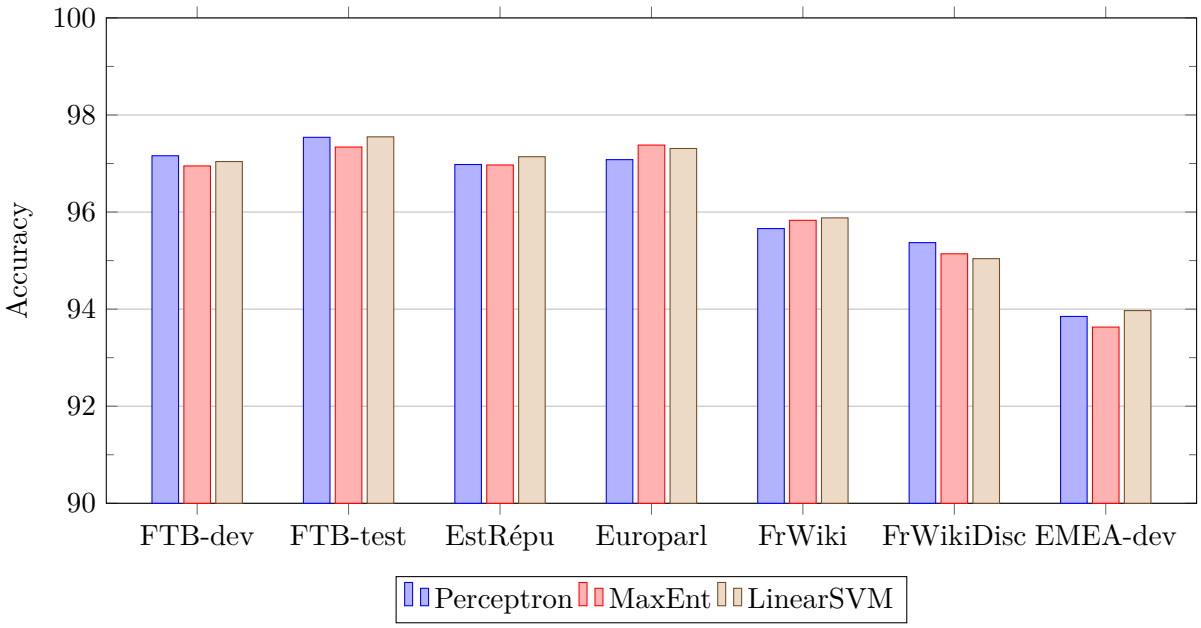


Figure 5.6: Pos-tagging classifier comparison: accuracy by corpus

highest score for different corpora, and no clear patterns. Table 5.3 gives more exact comparison statistics. The results are indeed so close that we can ask if the difference is statistically significant. Testing with McNemar's test on all of the evaluation corpora combined, we find significant differences between MaxEnt and both Perceptron and LinearSVM (p-value<0.001), but insignificant differences between Perceptron and LinearSVM (p-value>0.20). In terms of these results, we would tend to prefer the LinearSVM classifier, unless training speed and model size are an issue. However, in the current thesis, we are particularly interested in the model which gives the best results when combined with parsing. To this end, we now review methods for combining the pos-tagger and parser in both training and analysis.

5.2.3 Combining the pos-tagger and the parser

In the above tests we trained the parser models using gold standard pos-tags. A potential problem with this approach is that in real situations, only the pos-tags predicted by the pos-tagger are actually available to the parser. In order to make the parsing model more robust to the type of pos-tag it is likely to receive, we can attempt to train it on predicted pos-tags rather than gold standard pos-tags. To this end, a 10-way jackknifing has become the standard in many studies [Candito et al., 2010b, Zhang and Nivre, 2011]: the training corpus is divided into 10 sections, and the pos-tags for each section are predicted using a pos-tagger model trained on the nine remaining sections. We then use these predicted pos-tags, instead of the gold standard pos-tags, as training input to the parser. Note that this implies that the parser is being trained on "noisy" input: for example, if the pos-tagger mistakenly predicted that a verb was a noun, the parser will be learning to attach subjects and direct objects to nouns. The assumption is that this noise makes the parser more robust in real situations.

We run the following three tests:

1. Parsing accuracy based on gold standard pos-tag inputs. This is the ideal situation,

Classifier	Perceptron	MaxEnt	LinearSVM
Cutoff	1	10	3
Other parameters	$i = 70$	$i = 200$	$C = 0.5$ $\epsilon = 0.01$
Accuracy (mean)	96.23	96.18	**96.28**
FTBDep-dev	**97.16**	96.95	97.04
FTBDep-test	97.54	97.34	**97.55**
EstRépu	96.98	96.97	**97.14**
Europarl	97.08	**97.38**	97.31
FrWiki	95.66	95.83	**95.88**
FrWikiDisc	**95.37**	95.14	95.04
EMEA-dev	93.85	93.63	**93.97**
Training time	9m35s	**5m11s**	10m47s
Analysis time	40s	**23s**	50s
Setup time	22s	**6s**	33s
Tagging time	18s	**17s**	18s
Model size	51 Mb	**2 Mb**	117 Mb

Table 5.3: Comparison of the best classifier configurations for pos-tagging

giving a maximum possible score. It is assumed all other scenarios will get lower scores.

2. Parsing accuracy when trained on gold standard pos-tags, based on model-predicted pos-tag input

3. Parsing accuracy when trained on jackknifed pos-tags, based on model-predicted pos-tag input (jack-knifed pos-tags and pos-tagger use identical pos-tag training configuration)

All tests are run with the LinearSVM parser, as this gives the highest scores consistently. Tests 2 and 3 are run for each of the three best pos-tagging configurations: Perceptron, MaxEnt and LinearSVM.

Figure 5.7 shows the LAS loss per corpus and per method, with respect to the gold-standard pos-tags. Looking at the relative scores for individual evaluation corpora showed no clear winner. We therefore move to considering the total accuracy when all evaluation corpora are combined, as shown in table 5.4. Although using a linear SVM pos-tagger with a jack-knife trained parser gives slightly higher scores, the small difference between the methods begs the question of significance. Indeed, when we compare all of the other methods to the top-scoring method, none of them show significant differences. One thing is certainly clear: although jack-knifing is used almost universally, it does not provide statistically significant gains. In view of these results, we feel free to choose our winning combination based on other criteria than accuracy. We select the MaxEnt pos-tagger, due to its quick training time, its small size, and the ease with which we can explore the model, and combine it with the LinearSVM parser, which clearly gives the highest scores by a significant margin.

Another important fact arises from this discussion: the considerable effect of pos-tagging errors on the parsing score. Indeed, parsing LAS for all evaluation corpora falls by almost 3% when it is combined with pos-tagging. Since the pos-tagger scores are around 96.5%, we are getting more-or-less one parsing error for every pos-tagging error. In section 6.2.1 we will

attempt to identify the pos-tagging errors most responsible for parsing errors, so that we can concentrate our effort in pos-tagging improvement on these errors.

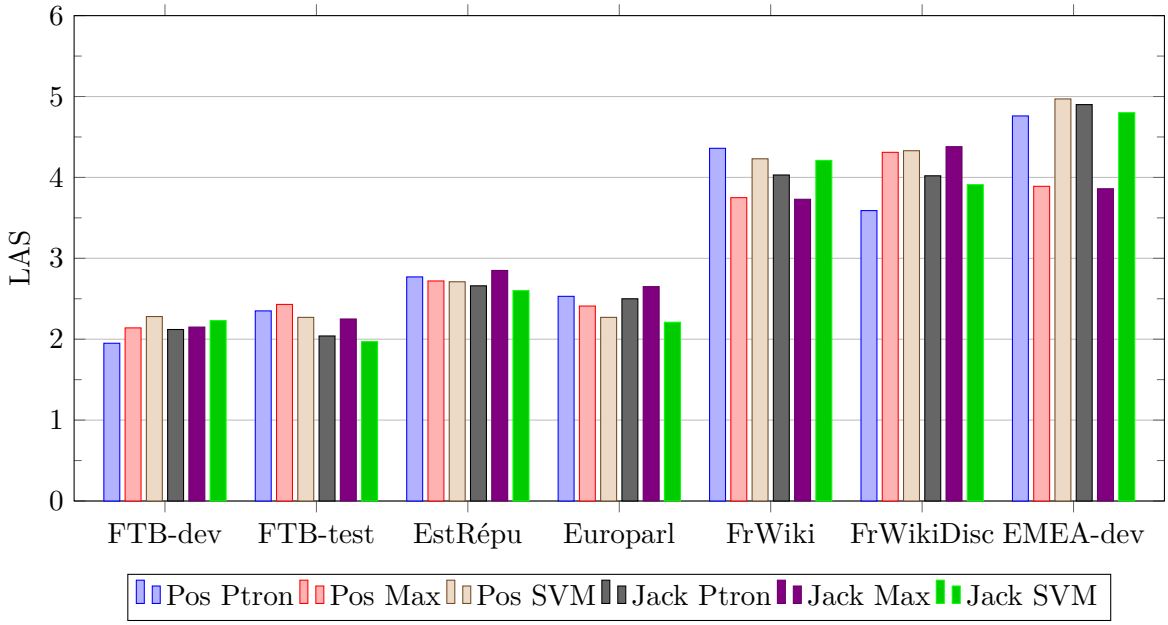


Figure 5.7: Accuracy loss (LAS) from gold-standard pos-tags, with and without jackknifing

Method	**LAS**	**UAS**
Gold	88.05%	90.34%
Perceptron postags	85.15%	87.91%
MaxEnt postags	85.19%	87.91%
LinearSVM postags	85.10%	87.87%
Perceptron + Jackknife	85.18%	87.91%
MaxEnt + Jackknife	85.21%	87.90%
LinearSVM + Jackknife	85.25%	87.96%

Table 5.4: Pos-tagging and parsing accuracy combined, with and without jackknifing

5.3 Experiments with system confidence

Having evaluated the various machine learning parameters and selected the best configuration, we now turn to experiments regarding the basic Talismane setup. Our first experiment attempts to prove the hypothesis that system confidence (i.e. the probability assigned to the best guess) is correlated to selecting the correct category. In other words, if the system is confident in its best guess, is it more likely to be right? It may seem obvious that the answer is "yes", since otherwise, how would we be getting accuracy in the region of 80% to 90%, far above random guessing. Still, the *degree* of correlation is important to a properly

functioning beam search, and may differ among classifier types. It is even more important for the various other system-confidence related methods presented in section 3.3.4.2, page 87, such as confidence-based filtering when constructing automatic resources. In this case, unlike the beam search, we have no direct way of measuring success, nor of determining the correct confidence cutoff value to use.

One way of measuring the degree of correlation is the point-biserial correlation [Tate, 1954], where the X variable is the system confidence, and the Y variable is 0 if the system selected the correct category, and 1 otherwise. We are particularly interested in the relative correlation for different classifiers, since, as was seen in section 2.8 (page 59), not all of the classifiers are naturally probabilistic.

The point-biserial correlation is given by the following formula:

$$r = \frac{M_1 - M_0}{S_n}\sqrt{\frac{n_1 n_0}{n^2}} \tag{5.2}$$

where M_1 is the mean system confidence when it guesses correctly, M_0 is the mean system confidence when it guesses incorrectly, S_n is the standard deviation for system confidence, n_1 is the number of times the system guessed correctly, n_0 is the number of times it guessed wrong, and n is the total number of linguistic contexts. It results in a value from -1 to 1, where -1 implies perfect negative correlation, 0 implies no correlation, and 1 implies perfect positive correlation.

For now, we limit ourselves to parser's confidence for the labeled dependency, ignoring pos-tagger confidence. On the FTBDep development corpus, this correlation is shown on table 5.5. Note that we also tested the correlation if the confidence measure is taken to be the product of parser confidence and pos-tagger confidence for the dependent, or the product of parser confidence, pos-tagger governor confidence, and pos-tagger dependent confidence: the best correlation is attained by taking into account the parser's confidence alone. The results shown are encouraging for both the MaxEnt and LinearSVM parsers, and much lower for the Perceptron parser, probably due to the fairly "flat" probability distribution which results from the various methods for transforming perceptron scores into probabilities. For now we are particularly interested in the LinearSVM parser, which gives the highest overall accuracy, together with the MaxEnt pos-tagger which was selected in the previous section. This combination also gives the highest confidence-to-correct parsing correlation: 0.4028.

Parser	Perceptron	MaxEnt1	LinearSVM
Perceptron pos-tagger	0.2043	0.3987	0.3982
MaxEnt pos-tagger	0.2062	0.3981	0.4028
LinearSVM pos-tagger	0.2077	0.4008	0.4016

Table 5.5: Point-biserial correlation for system confidence to correct parses on the FTBDep development corpus

Perhaps even more interesting is the concept of a confidence cutoff, where we only take into account dependencies whose confidence is above a certain value when constructing resources from automatically annotated corpora. Figure 5.8 shows, in blue, the accuracy of dependencies above a given confidence cutoff and, in red, the percent of dependencies remaining, for the FTBDep development corpus, using a Linear SVM parser and MaxEnt pos-tagger. The overall accuracy for this corpus 87.2%. If we apply a cutoff at a confidence of 70%, we get

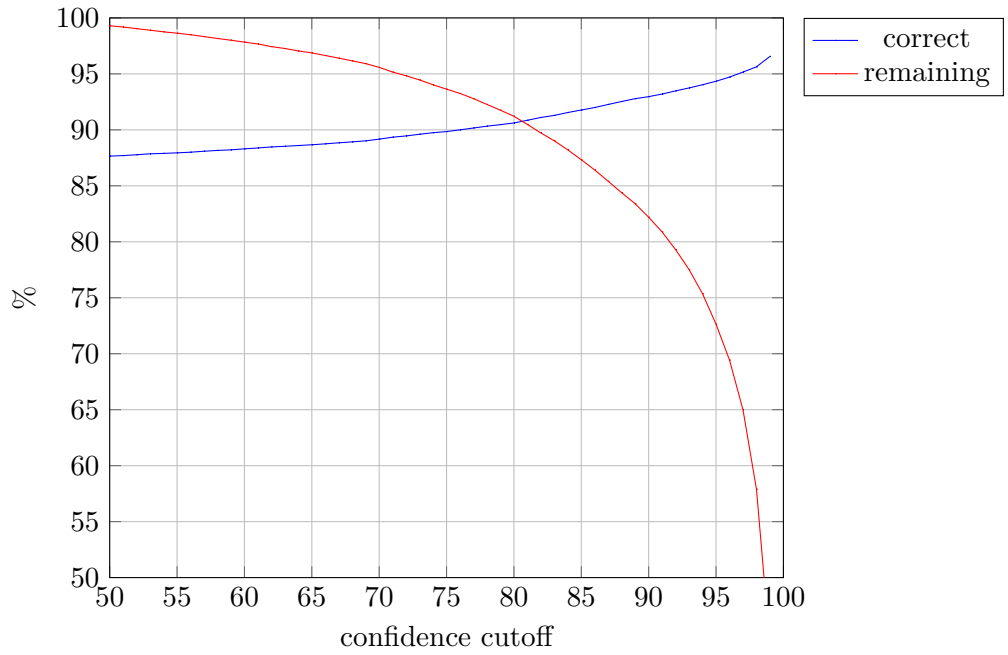


Figure 5.8: Correct answers and remaining dependencies based on confidence cutoff, for the FTBDep dev corpus

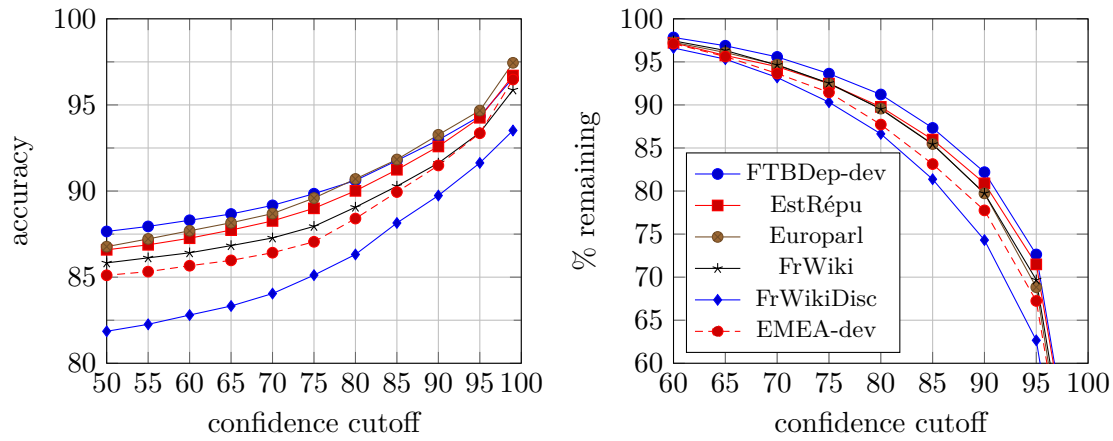


Figure 5.9: Accuracy and remaining dependencies based on confidence cutoff, for various evaluation corpora

89.2% accuracy, with 95.6% of the original dependencies. At 80%, we have 90.6% accuracy for 91.2% of the dependencies, at 90% we have 93.0% accuracy and 82.2% of the dependencies and at 95% we attain 94.3% accuracy for 72.6% of the dependencies. In fig. 5.9, we see that the graphs are similar for our various evaluation corpora, so that the concept of confidence cutoff seems applicable to more distant corpora as well.

Table 5.6 shows the point-biserial correlation for system confidence to correct parses by dependency label, as well as accuracy without applying a cutoff, and total count in the FTBDep dev corpus. Correlation varies widely, with high scores for `root`, `p_obj` and `suj`, and

Label	Correlation	Accuracy	Count
Total	0.4028	87.27%	31616
a_obj	0.3132	70.47%	359
aff	0.2110	92.05%	239
arg	-0.3170	29.41%	51
ato	0.0708	23.33%	30
ats	0.3171	76.32%	342
aux_caus	-0.1566	46.67%	15
aux_pass	0.3509	93.00%	243
aux_tps	0.3554	96.86%	509
coord	0.0773	57.99%	826
de_obj	0.0716	75.26%	287
dep	0.2910	81.25%	3301
dep_coord	0.3857	81.63%	931
det	0.4121	96.69%	5190
mod	0.3616	82.89%	7584
mod_rel	0.3038	69.79%	331
obj	0.4153	94.02%	7920
p_obj	0.5359	54.72%	265
root	0.5782	92.05%	1183
suj	0.5278	87.99%	1998

Table 5.6: Point-biserial correlation by label for system confidence to correct parses on the FTBDep dev corpus

relatively lower scores for `de_obj` and `coord`. The only relations with a negative correlation are those which are extremely rare. This gives us an idea of the types of relations we are likely to capture with more accuracy as we set the confidence cutoff higher. However, all relations with a positive correlation will benefit somewhat.

A last point of interest with confidence is the ability to predict Talismane's accuracy for a given corpus. Figure 5.10 shows LAS and UAS as a function of mean confidence, for the various available evaluation corpora. As can be seen, mean confidence is a reasonable predictor, although the regression lines have a fairly high standard error of 0.44% for LAS and 0.60% for UAS.

5.4 Experiments with beam search

The concept behind beam search was presented in section 2.6.1, page 55. Experiments with the beam search were already presented in Urieli and Tanguy [2013]. We reran the same experiments here with improved models, based on the baseline features from chapter 4. Moreover, whereas the previous study concentrated on MaxEnt, we now compare the beam search results for different classifier types. Finally, whereas the previous study used pos-tags learned from the full FTB corpus (minus the FTBDep evaluation corpora), we now limit ourselves to training on pos-tags from the FTBDep training corpus, thus giving more realistic picture of what is possible with a single training resource.

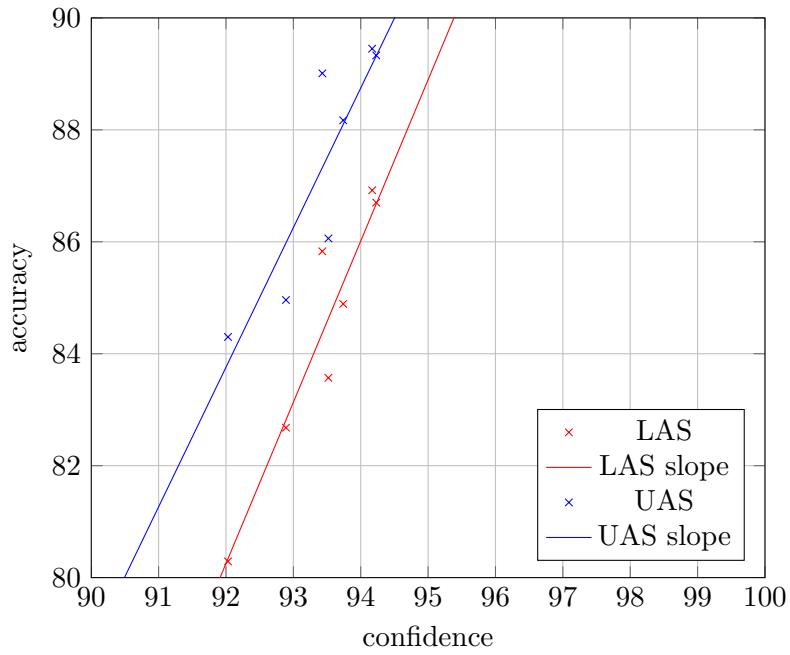


Figure 5.10: Mean confidence vs LAS/UAS

Regarding the tokeniser: a beam search on its own cannot affect tokeniser output, since the tokeniser contains no features which consider the tokenisation decisions already made in the current sentence. The only reason for using a beam search with the tokeniser is to attempt to correct tokenisation errors at a higher level of abstraction, via beam propagation. This will be explored in the following section. We therefore limit ourselves in the current section to the pos-tagger and parser.

5.4.1 Applying the beam to the pos-tagger

Our first question is: does the beam improve the pos-tagger's internal analyses, when the module is considered in isolation from the others? Note that a beam can affect the analyses through any features relating to decisions already made, especially through n-gram features. In order to remain consistent to other results in this chapter, we consider the pos-tagger as trained on the FTBDep training corpus, and tested on the FTBDep development corpus.

	Beam width				
Classifier	**1**	**2**	**5**	**10**	**20**
LinearSVM	97.04	97.07	97.07	97.07	97.07
MaxEnt	96.95	97.00	97.03	97.03	97.03
Perceptron	97.16	97.17	97.17	97.16	97.16

Table 5.7: Pos-tagger accuracy at various beam widths for different classifiers, on the FTBDep dev corpus

As can be seen in table 5.7, accuracy increases slightly for the LinearSVM and MaxEnt classifiers, and hardly at all for the Perceptron classifier, which has the highest score initially.

Note that these are the results with a perceptron analyser which is converted to a probability distribution using exponential normalisation, as presented in section 2.8.2. We also tested with standard additive perceptron scoring, where each analysis is assigned the cumulative sum of all individual decision scores, and with a linear normalisation, where all scores are scaled so that minimum=1, and then converted to a probability distribution. In the latter two cases, the accuracy decreases with the beam, and they will not be explored further.

The very minor increases for all classifiers beg the question: are these increases statistically significant? For the MaxEnt classifier (McNemar's test), the changes are statistically significant between a beam of 1 and 2, and between 2 and 5. Above 5, the output is strictly identical to that of beam 5. For the SVM classifier, none of the changes from one beam width to the next are significant. As was the case for the MaxEnt classifier, the output is identical above a beam 5. For the Perceptron classifier, none of the beams provide significant changes.

5.4.2 Applying the beam to the parser

Our next question is, when the parser is considered as an isolated module, can the beam improve results?

	Beam width				
Classifier	**1**	**2**	**5**	**10**	**20**
LinearSVM	88.84	89.14	89.42	89.42	89.48
MaxEnt	87.78	88.35	88.87	89.02	89.09
Perceptron	88.12	88.07	87.92	87.78	87.65

Table 5.8: Parser accuracy at various beam widths for different classifiers, for the FTBDep dev corpus

The results are shown in table 5.8 for the FTBDep dev corpus. The LinearSVM classifier improves by over 0.6% and the MaxEnt classifer by over 1.2% as we reach the higher beams. The perceptron algorithm does not behave well in beams above 1, and accuracy reduces. In terms of statistical significance (McNemar's test), for LinearSVM and MaxEnt, all changes from one beam to the next are signficant except for the LinearSVM from beam 5 to beam 10.

It is important to note that these improvements come with a price: the analysis speed is almost directly proportional to the beam width, so that beam 2 analysis takes twice as long as beam 1, and beam 20 takes 20 times as long.

5.5 Experiments with beam propagation

In most existing syntax analysers, each module provides the single best solution to the subsequent module. Thus, the pos-tagger will only provide a single pos-tagging solution to the parser. However, in some cases, the additional information available when parsing is necessary in order to disambiguate between various pos-tagging choices. We have thus decided in Talismane to enable **beam propagation**: when it is switched on, each module passes its entire beam to the following module. This doesn't mean we discard the scores assigned to each pos-tagging solution when parsing: the scores for the partial parsing solutions are each multiplied by the score for the underlying pos-tagging solution. Similarly, scores for partial pos-tagging solutions are each multiplied by the score for the underlying tokenisation solution.

Experiments with the beam propagation were already presented in Urieli and Tanguy [2013]. We reproduce the examples here for clarity. Furthermore, we now test beam propagation with improved statistical models using the baseline features presented in chapter 4. The tokeniser design has changed radically since the publication of this article as well, in the hope that the new design can help in better populating the tokeniser beam.

Level n: Tokenisation

Elle	pourrait	même s'		ennuyer	Score : 66%
Elle	pourrait	même	s'	ennuyer	Score : 34%

Level n+1 : Pos-tagging

Elle CLS	pourrait V	même ADV	s' CLR	ennuyer VINF	Tagging score	Tokenisation score	Total score
96%	99%	99%	88%	94%	**95%**	34 %	**32%**
Elle CLS	pourrait V	même CS	s'	ennuyer VINF	Tagging score	Tokenisation score	Total score
96%	99%	8%		24%	43%	**66%**	29%

Table 5.9: Beam propagation from the tokeniser to the pos-tagger

Table 5.9 shows an example of beam propagation for the sentence *"Elle pourrait même s'ennuyer"* ("She might even get bored"), and a beam width of 2. In this example, the word sequence *même s'* is ambiguous between the contraction of the subordinating conjunction *même si* ("even if"), which is tokenised as a single token, and the adverb *même* followed by the contracted reflexive clitic *se*. The correct tokenisation result here is to separate the two words. The tokeniser, however, makes the wrong choice, and assumes we have a single token. If there were no beam propagation, this incorrect tokenisation solution would be the only one available to the pos-tagger and parser. With beam propagation, the full beam is passed on from the tokeniser to the pos-tagger. The pos-tagger manages to place the correct solution on top of the beam, presumably because the sequence (V, CS, VINF), as reflected by trigram features, is vary rare in the training corpus. Similarly, we can imagine a case where the correct tokenisation solution is only placed on top when we reach the parser, or where the parser reorders the pos-tagging solutions to place a correct pos-tagging solution on top.

5.5.1 Using the parser and pos-tagger to correct tokenisation errors

Our first question is: can the pos-tagger and parser correct errors made by the tokeniser, when the correct response is placed by the tokeniser somewhere on the beam other than the first position.

In order to test this, rather than performing 10-fold cross validation for tokeniser tuning as was done in the previous chapter, we use the full FTB corpus corrected for tokenisation as per section 4.1.2, from which we removed the sections corresponding to FTBDep dev and train, which we save for evaluation. This allows us to attempt to correct tokenisation errors using pos-tagger and parser models trained on the FTBDep training corpus, since they have not seen any of the sentences in the tokenisation evaluation corpus. The resulting corpus has 81497 tokens, 3541 of which begin a potential compound word. For the tokeniser we limit ourselves to a MaxEnt model with a cutoff of 3.

The results on the FTBDep corpus are not conclusive. We concentrate on areas that are identified by tokeniser patterns only, as these are the only ones likely to change in the beam. Without propagation we attain an accuracy of 95.45%, whereas with propataion we have an accuracy of 95.48% with at beam width 2 (12 errors corrected, 11 introduced) and 95.56% with at beam width 5 (10 errors corrected, 6 introduced). However, a review of the errors shows that in most of the cases, the improved accuracy was due to an error in the original annotation, which was corrected by the tokeniser without beam propagation, and re-introduced by beam propagation. Indeed, of the respectively 12 and 10 errors corrected by beams 2 and 5, only three in each case are valid corrections. This seems to imply that beam propagation is not a good idea from the tokeniser.

To verify this tendency, we turn to the unannotated corpora described in section 4.2.3, page 112. We analysed these corpora with and without tokeniser beam propagation, at a beam width of 2 and 5, and examined the first 20 or so differences in each corpus (a total of 113 differences). In total, we have just under 1 difference per 1,000 words. The results are shown in table 5.10.

Total	**No propagation**	**Beam 2**	**Beam 5**
113	89	35	41

Table 5.10: Testing tokeniser beam propagation on unannotated corpora

This experiment confirms the assumption that tokeniser beam propagation is not a good idea: the best results by far are obtained without propagation. Indeed, almost all of the cases where the pos-tagger or parser corrected tokenisation concern the fairly arbitrary cases *il y a* and *plus de*. For *il y a*, it was decided in FTB to annotate it as a single token when we have a time expression, e.g. *"Il y a vingt ans... "*, and as multiple tokens when it indicates existence, e.g. *"Il y a vingt livres sur l'étagère"*. For *plus de*, there seems to be no hard and fast rule. Ideally, we would always mark it as a compound when it plays the role of a determiner "*Cette mesure est destinée à donner plus de sécurité juridique...* ", and only split it in the case where *plus* belongs to a previous construct, e.g. "*Mr Rocard n'a pas craint non plus de dire des choses difficiles aux patrons.*". However, neither of the above sentences marks it as a compound in the FTB: instead, the compound determiner is marked as compound $\approx$ 130 times, and as split $\approx$ 230 times, without any detectable reason for selecting one or the other. Corrections of *plus de* can therefore only be seen as arbitrary. There are no cases where the beam corrects a clearly ambiguous expression requiring careful contextual analysis, such as *bien que* where *bien* is an adverb and *que* a subordinating conjunction.

It may well be that the types of ambiguities left open by the tokeniser in the beam are not efficiently handled by a transition-based parser. They may require semantic information unavailable to the parser with the current feature set, or, as in the case of *bien que*, the parser may need to look at the total sentence structure to see if the preceding verb has a direct object or not, whereas transition-based parsing makes decisions in a linear sequence and has no direct visibility into the complete sentence structure until the parsing has completed.

5.5.2 Using the parser to correct pos-tagging errors

In this section, we look into the possibility of the parser correcting pos-tagging errors, via the ambiguities left on the beam by the pos-tagger. In view of the pos-tagging beam results

from section 5.4.1, we will only consider the LinearSVM and MaxEnt classifiers for both the pos-tagger and the parser.

	Beam width				
Classifier	**1**	**2**	**5**	**10**	**20**
PosSVM (no propagation)	97.04	97.07	97.07	97.07	97.07
PosSVM/ParseSVM	97.04	97.12	97.20	97.21	97.21
PosSVM/ParseMaxEnt	97.04	97.21	97.25	97.29	97.28
PosMaxEnt (no propagation)	96.95	97.00	97.03	97.03	97.03
PosMaxEnt/ParseSVM	96.95	97.04	97.12	97.12	97.13
PosMaxEnt/ParseMaxEnt	96.95	97.04	97.17	97.18	97.20

Table 5.11: Pos-tagger accuracy at various beam widths for different classifiers, for the FTB-Dep dev corpus, with and without propagation

Table 5.11 shows pos-tagger accuracy for the FTBDep dev corpus, using various combinations of LinearSVM and MaxEnt classifiers for the pos-tagger and parser, with and without propagation. As can be seen, the parser manages to correct far more errors with beam propagation than the pos-tagger on its own. In terms of statistical significance (McNemar's/binomial test), going from one beam to the next higher beam, only the move from 1 to 2 and from 2 to 5 are significant, except for the LinearSVM pos-tagger with MaxEnt parser, for which the move from 5 to 10 is significant as well. Between the identical beam width with and without propagation, for beam widths ≥ 5, all of the propagation changes are significant. At a beam width of 2, only the move from PosSVM (no propagation) to PosSVM/ParseMaxEnt is statistically signficant.

We now turn to the unannotated corpora described in section 4.2.3, page 112. We analysed these corpora with and without pos-tagger beam propagation, at a beam width of 2 and 5, and examined the first 20 or so differences in each corpus (a total of 109 differences). In total, we have 1.1 difference per 100 words at beam 2, and 1.3 differences per 100 words at beam 5. The results are shown in table 5.12.

Total	**No propagation**	**Beam 2**	**Beam 5**
107	29	53	71

Table 5.12: Testing pos-tagger beam propagation on unannotated corpora

This unannotated test amply confirms the usefulness of pos-tagger beam propagation: beam 2 propagation almost doubles the number of correct responses, and beam 5 propagation more than doubles them. A closer examination of results shows many corrections for the less common adjective-noun word order (e.g. "*le redouté bug informatique*") which would probably not affect the overall parse immensely, but also many very important fixes, such as recognising the verb in "*cette nouvelle année réserve encore...*", or the past participle in "*un habil feuilleton fait de dix épisodes*". There are also corrections for the negative *que*: "*et ne plus penser qu'à son bien-être*". Generally, as might be expected, the corrections take into account information farther afield than the trigram available to the pos-tagger in the baseline features.

In terms of cost, propagation does not add significant time to analysis, over and above the additional time taken by using a wider beam.

5.5.3 Using beam propagation to improve parsing

Having explored the extent to which tokenisation and pos-tagging errors are corrected by downstream modules, we now turn to an overall evaluation of Talismane with and without propagation. In other words, we see to what extent delaying the final pos-tagging decision helps to improve the parsing accuracy. To remain compatible with other results in this chapter, and also because it was already shown that beam propagation does not help tokenisation, we consider the pos-tagger and parser only, and leave out the tokenisation step.

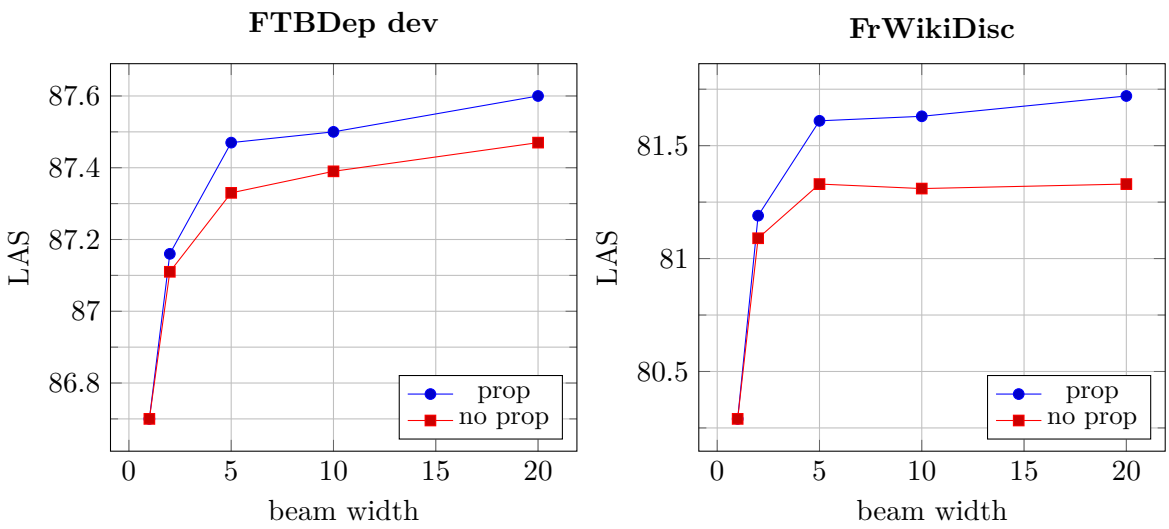


Figure 5.11: LAS with and without propagation for the FTBDep dev and FrWikiDisc corpora

First, fig. 5.11 shows results at different beam widths with and without beam propagation from the pos-tagger to the parser. We show the difference in LAS for the FTBDep dev and FrWikiDisc corpora. We chose to show results for the two most distant corpora, but all other corpora display similar behaviour. As can be seen, propagation systematically improves the results.

Figure 5.12 shows LAS for the various evaluation corpora at different beam widths, with propagation. The UAS curves are similar. Results tend to increase between 1% and 1.5% overall, and stabilise more or less after a beam width of 5, with minor gains afterwards. For FTBDep dev, the changes are significant (McNemar's test) from beam 1 to 2, 2 to 5, and 10 to 20, but not from beam 5 to 10.

One of the known issues with transition-based parsers, as opposed to graph-based parsers, is their propensity to erroneously privilege short-distance attachments, since the short-distance possibility is always considered prior to a longer distance possibility, and the two are never directly confronted. Using a beam helps overcome this propensity, by keeping attachment ambiguities in the beam a bit longer into the parsing process, thus enabling the system to compare shorter and longer distance possibilities in some cases. Figure 5.13 shows LAS for all dependencies up to a certain distance, where adjacent tokens are considered to have a

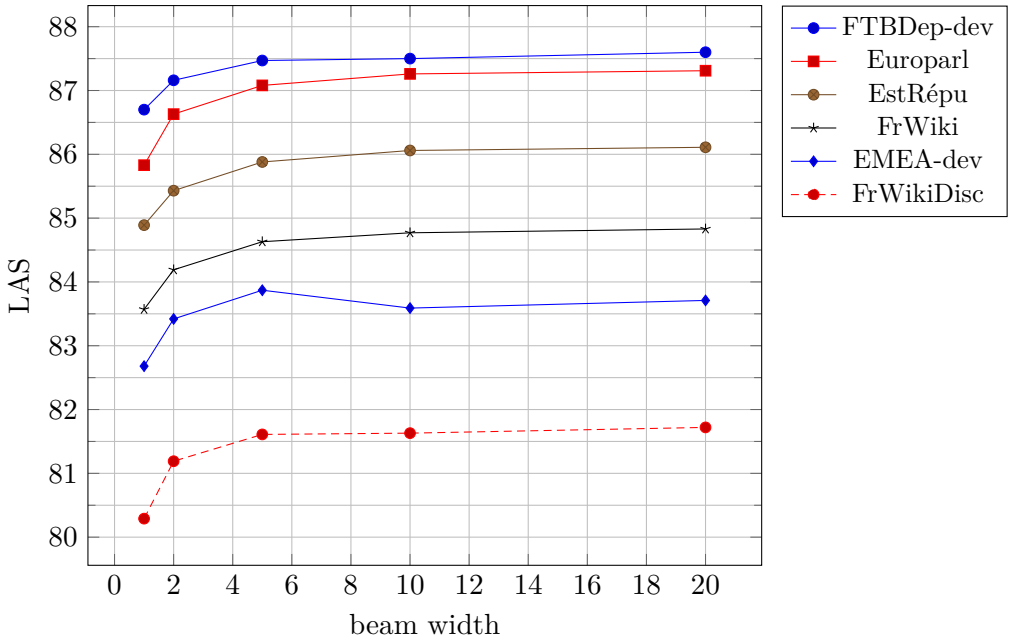


Figure 5.12: LAS by beam size, with propagation

distance of 1, for beams 1, 2, 5 and 10. Although the LAS lowers for all beams as the distance increases, the gap between the beams increases with the distance. Thus, the difference between beam 1 and beam 10 is 0.8% at distance 1, but 1.1% at beam 20. The difference between beam 2 and beam 10 is 0.29% at distance 1, but 0.38% at distance 20. Other corpora all show similar behavior, to a more or less marked extent.

Finally, different labels behave differently as the beam size increases. Table 5.13 shows the change in precision, recall and f-score, from beam 1 to beams 2 and 5, for each label appearing at least 1000 times in all evaluation corpora combined. It is noteworthy that precision generally increases only slightly, and all dramatic increases are in recall (changes above 2% shown in bold), possibly due to the ability to search farther afield for governors. The five relations benefiting the most from a wider beam are `coord`, `dep_coord`, `mod_rel`, `root` and `suj`.

5.6 Comparison to similar studies

The main equivalent study for French is Candito et al. [2010b], giving scores for pos-tagging and parsing combined for three radically different statistical parsers: the Berkeley parser [Petrov et al., 2006] which is a consituent-based parser, the MSTParser [McDonald et al., 2005], which is a graph-based dependency parser, and the MaltParser [Nivre et al., 2007b], which is a transition-based dependency parser. Of the three, Talismane is directly comparable to the MaltParser. For all this comparison, we have used Talismane with a MaxEnt pos-tagger, a LinearSVM parser, and beam propagation activated.

Table 5.14 shows a comparison of Talismane to the parsers in this benchmarking study. As can be seen, Talismane is slightly below the MaltParser scores at beam 1. As it turns out, there is little difference between the two setups: in the study, MaltParser uses pos-tags as input

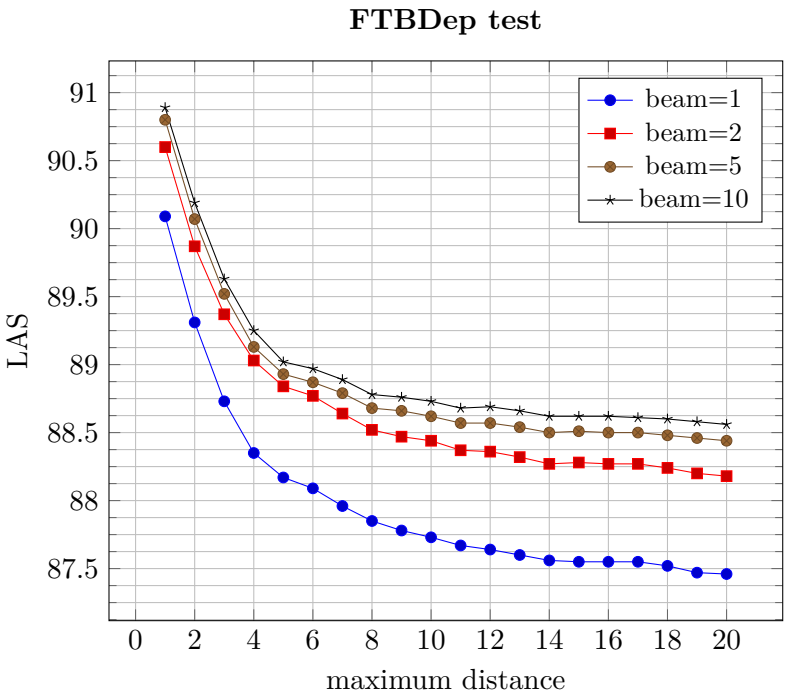


Figure 5.13: LAS for all dependencies where distance $\leq n$, at different beam widths, FTBDep test corpus

from the maximum-entropy trained MElt tagger [Denis and Sagot, 2012], and uses a linear SVM classifier trained on jackknifed pos-tags, whereas Talismane uses its own maximum-entropy trained tagger, and a linear SVM classifier trained on the gold pos-tags. As was shown before, tests with jackknifed pos-tags showed no significant improvement. Although it is not stated which final feature set has been used, most of the features considered are similar, with the exception of word form cluster features [Candito and Crabbé, 2009]. It seems likely that Malt Parser's better results at beam 1 are due to these features. Note also that Talismane's configuration parameters were tuned for the full set of evaluation corpora, rather than for FTBDep dev alone, which may well account for some of the differences.

As soon as we pass to the higher beams, Talismane's scores climb above those for the MaltParser, and approach the scores of the graph-based MSTParser. Although Talismane maintains linear time complexity, this does not necessarily mean it is faster: the analysis time is more or less multiplied by the beam width. Thus, Talismane can process ≈ 2 million words an hour at beam 1 (using the server setup given at the start of this chapter), 1 million words/hour at beam 2, and 400,000 words/hour at beam 5. Beyond beam 5, the accuracy gains seem insufficient to justify the lower analysis speed.

5.7 Discussion

In this chapter, we performed an evaluation of Talismane using a wide array of configuration parameters and methods. We explored different classifiers, system confidence measures, beam widths, and beam propagation. As a result, we were able to identify the best machine learning

Label	**Count**	Δ precision		Δ recall		Δ f-score	
Beam:		**2**	**5**	**2**	**5**	**2**	**5**
TOTAL	133631	0.65	1.02	0.65	1.02	0.65	1.02
a_obj	1496	-0.11	0.07	1.27	1.47	0.75	0.94
ats	1586	0.16	0.08	-0.37	-0.44	-0.15	-0.22
aux_pass	1327	-0.14	-0.12	0.68	1.58	0.28	0.76
aux_tps	2094	0.01	0.01	0.57	0.86	0.30	0.45
coord	3917	0.17	0.12	1.30	**2.68**	1.12	2.23
de_obj	1024	0.06	0.64	-0.10	0.00	-0.03	0.29
dep	13762	0.19	0.24	0.71	0.90	0.52	0.66
dep_coord	4239	0.13	0.23	0.59	1.88	0.41	1.24
det	21385	0.08	0.08	0.25	0.29	0.18	0.20
mod	31175	0.13	0.20	0.73	1.02	0.47	0.67
mod_rel	1410	0.31	0.53	0.99	**2.62**	0.78	1.95
obj	33000	0.03	0.03	0.42	0.58	0.23	0.32
p_obj	1325	0.18	0.20	0.22	0.37	0.24	0.37
root	5967	0.07	0.03	1.32	**2.58**	0.82	1.54
suj	8693	-0.15	-0.11	1.61	**2.58**	0.83	1.38

Table 5.13: Change in precision, recall and f-score from beam 1 by label, for all corpora combined

Parser	**LAS Dev**	**UAS Dev**	**LAS Test**	**UAS Test**
Berkeley	86.5	90.8	86.8	91.0
MSTParser	87.5	90.3	88.2	90.9
MaltParser	86.9	89.4	87.3	89.7
Talismane beam 1	86.7	89.3	86.9	89.5
Talismane beam 2	87.2	89.7	87.5	90.0
Talismane beam 5	87.5	90.0	87.8	90.2
Talismane beam 10	87.5	90.0	87.9	90.4
Talismane beam 20	87.6	90.1	88.0	90.4

Table 5.14: Comparison of the Talismane baseline to other parsers for French

setup for our task (MaxEnt pos-tagger and linear SVM parser with appropriate parameters).

Regarding the classifiers, the MaxEnt classifier shows interesting behavior with respect to cutoff, for both the parser and pos-tagger, with a lower cutoff being better for corpora similar to the training corpus, and a higher cutoff being better for more distant corpora. This seems logical, as a higher cutoff only keeps those features which appear the most often, and are therefore the most likely to be generalisable. Linear SVM classifiers do not seem to function in the same way, and a full grid search is necessary to find the best parametrisation: however, in parsing, they give the best results by a clear 1%. Perceptron classifiers give promising baseline results, but do not seem to convert well to probability distributions, and do not behave well in a beam.

We were able to show interesting results in terms of system confidence, which seems

especially promising as a means of constructing automatic resources. Performing a confidence cutoff when constructing these resources allows us to reduce noise dramatically, at the cost of keeping only a portion of the dependencies identified by the parser.

In terms of the beam, it does not seem worthwhile to use a beam width above 5, given the linear increase in time required for analysis, and the negligible gains in accuracy. Beam propagation is useful between the pos-tagger and parser, but propagating the tokeniser's beam does not improve tokenisation results, probably because the type of ambiguities left open are not resolvable with the information the transition-based parser has available.

We have now established a baseline setup for Talismane. In the remainder of this thesis, we will attempt to see to what extent this baseline score can be improved using various methods for injecting additional linguistic knowledge using targeted features, rules and external resources.

Chapter 6

Targeting specific errors with features and rules

Having identified the best baseline configuration for Talismane, we now turn to a series of experiments that target improvements in specific phenomena. This chapter is, in many ways, at the very heart of the thesis: for the first time, we try to consciously inject linguistic knowledge into the robust statistical process, and see to what extent the results are affected.

To this end, we use two very different mechanisms: targeted features which aim at improving the statistical modelling of specific phenomena, and rules which bypass the statistical model in specific cases. Our goal here is a difficult one: we would like to design targeted features and rules that generalise well, but limit ourselves to exploring the training and development corpus for guidance. Indeed, the methodology developed in this chapter is centered around identifying errors in the development corpus, and then projecting features and rules onto the training corpus. This means that the only guidance we have regarding the ability of our features and rules to generalise is our knowledge of the language and the type of constructs that are allowable in our targeted sub-languages.

One of our initial questions is: when should we use features, and when should we use rules? This question is dealt with in section 6.1, along with an explanation of the significant differences between the two.

Turning then to targeted features, we first tackle the pos-tagger in section 6.2, attempting to identify the pos-tagger errors most responsible for parsing errors, and then concentrating on the function word *que* (section 6.3). We then turn in section 6.4 to targeted parser features, concentrating on coordination as a case study.

In section 6.5, we attempt to use rules to correct analysis errors in those cases where it seems unlikely that features will help. We look at three different rule types: closed-class constraints for the pos-tagger, specific pos-tagger rules around the function word *que*, and parser rules for duplicate subjects and for phrases within parentheses.

We base all of our experiments on a baseline system with a beam width of 1. This allows us to measure accuracy gain at the highest analysis speed. In some cases, we also compare the results with those for beam 2, to ensure the gains are not lost in higher beams. We evaluate all changes against both the annotated evaluation corpora and the much larger unannotated corpora.

6.1 Features or rules?

Features and rules are defined similarly, but their behavior is radically different. Features are used to define statistical tendencies: in the robust classifiers we use, there is no need for features to be independent, and two co-occuring features can even contradict each other. For example, one pos-tagger feature might state that, when the word *que* follows the word *ne* earlier in the sentence, it is most often an adverb. Another feature might state that when the word *que* immediately follows the word *fait* (e.g. "*le fait que...*") it is most often a subordinating conjunction. Now, take the following sentence:

Example 6.1 *Personne **n'**a remarqué le **fait** qu'Aziz n'était plus là.*

Both features will activate for this sentence, and it is up to the classifier to weight them correctly and take a decision.

Rules, on the other hand, are deterministic: if a positive rule applies in a certain case, it will be applied to the exclusion of all other possible decisions. The order in which the rules are defined can thus be critical. In terms of negative rules, which exclude a certain decision in a certain context, the order is not important, but the decision is nonetheless absolute, and therefore has to be defined with the utmost care.

One distinction between features and rules thus lies in their sensitivity to error. A feature can be used to capture tendencies: if X is true, than the classification is somewhat more likely to be Y than Y′. Rules, on the other hand, attempt to capture absolute truth: if X is true, than the classifiction must (or else cannot be) Y.

In general, therefore, features offer many advantages over rules, in that they do not need to define absolute truths, and we leave it to our classifier to make sure that their weights reflect the reality of their occurrences and associated labels in the training corpus. They can thus be made to apply to a much wider range of phenomena. However, there is no guarantee that even the best-defined feature will affect the probability distribution of decisions sufficiently to tip the scales in favor of the correct decision: features generally co-occur with a large number and variety of other features, each of which affect the decision to a greater or lesser extent in a certain direction. Trying to understand why a feature was not assigned enough weight in a certain context is a question of such mathematical complexity, that even if we do fully understand the mechanism via which the classifier calculated its weights, we still cannot give a solid answer. The best we can do is attempt to design features that, when projected onto the training corpus, cover as many occurrences as possible, and are as biased as possible in favor of one or more target decisions at the expense of the others.

However, when projecting features on to the training corpus, we often realise that the number of actual training occurrences is very small. It may be true that *que* after *fait* is always a subordinating conjunction, but if there are only 11 occurrences in the FTBDep training corpus, as opposed to hundreds of occurrences for other features (e.g. the pos-tag n-gram feature where *que* directly follows a noun), is there any hope that this feature will affect the final decision?

If, after analysing the remaining errors, we notice that the feature was not effective, and if we feel that we are safe in making a deterministic decision, we can then turn to rules. Rules can thus cover obvious cases that are insufficiently attested in the training corpus. In addition, the type of user who can make use of rules is different from the type of user who can make use of features. To make use of features, the user has to have access to the training corpus, and must have a sufficient understanding of the toolset to train a new statistical

model. To use rules, the user need only understand the syntax for defining them, and the mechanism inside of which they will be applied. Furthermore, rules can be defined for a specific corpus. The simplest case would be a word list associated with pos-tags, to indicate that a particular specialised word is always a noun or an adjective in a given corpus. But one could easily imagine more complex cases, for example certain syntactic turns-of-phrase in a legal text. Thus, whereas features allow us to make the most of the available training material, attempting to extract generalisations that can be applied to other corpora, rules are a means of going beyond the training material to inject our more comprehensive knowledge of the language or of a specific corpus.

6.2 Using targeted pos-tagger features

We first look at targeted features for the pos-tagger. Our methodology for selecting features is as follows:

1. Select phenomena worth studying. In the case of the pos-tagger, we decided to look at the lemmas responsible for the highest number of parsing errors.

2. Examine specific cases of errors in the development corpus and imagine useful features. We limit this examination to the phenomena identified previously.

3. Write features using the Talismane syntax, and project them onto training corpus. For each feature result obtained (e.g. `true` or `false` for a boolean feature), we count the number of times it appears with each pos-tag, and closely examine the training corpus occurrences in the case of unexpected pos-tags. We then tune the features to eliminate noise, trying to achieve the greatest possible imbalance between pos-tags counts while retaining as many occurrences as possible. For difficult cases, this can require several iterations.

4. Train a model using the training corpus and the tuned features, and evaluate. If necessary, return to step 2.

Recall, however, that our best pos-tagging model has a cutoff of 10, so that any feature we create must appear at least 10 times in the training corpus to be taken into account.

Furthermore, features need to capture information beyond what is already contained in the baseline features. For example, in the case of *que*, recognising that it is always a subordinating conjunction (`CS`) after *autre* is not useful as a new feature, since this is already captured by the baseline feature $L_{-1}W_0$ (previous token lemma + current token word form). On the other hand, any feature that looks farther afield than 2 tokens to the right or left of the current token, or that groups tokens together into classes for more occurrences and more statistical weight, has the potential to improve on baseline feature results.

6.2.1 Identifying important pos-tagger errors

Our first step above is selecting phenomena worth studying. In our case, we have thus decided to identify those pos-tagger errors which wreak the most havoc on the parser by provoking the most parsing errors. To this end, we performed an automatic comparison of three versions of the FTBDep development corpus: (A) the manually annotated reference version, (B) the

corpus as analysed by the parser with gold pos-tags, and (C) the corpus as analysed by both the pos-tagger and parser. We calculated a parse error count for each token with an erroneous pos-tag in version C, measuring how many parse errors the pos-tag error provoked. This score was calculated by examining the token's governing arc (governor and label), and all of the token's dependent arcs (dependent and label), as well as the dependents' dependent arcs when all other pos-tags in the dependency chain were correct (thus implying the governor's pos-tag error propagated errors). We only included an arc in the parse error count if it was correct in the version B with gold pos-tags but incorrect in the version C with guessed pos-tags—thus implying that the parsing errors were provoked by the wrongly guessed pos-tags.

We then totaled the parse error counts by pos-tag pair (correct/wrong) and lowercase word form (with the exception of numbers, which were grouped together), and looked at the entries with the highest total scores. The results for the combinations causing the most errors are shown in table 6.1, for function words and numbers only. The total number of dependencies (excluding punctuation) was 31826, and the total number of errors was 4234, of which 684 (16%) were provoked by pos-tag errors. Thus, 318 errors have to be fixed in order to increase the FTBDep dev accuracy by 1%, and 42 errors have to be fixed to lower the count of remaining parse errors by 1%.

Correct pos-tag	**Wrong pos-tag**	**Word**	**Parse error count**
DET	P+D	des	31
P+D	DET	des	29
CS	PROREL	que/qu'	20
DET	P	de/d'	15
ADJ	DET	123	14
P	ADV	+	13
CLO	DET	le/la/les/l'	11
PRO	DET	123	11
PROREL	CS	que/qu'	10
DET	ADJ	123	9
CLO	P	en	7
CS	P	comme	7
CS	CC	ainsi que	6
NC	DET	une	6
P	CC	c'est-à-dire	6
P	DET	de/d'	6
ADV	CS	que/qu'	6
CS	ADV	que/qu'	5
CC	P	comme	5
CS	ADVWH	quand	4
DET	P+D	du	4

Table 6.1: Pos-tagger function word and number errors causing the most parsing errors in the FTBDep dev corpus

We decided to concentrate on function words because they seem easiest to target and most likely to generalise across corpora. These include most notably the various forms of *de* (*des,*

du, de, d'), the various forms of *le* (*le, la, l', les*), as well as *que* (*que, qu'*), *comme* and *en*.

Among these, we selected the case of *que* as fairly representative of the process and methodology to be followed for pos-tagger error fixing. It is assumed that a similar methodology could be applied for the other pos-tag errors related to function words.

6.3 Improving the tagging of *que*

The problems with tagging the word *que* have already been explored by others, in particular Jacques [2005], who describes many different contexts in which *que* can be used, and proposes a method for correcting the tagging through both local surface rules and syntax analysis.

To summarise in terms of the annotation standards of the FTBDep corpus, there are six main possibilities for the token *que* (and its abbreviated equivalent *qu'*), annotated using 4 different pos-tags, as illustrated by the examples below:

1. Subordinating conjunction (`CS`): *Je pense qu'il a trop bu.*
2. Relative pronoun (`PROREL`): *Il boit le vin que j'ai acheté.*
3. Interrogative pronoun (`PROWH`): *Que buvez-vous ?*
4. Negative adverb (`ADV`): *Je n'ai bu que trois verres.*
5. Exclamatory adverb (`ADV`): *Qu'il est bon, ce vin !*
6. Comparative construction (`CS`): *Il a plus bu que moi.*

The baseline model confusion matrix for *que* in the full set of annotated evaluation corpora (including both dev and test sections) is given below, where rows represent the correct pos-tag and columns represent the guessed pos-tag:

Baseline	**ADV**	**CS**	**PROREL**	**PROWH**	**Total**
ADV	58	40	1	1	100
CS	20	917	49	1	987
PROREL	0	58	196	0	254
PROWH	0	5	15	4	24

Table 6.2: Baseline confusion matrix for *que*

We thus have a total of 190 errors for 1,365 occurrences (for a total corpus size of 152,141 tokens excluding punctuation), giving an accuracy of 86.08%. Note in the confusion matrix above two distinct squares: one concerning confusion between `CS` and `ADV`, and the other concerning confusion between `CS`, `PROREL`, and `PROWH`. We begin by tackling each of these separately: first attempting to separate negative adverbs from subordinating conjunctions, and then attempting to separate subordinating conjunctions from the various types of pronoun.

6.3.1 Recognising *que* as a negative adverb

We first tackle the problem of distinguishing *que* as a negative adverb from *que* as a subordinating conjunction. Following the methodology presented in section 6.2 above, the first step is to analyse errors in the development corpus in order to identify potential features.

6.3.1.1 Development corpus error analysis

In the FTBDep dev corpus, most of the errors resemble the following cases, where *que* is incorrectly tagged as a `CS`:

Example 6.2 *...mais **n'**ayant de comptes à rendre **qu'**/ADV à la présidence commune.*

Example 6.3 *Mais cela **ne** représente dans cette mouture, pour un couple avec deux enfants, **qu'**/ADV une prime maximale.*

In such cases, recognising *que* as a negative adverb is a simple matter of searching for a preceding occurrence of *ne* in the same sentence. There is no inherent limitation in distance, as illustrated by example 6.3, where the *que* follows several prepositional phrases.

Clearly, however, another intervening negative particle can complete the *ne*, after which the *que* is not generally an adverb, as in the following example:

Example 6.4 *Pour cela, il **n'**est **pas** question **que**/CS le zloty, la monnaie polonaise, soit "l'ancre de la stabilité" de l'économie polonaise.*

We thus refine our feature to search for a preceding *ne*, but without any other intervening negative particles (*aucun/e*, *autre*, *guère*, *jamais*, *ni*, *nul*, *pas*, *personne*, *plus*, *point*, *que/qu'*, *rien*).

However, two errors remain labeled by Talismane as `CS`, which are not compatible with the feature as it stands, since the intervening negative particle functions in combination with the negative *que*:

Example 6.5 *...ils **ne** sont **plus**, fin 1991, **que**/ADV 49 % à l' affirmer.*

Example 6.6 *...qui, faute de volonté politique, **ne** fut **jamais que**/ADV la caricature du système français.*

This sort of negative combination is possible with the negative particles *jamais*, *guère*, *pas*, *personne*, *plus* and *rien*, quite unlikely with *aucun/e*, *ni* and *point*, and impossible with *autre* (which, by convention, governs *que* as a `CS`) or another *que/qu'*. The negative combination construct is ambiguous by nature, as shown in the following two sentences:

- *Elle **ne** pense **pas qu'**/CS il viendra.*
- *Elle **ne** pense **pas qu'**/ADV à lui.*

Moreover, the ambiguity only exists for verbs which subcategorise a direct object with *que*, such as *dire* or *penser*. However, the training corpus contains 226 different verbs matching this criterion. In the case of a negative *que*, we decided not to make use of this information, since ambiguous cases are quite rare.

6.3.1.2 Feature list

The next step involved writing the features in the Talismane syntax, and projecting them onto the training corpus. After some refinement to increase feature scope and remove unwanted cases, we ended up with the following feature list. The numbers in the itemised lists refer to the number of occurrences for each feature result in the training corpus.

Unambiguous previous *ne*: return `true` if there is a preceding *ne* with no intervening negative particle (or only one that cannot combine with a negative *que*), and which is not itself preceded by a negative particle in the set {*personne*, *rien*, *aucun/e*, *nul/le*}. The latter restriction excludes sentences such as *"Personne ne sait que je suis venu."* Return `false` if there is no preceding *ne*. Otherwise, meaning if there is a preceding *ne* combined with a negative particle which renders the *que* ambiguous, we return nothing at all (thus not affecting the decision one way or the other).

- Previous unambiguous *ne* exists (total in training corpus: 233)
  - Pos-tag `ADV` (210).
    - Sample: *Très endettées, elles **n'**ont pu fonctionner durant des années **que**/ADV par un accès facile au crédit[...]*
  - Pos-tag `CS` (21). Well over half of these are annotation mistakes. The remaining cases are those where the particle *ne* is unaccompanied, following expressions such as *moin ADJ qu'on ne...* or variants of the verb *pouvoir*.
    - Sample: *[...] l'Amérique, moins superficielle qu'on **ne** l'imagine parfois, a entrepris une réflexion sur son identité bien avant **que**/CS [...]*
    - Sample: *[...] **ne** peuvent ainsi éviter **que**/CS, en la matière, l'histoire ne se repète.*
- No previous unambiguous *ne* (total 1623)
  - Pos-tag `CS` (1242).
  - Pos-tag `PROREL` (360).
  - Pos-tag `PROWH` (16).
  - Pos-tag `ADV` (5). Of these, 3 are annotation mistakes, in which a comparative *que* is annotated as an adverb, and 2 are exclamatory adverbs, as in the following sample.
    - Sample: *Mais pour parvenir à cela, **que**/ADV d'esprits à convaincre en France et plus encore au-dehors !*

We add two additional features to help the pos-tagger with cases involving negative particle combinations.

Negative *que* possible: is there a preceding *ne*, *ne pas* or *ne plus* whether or not it is followed by a negative particle, unless this particle excludes combination. This feature covers all cases where a negative *que* is possible. It is a less exclusive version of the previous feature.

- Negative *que* possible (total 389)
  - Pos-tag `ADV` (243). All occurrences of a negative *que* in the traning corpus.

  - Pos-tag `CS` (118). All cases of two negative particles, followed later in the sentence by a *que*.
  - Pos-tag `PROREL` (27). Same scenario as `CS`.
  - Pos-tag `PROWH` (1). Same scenario as `CS`.

- Negative *que* impossible (total 1646)
  - Pos-tag `CS` (1259).
  - Pos-tag `PROREL` (366).
  - Pos-tag `PROWH` (16).
  - Pos-tag `ADV` (5). These are exactly the same five as returned by the previous feature.

Nearby negative particle combination: this feature concentrates on the case where there is a second negative particle which can combine with a negative *que*. We noticed in the training corpus that cases of negative particle combination with *que*, there is generally a short distance between *que* and the negative particle ($\leq$ 6 tokens). We therefore return `true` if there is a *ne* followed by a negative particle capable of combination at a distance $\leq$ 6, `false` under the same circumstances but if the distance $>$ 6, and nothing at all otherwise.

- Negative particle combination, short-distance (total 103)
  - Pos-tag `ADV` (33). As can be seen, adverbs form almost 1/3rd of the cases for the short-distance case, as opposed to no valid adverbs in the long-distance case.
    - Sample: *Les spéculateurs sont désormais certains que la dévaluation **n'**est **plus qu'**/ADV une question de jours.*
  - Pos-tag `CS` (66).
    - Sample: *Il **n'**est **pas** exclu **que**/CS, d'ici là, ils fassent connaître leurs craintes avec insistance.*
  - Pos-tag `PROREL` (6).
  - Pos-tag `PROWH` (1).

- Negative particle combination, long-distance (total 77)
  - Pos-tag `ADV` (1). There is another intervening *ne* here - this occurrence could be excluded by further feature refinement.
  - Pos-tag `CS` (51).
    - Sample: *Si cela **n'**était **pas** possible, les Onze poursuivraient leur chemin sans perdre l'espoir **que**/CS cela se ferait plus tard.*
  - Pos-tag `PROREL` (25).

6.3.1.3 Results

This feature test gives excellent results when tested on the full set of evaluation corpora (dev + test). We show below the confusion matrix for *que* after applying the new features for a negative *que*. Numbers in parentheses show the change from the original confusion matrix.

	ADV	**CS**	**PROREL**	**PROWH**
ADV	88 (+30)	11 (-29)	0 (-1)	1
CS	10 (-10)	926 (+9)	51 (+2)	0 (-1)
PROREL	1 (+1)	57 (-1)	196	0
PROWH	0	5	2 (-13)	17 (+13)

Table 6.3: Confusion matrix for *que* with targeted negative adverb features

We have reduced cases where an ADV is reported as a CS from 40 to 11, and where a CS is reported as an ADV from 20 to 10. Some of the cases we have corrected are far beyond the capabilities of an n-gram model, e.g. *Ce dernier [...]* ***n'****avait, à ce moment, accepté de finir son mandat de trois ans* ***qu'****/ADV à condition qu'un successeur lui soit désigné.* Note in passing that, without attempting to do so, we have also corrected a large number of errors for *que* as an interrogative pronoun (`PROWH`).

For the word *que* viewed on its own, there are 62 new corrections for only 10 new errors. Note, however, that features in robust statistical classifiers never act alone, and adding new features necessarily affects the weights of all other features via the features with which they co-occur (pos-tag n-grams, etc.). We therefore have considerable side effects on other tokens, with a net positive influence: 166 new corrections outside of *que*, for 93 new errors. An examination of these errors yields nothing systematic.

We also examined the 23 remaining errors for *que* as an adverb, to see which cases were missed: 12 are naturally ambiguous negative particle combinations, 6 are simple negations which we would expect to be resolved correctly by the new features, 2 are annotation mistakes, and the rest are other rare cases. Regarding the 6 simple negations, the feature apparently was not assigned enough weight by the training mechanism to enable correct annotation. Questions such as feature weight are outside our control, and so this sort of problem can only be handled by refining the existing features, by introducing additional evidence in terms of more features, by identifying and removing other features which add noise, or by adding rules (section 6.5.2 below).

6.3.2 Recognising *que* as a relative or interrogative pronoun

Unlike the case of the *que* as a negative adverb, where the presence of a previous *ne* gave a very strong surface indicator, there is no simple surface indicator for distinguishing *que* as a relative pronoun from *que* as a subordinating conjunction, given the information available to the pos-tagger.

6.3.2.1 Development corpus error analysis

Following the methodology described in section 6.2 above, we first examine the development corpus errors to attempt to identify useful features.

Example 6.7 (...) *la Commission des opérations de bourse (COB) a annoncé le 14 janvier* ***qu'****/CS elle saisit la justice [...]*

With the baseline model, this occurrence is annotated as `PROREL` instead of `CS`. Through a cursory examination, we can imagine several features that can help us here: first of all, there are only certain verbs which subcategorise with *que* as a direct object, and *annoncer* is one of them. Secondly, the verb *annoncer* is transitive, but has no direct object prior to the word *que*. Furthermore, the verb following *que* does have a direct object (*justice*), which would not usually occur with a relative pronoun *que*, except in certain rare cases. Note also that the ambiguity between `PROREL` and `CS` only exists when there is a noun between the main verb (*annoncé*) and the *que*, in this case *janvier*. The nature of this noun as a month name is another indicator: it would be quite rare for a month name to be modified by a relative clause, and extremely rare for a month name to be a verbal argument rather than an adjunct.

Example 6.8 *Les investisseurs étrangers [...] devront échanger leurs devises à un taux* ***que****/PROREL le gouvernement veut situer entre 8 et 10 roubles [...]*

In this case, the baseline model annotates the occurrence as `CS` instead of `PROREL`. Here the opposite holds true: the main clause verb *échanger* already has a direct object, while the relative clause verb *situer* does not. Also, *échanger* cannot subcategorise with *que* as a direct object.

Example 6.9 *Le fait* ***qu'****/CS ils aient accepté de reprendre les pourparlers est interprété de façon positive.*

Here we have some additional clues: certain nouns introduce subordinate clauses almost systematically (e.g. *fait*), and the subjunctive form *aient* is generally indicative (no pun intended!) of an independent subordinate rather than a relative clause.

Having identified some possibly useful indicators, we now need to translate them into features using only the knowledge available to the system at the moment when the decision has to be taken: the word forms of all tokens, and the pos-tags assigned to tokens to the left of *que*.

To this end, we put together a list of verbs which can subcategorise with *que* as a direct object. We took the full list of such verbs governing *que* as a subordinating conjunction in the training corpus (226 verbs in all), and removed verbs unlikely to subcategorise with *que* given our knowledge of French and an examination of the training contexts, leading to a final list of 129 verbs. We did not make an attempt to recognise complex verbal structures: for example, we accepted *rendre* on the list because of the expression *rendre compte que*. This list includes both *avoir* (from expressions such as *avoir peur que*) and *être* (from expressions such as *toujours est-il que* or *il est probable que*). Analysing complex verbal structures more finely would allow us to refine this list and eliminate noise.

6.3.2.2 Feature list

At this point we were ready to write the features using the Talismane syntax and project them onto the training corpus. We will not go into the full details of this iterative process, but present instead its final results in terms of final features and pos-tag counts within the

training corpus. Note that while examining the training corpus sentences in an attempt to increase the imbalance between the different pos-tags, certain additional features were identified not directly related to errors in the development corpus. The numbers in parentheses below indicate, for each feature result, the number of occurrences for each pos-tag in the training corpus. The final feature list is thus as follows:

Coordinated structure: if the current *que* directly follows a coordinating conjunction, find the category assigned to the previous *que*. We also consider *que* directly following a comma, as long as the next *que* in the sentence follows a coordinating conjunction. We're expecting in most cases for the current category to be the same as that of first conjunct, although in this case we can only guess that they are indeed coordinated, since syntax analysis has yet to be applied.

- Previous conjunct annotated `CS` (total of 53 in the training corpus)
  - Current conjunct annotated `CS` (51)
    - Sample: *Elle estime* ***que****/CS d'ici à l'an 2000 le spectateur aura accès à vingt chaînes de télévision et quinze radios, et* ***que****/CS l' audience [...]*
  - Current conjunct annotated `PROREL` (1)
    - Sample: *Encore faudrait-il* ***que****/CS, pour faire passer la pilule des réformes nécessaires—et* ***que****/PROREL beaucoup d'Italiens, en dépit de leur enthousiasme, risquent, une fois au pied du mur, de trouver plus amère que prévu, —qu'un gouvernement fort et surtout crédible se constitue.*
  - Current conjunct annotated `PROWH` (1)
- Previous conjunct annotated `PROREL` (total 5)—insufficient for cutoff
  - Current conjunct annotated `CS` (1)
  - Current conjunct annotated `PROREL` (4)

Has intervening noun: First, we need to find the nearest "content" verb previous to *que* (or to the first conjunct *que* in case of a coordination). The rules for recognising the previous content verb, given the information available to the pos-tagger, are not as straightforward as they may at first seem: we need to go backwards from the word *que* until we hit a verb (indicative, subjunctive, infinitive, imperative, present participle), but cannot stop at a past participle, since this could be a noun modifier, in which case it is systematically passive (e.g. *90% des automobilistes controlés*) and cannot take *que* as a direct object. Once we find this verb, we need to check if it isn't followed by a past participle, skipping certain intervening words (e.g. *n'a pas tout mangé*), since in this case, the past participle is the content verb. Having identified the previous content verb, we check whether any word between it and the following *que* is a common noun, proper noun, pronoun or interrogative pronoun, excluding days of week and month names. If this feature returns `false`, we are not expecting any relative pronouns, since there is no antecedent for them to modify.

- Has intervening noun = `false` (total 1074)
  - Pos-tag `ADV` (210)

  - Pos-tag `CS` (858)
  - Pos-tag `PROREL` (4)
    - Sample: *Adidas était une affaire merveilleuse qu'il aimait beaucoup,* ***que****/PROREL j'aimais beaucoup, mais je crois qu'il a compris ces derniers mois qu'il ne pouvait pas tout faire.* In this case we do not recognise the coordination, because there is no coordinating conjunction, and the comma is too ambiguous to be used as an indicator of coordination on its own.
    - The other 3 examples are all training corpus annotation errors: either *que* should be marked `CS`, or the preceding noun was not marked as a noun.
  - Pos-tag `PROWH` (1)

- Has intervening noun = `true` (total 685). This is the classic case of ambiguity which needs to be resolved by other features.
  - Pos-tag `ADV` (34)
  - Pos-tag `CS` (386)
    - Sample: *C'est à ce* ***prix****/NC seulement* ***que****/CS l'Ukraine peut convaincre sa population[...]*
    - Many of the cases are comparative expressions, which could be excluded from this feature if it were further refined.
  - Pos-tag `PROREL` (263)
  - Pos-tag `PROWH` (2)

Que **directly follows "explanatory" noun:** We compiled a list of nouns found in the training corpus to be typically governing *que* as a subordinating conjunction, often in a fixed expression: *doute* (from *nul doute que*), *enseigne* (from *à telle enseigne que*), *espoir*, *fait*, *idée*, *point* (from *au point que*), *prétexte* (from *sous pretexte que*), *preuve*, *propos*. We then checked the pos-tag of the following *que*.

- Follows explanatory word (total 27)
  - Pos-tag `CS` (26)
  - Pos-tag `PROREL` (1)
    - Sample: *[...] pour l'idée* ***qu'****/PROREL ils se faisaient de leur journal[...].*

As mentioned in the introduction to this chapter, this is a classic case where we allow ourselves to be guided (or misguided) by the examples found in the training and development corpora, in the hope that they will generalise well. The list of nouns that can govern *que* as a subordinating conjunction is definitely much larger than this. While reviewing differences in the unannotated corpora, we encountered by chance the example of *impression*, and there are certainly many others. Nevertheless, out linguistic intuition tells us that the nouns on this list, even if it is incomplete, and regardless of the corpus, are very likely to govern *que* as a subordinating conjunction.

There are also considerable differences in behavior for some of these nouns: while it would be quite difficult to imagine *fait* taking *que* as a relative pronoun outside of certain

fixed expressions (e.g. *"L'état des faits qu'il a exposé..."*, it is easy to imagine *espoir*, *idée*, *prétexte* or *preuve* as governing a relative clause. There are several consequences here: on the one hand, by grouping these words together with *fait*, we allow them to gain statistical weight from the many occurrences of *fait*, which is by far the most common of the lot. On the other hand, this might put a false heavy weight in favor of the subordinating conjunction choice for these words. It may well be that this particular feature serves no valid purpose, since *fait* on its own is already covered by the baseline features, whereas the other nouns are not nearly as systematic in their behavior. Or else, that the feature should be refined to group together only certain fixed idiomatic expressions.

***Que* following comparative word:** This is a set of several features trying to identify *que* as part of a comparative structure (annotated `CS` by convention). We check for adjective comparatives: whether it follows a comparative adverb at position $n-2$ (e.g. *plus*, *moins*, *aussi*, *davantage*, etc.) and an adjective or past participle at position $n-1$, giving the structure *plus ADJ que*. We check for simple comparatives, in a structure such as *davantage que Y*: whether *que* directly follows a comparative word (adverbs such as *plus*, *moins*, *autant*, *davantage* and adjectives such as *même*, *différent*, *tel*, etc.), except for *plus* when there is a preceding *ne*. Finally we attempt to identify more complex nominal comparative expressions, such as *autant de X que de Y* and *les mêmes X que Y*.

- Adjective comparative (total 110)
  - Pos-tag `ADV` (2)
    - Sample: *Aujourd'hui, le sobriquet n'est* ***plus*** *utilisé* ***que****/ADV tourné en ridicule.* The other example is a training corpus error.
  - Pos-tag `CS` (108)
- Simple comparative (total 80)
  - Pos-tag `ADV` (1) (annotation error)
  - Pos-tag `CS` (76)
  - Pos-tag `PROREL` (3). All three concern a variant of the expression *tel que*, and could be considered annotation errors, although they are borderline cases. The vast majority of similar *tel que* expressions are annotated as `CS`.
    - Sample: *Telle/ADJ* ***qu'****/PROREL elle existe, cette législation est inopérante.*
- Complex comparative (total 56)
  - Pos-tag `ADV` (1)—expression with "ne plus", could be excluded if we refined the feature further.
  - Pos-tag `CS` (54)
    - Sample: *[...]* ***plus*** *préoccupés par leurs rivalités internes* ***que****/CS par les attentes des médecins [...]*
    - Sample: *[...] touche* ***tout aussi bien*** *la province québécoise* ***que****/CS les populations indiennes.*
    - Sample: *[...] qui produisent désormais trois fois* ***plus d'****eau* ***que*** *de brut.*
  - Pos-tag `PROREL` (1).

  * Sample: *[...] conséquence de **plus de** vingt ans de déficits cumulés **que**/PROREL n'ont pu effacer les profits récents.* We could try refine the feature further by ensuring that after *plus de*, the *que* is followed by *de* as well, yielding *plus de X que de Y*, but would have to make exceptions for past participles, adverbs, time-phrases etc. as in *plus de X que prévu*, *plus de X qu'ailleurs*, *plus de X que la semaine dernière.*

Comparative expressions are among the most difficult to annotate correctly, both because of their natural variety, and because they are not bounded in distance. An *autant* early in the sentence can justify a *que* near the end. On the positive side, the *que* later in the sentence is often mandatory, so that we are guaranteed to find at least one instance of *que* that completes the comparative. We have not yet designed features that take this fact explicitly into account: a much deeper study of comparative structures and their various surface manifestations would be required to do so.

Previous verb sub-categorises *que* as direct object: For the previous "content" verb found above, we check whether or not it is on our list of 129 verbs. To reduce noise, we exclude *que* when it follows an explanatory noun, or any comparative word, since these are cases where *que* does not directly depend on the preceding verb. We would expect no subordinating conjunctions when this feature returns false.

- Previous verb does not sub-categorise *que* as direct object (total 280)
  - Pos-tag `ADV` (110)
  - Pos-tag `CS` (58). This number is much larger than expected, but contains a variety of phenomena, from tokenisation errors to complex sentences.
    * Sample: *Il vaut mieux perdre un salarié compétent qui sera plus motivé dans un autre projet professionnel extérieur qui lui **tient** à coeur **que**/CS d'avoir des salariés frustrés ou aigris.* In this case, the *que* depends on the distant comparative verbal expression *vaut mieux*, rather than the nearest verb *tenir*.
  - Pos-tag `PROREL` (112)
- Previous verb does sub-categorise *que* as direct object (total 934)
  - Pos-tag `ADV` (102)
  - Pos-tag `CS` (739)
  - Pos-tag `PROREL` (92)
  - Pos-tag `PROWH` (1)

We also combine the previous feature with the existence or not of an intervening noun, further isolating the cases where `PROREL` is possible, and with the absence of a preceding *ne*, to eliminate most of the `ADV` occurrences.

Clause contains subjunctive verb: is the *que* followed by an unambiguously subjunctive verb? We remove cases where there is no preceding verb, and where the word *que* directly follows an interrogative pronoun or determinant (e.g. *Quelles que soient les...*). For the remaining cases, we expect most of the positive responses to be subordinating conjunctions.

- Followed by subjunctive (total 48)

- Pos-tag `ADV` (1)
  - ∗ Sample: *L'effondrement de cette place n'est survenu **qu'**/ADV en février 1991, et les **émissions** japonaises n'ont repris qu'à un rythme modéré ces derniers mois.* Note here the danger of look-ahead features: in this case we mistake the noun *émissions* for the past subjunctive 1st person plural of the verb *émettre*! Since tokens to the right have not yet been assigned a pos-tag, we need to rely on the glossary to guide us in identifying which words can actually be verbs.
- Pos-tag `CS` (46).
- Pos-tag `PROREL` (1)
  - ∗ Sample: *[...] veiller à ce **que**/PROREL la croissance ne se fasse pas au détriment de la qualité [...].* This should probably be considered an annotation error. Despite the superficial appearance of a relative clause (given the preceding pronoun *ce*), the *que* in expressions such as *attendre à ce que* and *veiller à ce que* should probably be considered a subordinating conjunction, since it plays no grammatical role in the subordinate clause.

***Que* directly followed by verb:** while trying to identify direct objects in the subordinate clause, we noticed that, in the case of relative clause, the verb and subject are often inverted, so that the clause starts with a verb, and the subject is in the position typically taken by the direct object. We thus expect the majority of cases where the clause starts with a verb to be relative clauses. Here again, since in a lookahead feature we rely on the lexicon alone, we had to exclude words with several grammatical categories, such as *cela* (demonstrative pronoun and simple past of the verb *celer*) and *entre* (preposition and 1st/3rd person singular present of the verb *entrer*).

- Directly followed by verb (total 114)
  - Pos-tag `ADV` (2): both feature mistakes for ambiguous words, *louanges* (2nd person singular present of the verb *louanger*) and *hausse* (1st/3rd person singular present of the verb *hausser*).
  - Pos-tag `CS` (14). Mostly intransitive verbs with subject inversion.
    - ∗ Sample: *[...] c'est évidemment là aussi **que**/CS* réside *une des causes des pénuries..*
  - Pos-tag `PROREL` (95)
    - ∗ Sample: *[...] l'hymne à la croissance **qu'**/PROREL* entendent *célébrer les Etats-Unis à Munich [...].*

Direct object in subordinate clause: this was one of the features clearly identified in the preliminary phase. Unfortunately, it is fairly difficult to implement, since recognising a direct object in lookahead features can rely on the lexicon only. We first exclude clauses starting with a verb (to exclude cases of subject inversion). We also leave out verbs capable of subcategorising two direct objects: *appeler*, *nommer* and *designer*. We then check if, following the verb of the clause, we have a definite article (*le*, *la*, *l'*, *les*), followed by either a word that is listed as a noun in the lexicon, or a pair of words that are listed as respectively an adjective and a noun. We exclude the case of weekdays and month names. In hindsight, we could have

extended this feature to the indefinite articles *un* and *une* as well as to cardinal numbers, but have to exclude the indefinite articles *de* and *des* so as to exclude prepositional phrases.

- Direct object in subordinate (total 24)
  - Pos-tag `CS` (24).
    - Sample: *[...] ont expliqué **que**/CS la crise qui frappe les journaux depuis la fin de 1989 [...].*

Although we have pos-tags to help us, identifying a direct object in the previous verb turned out to be even more problematic, and was eventually left out. Perhaps it could be re-introduced with further analysis and refinement.

No preceding content words: we noticed that if there are no preceding content words (verbs, nouns, etc.), the *que* is likely to be either a subordinating conjunction or an interrogative pronoun. There is also a single case of an exclamatory adverb in our corpus.

- No preceding content words (total 35)
  - Pos-tag `ADV` (1).
    - Sample: ***Qu'/ADV** il était insouciant, le mois de janvier 1992.*
  - Pos-tag `CS` (22).
    - Sample: ***Qu'/CS** on ouvre immédiatement le guichet, ordonne-t-il..*
  - Pos-tag `PROREL` (1). In this case a broken sentence.
    - Sample: ***Qu'/PROREL** elle n'aime guère voir rapprocher de celui des sociétés concurrentes.*
  - Pos-tag `PROWH` (11).
    - Sample: ***Que/PROWH** fait-il de l'excédent de ses revenus ?*

Curiously enough, all sentences starting with interrogative pronouns end with question marks: a fact we immediately seized upon and added as a feature to distinguish these from subordinating conjunctions. We also added another feature for interrogative pronouns: the presence of verb directly after *que*, which is quite unlikely in most other cases.

6.3.2.3 Results

The end results are positive, but not as impressive as those for the negative *que*, given the amount of work required to define, test, and refine these features (approximately 3 full days).

	ADV		**CS**		**PROREL**		**PROWH**	
ADV	75	(+17)	23	(-17)	1		1	
CS	12	(-8)	934	(+17)	41	(-8)	0	(-1)
PROREL	0		49	(-9)	205	(+9)	0	
PROWH	0		2	(-3)	3	(-11)	18	(+14)

Table 6.4: Confusion matrix for *que* with targeted relative pronoun features

If we ignore the considerable ancillary gains for adverbs, we see a delta of 17 correctly identified subordinating conjunctions, 9 correctly identified relative pronouns, and a much better categorisation of interrogative pronouns: 14 cases. All in all for *que* we have introduced 73 new corrections for 16 new errors, with a 30% improvement in the overall error rate. Regarding other tokens, we have introduced 46 new corrections for 67 new errors. Again nothing systematic is seen.

If we combine these features with the negative adverb features in a single model, we get the following final confusion matrix:

	ADV		**CS**		**PROREL**		**PROWH**	
ADV	91	(+33)	9	(-31)	0	(-1)	0	(-1)
CS	12	(-8)	934	(+17)	41	(-8)	0	(-1)
PROREL	0		48	(-10)	206	(+10)	0	
PROWH	0		2	(-3)	2	(-13)	20	(+16)

Table 6.5: Confusion matrix for *que* with all targeted features combined

The overall picture is much cleaner than it was initially, with gains in all four categories, and accuracy increasing from 86.08% to 91.65%, and the number of errors reducing from 190 to 114, a 40% reduction in the error rate. For a total of 1,365 occurrences in all corpora, we have fixed 92 new cases for 16 errors introduced. There is of course considerable room for improvement here, since there are 114 remaining errors, and we look at one way of improving via rules in section 6.5.2. When we look at the total picture, the accuracy for all corpora combined has increased from 96.51% to 96.59%. For the 150,776 tokens other than *que* (out of a total of 152,141), we have corrected 163 other tokens, and introduced 121 errors. We thus have a total of 255 new corrections and 137 new errors, for a delta of 118 additional true positives, 76 of which concern the word *que*. An examination of the corrections and errors outside of *que* again yields no obvious patterns—indeed, it is quite puzzling to see completely unrelated words changing categories (for better or worse) without a single instance of *que* in the sentence, but this is one of the prices to pay when using robust statistical methods.

Although the gain in accuracy for *que* is undeniable, the cost of all of these new features in terms of performance is far from negligible. There are 22 new features over and above the 66 baseline features, and yet, for a one million word corpus, pos-tagger feature analysis alone took 287 seconds (just under 5 minutes) in the baseline version, and 602 seconds (just over 10 minutes) in the targeted version. This accounts for about 1/6th of total analysis time for the full processing chain, where the baseline parser features account for another 1/3rd.

6.3.3 Effect of targeted pos-tagger features on the parser

As presented at start of the present section, our goal was to identify and correct pos-tagging errors responsible for the most parsing errors. To this end, we now look at the effect of the corrections induced by the new targeted features on the parser, when the pos-tagger and parser are chained in sequence. In other words, if we take the gold tokens as input, and then apply the new pos-tagger model to select the pos-tags, and the baseline parser model to add dependencies, what improvement in LAS/UAS do we get with respect to a setup identical in all respects except that it uses the baseline pos-tagger model?

LAS	**Beam 1**	**Beam 2**
Baseline LAS	85.34	85.95
Que features LAS	85.42	86.01
Baseline UAS	88.03	88.58
Que features UAS	88.10	88.66

Table 6.6: Parsing improvement for using a pos-tagger model with targeted *que* features

We have a total of 133,643 dependencies in all of the evaluation corpora combined. For a delta of 118 pos-tagger corrections, we have respectively 106 and 78 new labelled corrections at beams 1 and 2, and respectively 103 and 106 new correct governors. In beam 1, we have 305 corrections for 199 new errors. In beam 2, we have 372 corrections for 294 new errors. For the full set of corpora regardless of the beam width, all changes are statistically significant.

Of course, we were particularly intrigued by the large number of errors introduced. The main question is, are they in any way related to the corrections or errors for *que*, or are they related to the ancillary errors introduced for other tokens? To resolve this question, we looked into the differences in the FTBDep dev corpus, but found nothing conclusive—indeed, a large portion of differences did not occur in the vicinity of the word *que*, but rather concerned some other word whose pos-tag had changed. Most corrections for the word *que* result in 2 or more parsing corrections, and most errors for this word result in 2 or more parsing errors. Some errors for other words result in 4 or more parsing errors, but nothing systematic appeared in such a small sample.

We therefore decided to turn to the vaster unannotated corpora presented in section 4.2.3, to see if any clearer patterns appear. We reviewed the first few differences in the Frantext, Est Républicain, and Revues.org corpora. There was about one difference per 100 words. The results are very much in favor of the new features. Of 50 cases reviewed, 8 had the correct pos-tag in the baseline version, and 40 in the version with targeted features. This time, the total parsing error count is even less telling: the pos-tag differences resulted in 100 errors in the baseline version, and 44 errors in the version with targeted features.

One obvious culprit for introducing parsing errors in the unannotated corpora was the case of comparative structures. These are almost systematically mishandled by Talismane, even more so when the pos-tag is correctly annotated. According to the FTB annotation guide [Abeillé and Clément, 2006], *que* should always be annotated as a subordinating conjunction in the case of comparatives. In the FTBDep annotation guide [Candito et al., 2011b], since a comparative *que* is justified by the preceding comparative adverb, it should depend on it, generally introducing non-projectivity, since they are separated by the object being compared. However, since Talismane currently uses a strictly projective version of the transition-based parsing algorithm, and since the version of FTBDep used for training is strictly projective as well, we must find an alternative governor for the comparative *que*—logically the object being compared, which is the governor of the previous comparative adverb. This means, however, that we have a different governor pos-tag for each type of comparative. Take the following set of examples drawn from the training corpus and unannotated corpora, with the projective governor and the object of *que* marked in bold:

1. ***Plus*** *qu'un* ***gadget****, elle a été le remède miracle [...]*

2. *[...] accueilli aussi bien* ***par*** *le public que* ***par*** *la critique.*

3. *C'est peut-être pour ça que Suzanne est arrivée plus* ***tôt*** *que* ***prévu*** *!*

4. *[...] la Commission écarte autant la* ***libéralisation*** *complète du secteur postal que son* ***harmonisation*** *totale.*

5. *[...] les hommes sont plus* ***frappés*** *par cette crise que les* ***femmes****.*

There are several difficulties with these structures from Talismane's perspective. The first is the wide variety of pos-tags that can govern the *que*: in the above cases respectively `ADV`, `P`, `ADJ`, `NC` and `VPP`. One of the primary clues currently used by Talismane to select the correct governor is thus eliminated. Secondly, subordinating conjunctions typically govern a verb. However, in the case of comparatives, the subordinate clause, while it is generally possible to reconstruct one, is highly elliptic, and hardly ever contains a verb. In the above cases, the dependent pos-tags are respectively `NC`, `P`, `VPP`, `NC` and `NC`. Therefore, Talismane needs to look for other clues in order to select the correct governor and dependent—in the case of the governor, the fact that it is immediately preceded by a comparative word, and, in the case of the dependent, an analysis of the structure following the word *que*. The current feature set is not well adapted to such clues but, even if it were, both the structural variety and fairly reduced number of camparatives in the training corpus (approximately 230), especially when compared to the large number of similarly annotated subordinating conjunctions (approximately 1150), means it may be difficult to generalise.

6.4 Using targeted parser features

We now turn to targeted features in the parser. Again, we wish to concentrate on those areas where there is the most room for improvement. An obvious candidate is coordination, which involves two dependencies, `coord` tying the coordinator to the previous conjunct and `dep_coord` tying the next conjunct to the coordinator. In the FTBDep development corpus with baseline model, `coord` has a precision of 99.59% (it is very unlikely to confuse `coord` with anything else, given the strong indicator of a coordinating conjunction or comma), a recall of 58.53% (given the difficulty of selecting the correct first conjunct), and an f-score of 73.73%. As for `dep_coord`, we have a precision of 99.49% (again, because of the same strong indicators), a recall of 82.21% (the 2nd conjunct is generally easier to identify than the first, since it closely follows the conjunction), and an f-score of 90.62%. Since the case of coordinating commas accounts for only 22 errors out of a total of 345, we decided to concentrate on coordinating conjunctions.

6.4.1 Parser coordination features

As presented in section 1.1.3.2, page 30, it is generally impossible to correctly identify the first conjunct without looking ahead beyond the coordinating conjunction, as illustrated by the three sentences taken from that section:

1. J'ai mangé une pomme rouge et mûre.

2. J'ai mangé une pomme rouge et une orange.

3. J'ai mangé une pomme rouge et Georges a bu du thé.

This implies that in order to correctly select the first conjunct we need to make a fairly reliable guess at the identity of the second conjunct.

Furthermore, in the shift-reduce algorithm used by transition-based parsing, the decision as to which two elements are coordinated is split into multiple individual decisions, as illustrated by table 6.7. In this table, we show the transition sequence during which the coordination dependencies are added. The most difficult decisions are the `reduce` at n+1, where we have to decide whether or not the 2nd conjunct coordinates with *rouge*, and the `reduce` at n+2, where we have to decide whether or not the 2nd conjunct coordinates with *pomme*. The `right-arc`$_{\text{coord}}$ decision at n+3 is trivial, since there is nothing else left to coordinate with *et* at this point. Also, having recognised the 1st conjunct as *mangé*, it is fairly straightforward to know that the 2nd conjunct should be a verb as well. Still, the `shift` decision at n+4 needs to look at this 1st conjunct in order to make its decision, as well as ensuring that the coordination is not elliptical (e.g. *"J'ai mangé une pomme et Georges une banane."*). The `shift` decision at n+5 has a bit of logic behind it as well, as it needs to recognise that we have a composite past tense, rather than present tense verb *avoir*. At this point the two `left-arc` decisions at positions n+6 and n+7 are natural consequences of the previous two `shift` decisions, and the final `right-arc`$_{\text{dep_coord}}$ is trivial as well, since the verb must necessarily be the 2nd conjunct.

	transition	**stack**	**buffer**	**dependencies added**
n		*root*, mangé, pomme, rouge	et, Georges, a, bu, du, thé	
n+1	**reduce**	*root*, mangé, pomme	et, Georges, a, bu, du, thé	
n+2	**reduce**	*root*, mangé	et, Georges, a, bu, du, thé	
n+3	right-arc$_{\text{coord}}$	*root*, mangé, et	Georges, a, bu, du, thé	coord(mangé,et)
n+4	**shift**	*root*, mangé, et, Georges	a, bu, du, thé	
n+5	**shift**	*root*, mangé, et, Georges, a	bu, du, thé	
n+6	left-arc$_{\text{aux_tps}}$	*root*, mangé, et, Georges	bu, du, thé	aux_tps(bu, a)
n+7	left-arc$_{\text{suj}}$	*root*, mangé, et	bu, du, thé	suj(bu, Georges)
n+8	right-arc$_{\text{dep_coord}}$	*root*, mangé, et, bu	du, thé	dep_coord(et, bu)

Table 6.7: Arc-eager transition sequence for coordination, with difficult decisions in bold

We have thus recognised four difficult decisions in the present case. If we examine more closely the information that can help us make the correct decision for the first two `reduce` decisions, it is (A) correctly guessing the 2nd conjunct ahead of time and (B) examining the elements deeper in the stack, to see which is most likely to coordinate with this 2nd conjunct. Further downstream, when we reach the `shift` decisions, we already know which 1st conjunct

was selected, and so we now have to select the most likely 2nd conjunct by reviewing the items on the buffer as far as the content verb *bu*.

However, correctly guessing the 2nd conjunct is not always as simple a task as in toy examples like these. If the conjunction is directly followed by a verb, a preposition or a relative pronoun, there is no ambiguity, with the 2nd conjunct being respectively the content verb, the preposition itself, and the content verb of the relative clause. If, however, the conjunction is followed immediately by a noun and a verb later in the phrase, we can only decide by analysing the sentence as a whole. Take the following examples of 2nd conjunct ambiguity, taken from the FTBDep development corpus, and showing the correct annotation, with the two verbs in italics, the conjuncts underlined and the conjunction in bold.

1. Il s'*agit* ici d'un jour normal de la semaine **et** un inventaire scrupuleux *exigerait* que l'on prenne également en compte l'offre accrue du mercredi.
2. Les chiffres parlent d'eux-mêmes : les Japonais *occupent* 30 % du marché américain **et** leurs exportations *représentent* près de 75 % du déficit commercial global annuel.
3. A Lourdes, nous *signale* notre correspondant Jean-Jacques Rollat, la venue **et** la circulation des pèlerins *ont* été très perturbées.
4. Le taux de salaire horaire ouvrier *a* progressé de 0,7 % au cours du troisième trimestre 1991, soit une croissance de 4,5 % en un an **et** un gain de pouvoir d'achat de 2 % sur la même période, *indique* le ministère du travail.
5. Les émissions d'éveil qui *ont* fait la richesse des chaînes de service public entre 1975 **et** 1985 *ont* toutes disparu.

We identified the following rule: if the sentence contains a conjugated verb prior to the conjunction, then the verb following the conjunction is the 2nd conjunct, unless it follows a relative pronoun or subordinating conjunction. However, there are several exceptions that have to be excluded from this rule: if the preceding or following verb is in a reporting clause or comment clause (respectively *signale* in example 3 and *indique* in example 4), and if the preceding verb is itself in a relative clause (as in example 5). With these exceptions, the rule becomes almost necessary and sufficient: if these conditions are *not* met, the 2nd conjunct cannot be the verb following the coordinating conjunction (although relative clause and comment clause limits have to be carefully delimited).

Once the verb has been eliminated, there is little remaining ambiguity possible on the 2nd conjunct: it must be the noun if there is one, otherwise an adjective if there is one, otherwise an adverb. The remaining exception is the case of coordinated determiners (e.g. *"un ou plusieurs"*), which we have not yet attempted to handle in the present study.

6.4.1.1 Development corpus error analysis

Having thus identified the 2nd conjunct, we now turn to the methodology already presented in section 6.2 for selecting useful features. The first step is reviewing the errors in the development corpus, in order to imagine features that are likely to be useful. In the examples below, the reference annotation is in bold, and Talismane's baseline guesses are underlined.

Example 6.10 *Toutefois, les tarifs réduits* ***progresseront*** *moins que l'*inflation *et même* ***diminueront****, pour les jeunes [...]*

In this example, it is clear that the 2nd conjunct is a verb, since it follows the coordinating conjunction without an intervening noun. The baseline model incorrectly selected the nearest noun as the 1st conjunct. An obvious feature is one where we attempt to find a preceding verb to coordinate with, if the 2nd conjunct is a verb as well.

Example 6.11 *L'arrêté est **sorti** le 7* janvier *et la* circulaire ***est** prête, indique le ministère du travail.*

In this example, the baseline model failed to recognise that the 2nd conjunct is a verb, and therefore attempted to coordinate two nouns. Thus, features are required to recognise when the 2nd conjunct is likely to be a verb, using the rules described earlier in the present section.

Example 6.12 *Dix millions de francs sont **prévus** pour les* employés *de particuliers et 190 **millions** pour les salariés des associations.*

In this case, we have an elliptical construct, which is quite common with coordination. The annotators have chosen to coordinate the main verb as the first conjunct with the subject of the second conjunct, since the verb is omitted in the second construct. However, due to the complexity and rarity of these structures, we will not attempt to identify them for now via features.

Example 6.13 *Hafnia, pour sa part, a perdu 800 millions de couronnes (696 millions de francs) en partie* à cause de *sa participation de 33,7 % **dans** Baltica et **dans** le groupe suédois d'assurances Skandia, numéro un en Scandinavie.*

In this example, we see that we should encourage coordination with an identical preposition rather than a different one, if the choice is offered.

Two additional errors concern non-parallel coordination between clauses, where the governor and dependent are not easy to identify:

Example 6.14 *Les deux firmes* ont *la même caractéristique : elles **ont** depuis peu dans leur tour de table, mais **à** des degrés différents, l'italien Agnelli.*

Example 6.15 *D'où son* idée *depuis de **calmer** le jeu sans perdre la face, et ce* tour de passe-passe *serait **destiné** à y contribuer.*

In the first case, the reference annotation seems correct: the preposition should be attached to the nearest verb. In the second case, there is no clear first conjunct, but a better option would probably be to coordinate the prepositional phrase as a whole, with *d'où* as the first conjunct and *destiné* as the second.

There are also a very large number of annotation errors in the development corpus: 13 among the first 32 errors, or 41%. Among these, 12 concern multiple conjuncts, where the FTBDep corpus incorrectly coordinates all of the conjuncts with the first conjunct, whereas Talismane correctly coordinates each conjunct with the previous conjunct (as per the annotation convention set out in chapter 1). The example below shows the type of error:

Example 6.16 *L'application est large puisqu'elle couvre, aussi bien, l'embauche d'une femme de ménage que la garde d'enfants en passant par l'**aide** aux personnes âgées, les* travaux *de jardinage et le **soutien** scolaire, par exemple.*

If this tendency is confirmed on the rest of the corpus, the initial accuracy needs to be taken with a pinch of salt.

All in all, among the first 32 errors, we have:

- 12 annotation errors with multiple conjuncts, where Talismane annotated correctly
- 1 additional annotation error, incorrectly annotated both in the reference and by Talismane
- 8 cases where Talismane incorrectly coordinated two different parts of speech (typically noun/verb, verb/noun or noun/preposition)
- 4 cases where Talismane didn't find the correct 2nd conjunct, and coordinated two nouns instead of two verbs. One of these cases concerns an adverbial comment starting with *selon* immediately after the conjunction, which is mistaken for the 2nd conjunct.
- 7 cases that are less easy to classify: preposition mismatch, parallelism not respected (coordination of X and B in *"A pour X et B pour Y"*), idiomatic structures not respected (*"entre A et B"*), coordination with the helper verb instead of the content verb, ellipsis, and non-parallel clause coordination.

6.4.1.2 Feature list

Having analysed the types of errors produced, we next translated these features into logical constructs readable by Talismane, and refined them against the training corpus. We show below the list of features obtained after refinement. These features use the same terminology for the stack and buffer as the rest of the thesis, with σ_0 representing the top of stack, σ_1 the previous item on the stack, etc., and β_0 representing the head of buffer, β_1 the next item on the buffer, etc. When giving samples, we show both σ_0 and β_0 in bold, with σ_0 by definition to the left of β_0. The two conjuncts are underlined. The vast majority of features concentrate on identifying the relationship between the 1st conjunct and the coordinating conjunction, as this is generally far more difficult than identifying the 2nd conjunct. When the coordinating conjunction is immediately followed by a comma, we look for the 2nd conjunct after the following comma on the assumption that we have an intervening adverbial.

Coordination between verbs: if the 2nd conjunct is a verb, is the 1st conjunct under consideration a verb as well? Translated to the transition-based parsing paradigm, if β_0 is a coordinating conjunction and the 2nd conjunct is determined to be a verb, is σ_0 a conjugated verb as well? Conjugated verbs are considered to be any verb that is indicative, subjunctive or imperative, as well as past participles if they govern a passive or tense helper verb, and infinitives if they govern a causative helper verb. If the result is `true` (σ_0 is a verb as well), we're expecting a transition of `right-arc`$_{\text{coord}}$ most of the time, and if the answer is false (σ_0 is not a verb) we're expecting `reduce` most of the time.

- σ_0 is a verb as well (total 635)
    - Transition `right-arc`$_{\text{coord}}$ (583)

    * Sample: *Le syndicat IG-Metall avait immédiatement* ***protesté*** *contre ce projet* ***et*** *il a déclenché la grève.*

  - Transition `reduce` (48)
    * Generally a coordination with three or more verbal conjuncts, where the annotation incorrectly coordinates the 3rd conjunct with the 1st one instead of the 2nd. There are also a number of cases where the first verb is in a relative clause—these ought to be removed from this feature by further refinement.

- σ_0 is not a verb (total 1251)
  - Transition `reduce` (1172)
    * Sample: *La priorité est à la* ***reconstruction*** *de l'Est* ***et*** *la plupart des ministères voient leur budget sérieusement amputé.*
  - Transition `right-arc`$_{\text{coord}}$ (28)
    * Typically annotation mistakes or cases where our feature mistakenly identified a comment clause or relative clause.

Mismatched prepositions: if the 1st and 2nd conjuncts under consideration are two different prepositions, does the same preposition exist as a candidate earlier in the sentence? Translated into the transition-based parsing paradigm, if β_0 is a coordinating conjunction immediately followed by a preposition as 2nd conjunct, and σ_0 is a preposition as well, but a different presposition from the 2nd conjunct, does the stack contain a preposition identical to the 2nd conjunct? When comparing prepositions, we allow for cases of compound prepositions which are later abbreviated, as in *grace à X et à Y*. We only consider the case when the answer is yes. If this feature indicates a preposition mismatch with a better preposition on the stack, we expect a preponderance of `reduce`.

- Mismatched prepositions = `true` (total 135)
  - Transition `reduce` (132)
    * Sample: *Les gouvernements seront cependant prudents dans tous les domaines qui touchent au niveau de vie* ***de*** *la population* ***et*** *à l'emploi.*
  - Transition `right-arc`$_{\text{coord}}$ (2)
    * Sample: *D'origine chinoise, il travaille dans un grand cabinet juridique français installé à la fois* ***à*** *Pékin* ***et*** *dans la colonie britannique [...]*

Avoid noun-preposition coordination: if the 1st conjunct under consideration is a noun, and the 2nd is a preposition, is there a prepositional candidate earlier in the sentence? Translated into the transition-based parsing paradigm, if β_0 is a coordinating conjunction immediately followed by a preposition as 2nd conjunct, and σ_0 is a noun, does the stack contain a preposition? We only consider the case when the answer is yes. In this case, we expect a preponderance of `reduce`.

- Noun-preposition coordination proposal with preposition on stack (total 2134)
  - Transition `reduce` (1942)

    * Sample: *L'exemple des **banques et** des assurances pourrait le démontrer.*
- Transition `right-arc`$_{\text{coord}}$ (157)
    * Amazingly enough for such a large number of cases, the first 30 cases on the list were systematically annotation mistakes.

The very large number of annotation mistakes identified by this feature suggests another possible use for feature projection and refinement, as a means of identifying and correcting training corpus errors. However, the methodology and effect of such fixes has not been explored in the present thesis.

Pos-tag mismatch: if the 1st conjunct under consideration has a different pos-tag from the 2nd conjunct, does a candidate with the same pos-tag exist earlier in the sentence? Translated into the transition-based parsing paradigm, if β_0 is a coordinating conjunction, σ_0 has a different pos-tag from the guessed 2nd conjunct, does the stack contain an item with the same pos-tag? We only consider the case when the answer is yes, in which case we expect a preponderance of `reduce`. We test this feature in two versions: one where we only look at the item at σ_1, and one where we search the entire stack for the identical pos-tag.

- Pos-tag mismatch with match at σ_1 (total 3105)
    - Transition `reduce` (2860)
        * Sample: *Nous n'avons pas de réserves de gaz, pas de **réserves** de pétrole **et** l'exploitation du charbon diminue.*
    - Transition `right-arc`$_{\text{coord}}$ (163)
        * There are a huge number of annotation mistakes, including all of the ones from the previous feature. One correct sample is: *Il y a, bien sûr, les films qui défendent directement cette cause, **ceux** des catégories "causes charitables" [...], **ou encore** l'impressionnant spot de Greenpeace sur la protection de l'antarctique.* In this case, we see a mismatch between `PRO` and `NC`—although such cases should be excluded from mismatches by further feature refinement, grouping together all pos-tags that act as nominal expressions.
- Pos-tag mismatch with match somewhere on stack (total 4763)
    - Transition `reduce` (4255)
        * Sample: *[...] ses adversaires politiques ne proposent aucune solution **alternative et** considèrent, avec un bel ensemble, que la défense des intérêts des malades [...]*
    - Transition `right-arc`$_{\text{coord}}$ (382)
        * In addition to the annotation mistakes from previous features, there are several valid cases of `NPP`/`NC` agreement, and `PRO`/`NC` agreement which should be excluded by feature refinement.

Pos-tag match: if the 1st conjunct under consideration is the same pos-tag as the 2nd conjunct, are there any other candidates with this pos-tag earlier in the sentence? Translated into the transition-based parsing paradigm, if β_0 is a coordinating conjunction, and σ_0 has the same pos-tag as the guessed 2nd conjunct, does the stack contain only items with a

different pos-tag? We only consider the case when the answer is yes, in which case we expect a preponderance of `right-arc`$_{\text{coord}}$.

- Pos-tag match with no other match on stack (total 2424)
  - Transition `right-arc`$_{\text{coord}}$ (2080)
    - Sample: *Aussi devrait-on revenir à un certain équilibre* ***en termes d'****offre* ***et*** *de demande.* Note that in this case, the preposition *à* has already been correctly removed from the stack by a previous `reduce` transition. This suggests that we might also gain some ground by creating features that tackle coordination earlier in the process, when the coordinating conjunction is farther down the buffer than position β_0, in order to avoid or encourage a `reduce` transition for a potential 1st conjunct.
  - Transition `reduce` (319)
    - Again, a very large number of annotation mistakes. No correctly annotated cases were identified in the first 30.

2nd conjunct is content verb: this feature, unlike most of the others, concerns the `dep_coord` relationship. We attempt to ensure that when there is a verbal kernel consisting of multiple verbs, we coordinate with the content verb governing the structure rather than the helper verbs. If the dependency label govering σ_0 is `coord`, and the governor of σ_0 is a verb, is β_0 a helper verb? If so, we're expecting the `shift` transition.

- β_0 is helper verb (total 45)
  - Transition `right-arc`$_{\text{coord}}$ (1)
    - This case is an error in the feature definition when identifying helper verbs in the verbal phrase: *[...]* ***et*** *les ignorer* ***seraient*** *alimenter abusivement le pessimisme ambiant [...]*
  - Transition `shift` (42)
    - Sample: *La plupart sont très jeunes* ***et ont*** *trouvé là leur premier emploi.*

6.4.1.3 Results

The results of these features across all evaluation corpora (dev+test) on the label `coord` are as follows:

	beam 1		**beam 2**	
`coord`	**baseline**	**targeted**	**baseline**	**targeted**
true+	2230	2262	2287	2320
false+	25	27	27	31
false-	1687	1655	1630	1597
precision	98.89%	98.82%	98.83%	98.68%
recall	56.93%	57.75%	58.39%	59.23%
f-score	72.26%	72.90%	73.41%	74.03%

Table 6.8: Parsing improvement for the relation `coord` using targeted features

The results of these features on the label `dep_coord` are as follows:

	beam 1		**beam 2**	
`dep_coord`	**baseline**	**targeted**	**baseline**	**targeted**
true+	3474	3479	3510	3523
false+	36	36	36	37
false-	765	760	729	716
precision	98.97%	98.98%	98.98%	98.96%
recall	81.95%	82.07%	82.80%	83.11%
f-score	89.66%	89.73%	90.17%	90.34%

Table 6.9: Parsing improvement for the relation `dep_coord` using targeted features

In terms of significance, we have the following contingency matrix the label `coord` in beam 1 (p-value < 0.005):

`coord` beam 1	**targeted true**	**targeted false**	**baseline totals**
baseline true	2184	46	2230
baseline false	78	1609	1687
targeted totals	2262	1655	3917

Table 6.10: Contingency table for `coord` with and without targeted features, at beam 1

We have the following contingency matrix the label `coord` in beam 2 (p-value < 0.01):

`coord` beam 2	**targeted true**	**targeted false**	**baseline totals**
baseline true	2234	53	2287
baseline false	86	1544	1630
targeted totals	2320	1597	3917

Table 6.11: Contingency table for `coord` with and without targeted features, at beam 2

Thus, there are a total of 78 new correct answers for 46 new errors in beam 1, and 86 new correct answers for 53 new errors in beam 2. The error rate for `coord` at beam 1 has only reduced by 1.9%, from 1687 to 1655. These results are somewhat disappointing in terms of the amount of effort required, and especially in view of the very large proportion of errors which remain untouched in both beams. When we review the errors remaining in beam 1 and compare them to the first 32 errors analysed earlier in this section, we see that only 4 errors have been fixed: only 1 out of 8 pos-tag mismatches, 2 cases where the 2nd conjunct was not correctly identified as a verb, and 1 complex coordination between clauses. The 12 manual annotation mistakes are still corrected. For these 32 errors, we have thus passed from an accuracy of 37.5% to 50.0%. The small proportion of pos-tag mismatches corrected either indicates that the pos-tag mismatch feature needs to be refined further, or that it is drowned out by other features. To test this, we would first need to test the features on these 8 errors to ensure they give the expected answers and, if so, see if anything can be done to widen the scope of the features or refine them further to remove cases in the training corpus that do not give the expected transition.

We also reviewed the first 40 of the errors introduced by the targeted feature model. We found that targeted feature model tends sometimes to privilege, as a 1st conjunct, the governor of a prepositional phrase rather than its dependent, and, more surprisingly, made several errors where the 1st conjunct was marked as a verb and the 2nd as a noun. The latter errors may be due to a mistake when guessing the 2nd conjunct during the 1st conjunct annotation, which is not repeated when the 2nd conjunct is actually annotated. This implies that the 2nd conjunct guessing heuristic is not sufficiently watertight. Finally, we have 3 new fixes for reference annotation mistakes.

We turn once again to the unannotated corpora to see if any clear tendencies are highlighted. In the unannotated corpora, we found about 1 difference for every 200 words. We checked the first 10 or so differences in each corpus: for a total of 55 differences, we noted 14 where the baseline was correct, 30 where the targeted features were correct, and 6 where neither was correct. Most of the corrections concern mismatched pos-tags in the baseline corpus, as shown in the example below, with the original annotation in bold and the corrected annotation underlined:

Example 6.17 *Les truquages ne* ***sont*** *pas ici pour leur illusion féerique, mais pour leur* <u>réalisation</u> *mécanique ou leur* <u>***matière***</u> *optique.*

6.5 Using rules

We now turn from features to rules. As discussed in section 3.5, page 95, rules are used to impose or prohibit certain local decisions during the analysis process, thus bypassing the statistical model. At each step of the analysis process, positive rules are applied first: if any of them return `true`, a certain category is imposed, and the statistical model is not consulted. Negative rules are applied after the statistical model has returned its probability distribution for the various categories. If any rule returns `true`, the associated category is removed from the distribution.

The methodology we typically use for rules is similar to that used for targeted features, and can be summarised as follows:

1. Analyse errors in Talismane's output—either automatically identified in reference evaluation corpora, or detected in unannotated corpora after manual review.

2. Identify errors that can be described with high precision, and for which the associated rule is likely to be applicable systematically.

3. Convert these rules into boolean features using Talismane's feature definition syntax, and project them onto the training corpus, ensuring that the results always correspond to expectations. Refine if necessary.

4. Analyse the evaluation corpora with the new rules applied, and ensure they function correctly. If necessary, return to step 3.

5. Analyse unannotated corpora with the rules applied, and compare the analysis results to the baseline analysis results, to ensure they function correctly. If necessary, return to step 3.

6.5.1 Pos-tagger closed vs. open classes

Recall from section 4.5.2, page 125, that the default French implementation comes with a set of baseline rules, which only allow us to assign a closed class pos-tag to a token if it is listed in the lexicon as belonging to this closed class. This prevents us from inventing new closed-class lexical forms. Similarly, we have a rule that does not allow us to assign an open class pos-tag to a token if it is only listed with closed classes in the lexicon. This prevents us from using function words as content words.

While this assumption sounds reasonable, the actual usefulness on real text has yet to be proven. To this end, we tested the pos-tagger without these rules in place. The results are shown below for all evaluation corpora:

Corpus	**baseline**	**closed class rules**
FTBDep dev	96.90%	96.98%
FTBDep test	97.27%	97.39%
EstRépu	96.63%	96.89%
Europarl	96.76%	97.12%
EMEA dev	93.03%	93.78%
EMEA test	95.59%	95.82%
FrWiki	95.62%	95.80%
FrWikiDisc	94.99%	95.26%
Total	96.30%	96.51%

Table 6.12: Pos-tagging accuracy improvement using closed class rules

There is a systematic improvement through the application of the rule, even for unedited corpora such as FrWikiDisc where we would expect spelling mistakes and typos to render such rules dangerous. In total, we have 445 new corrections for 127 new errors, with a total error reduction rate of 5.7%, and a McNemar p-value < 0.001.

An examination of the first 50 new errors is telling as well: many of them concern numbers in a non-standard French format (decimal marker using a period instead of a comma), and others result from words that have not been correctly lower-cased, since they don't appear at sentence start or after punctuation (e.g. "*Selon*" as a preposition with a capital S). All of the errors examined are valid, and most can be corrected by improving the lexicon or the token rules for recognising numbers or lower-casing.

An examination of the new corrections is telling of the limits of unconstrained statistical methods: in the unconstrained baseline version there are many cases of rare classes being replaced by more common classes (e.g. `DETWH` being replaced by `ADJ` in the case of "*quel*"). There are a considerable number of corrections for what would appear to be completely absurd pos-tags: `PONCT` for proper nouns, or `CLO` for verbs.

6.5.2 Pos-tagger rules for *que*

In section 6.3, we attempted to improve performance for the ambiguous word *que* through the use of features. In this section, we attempt to further improve the score using highly specific rules.

As was seen in the section on features, certain cases were not corrected even though the feature returned the correct result: the feature seemed to be drowned in the ocean of other more generic features, thus preventing it from tipping the scales decisively in the favor of the correct pos-tag.

We reviewed the remaining errors in the development corpus, and came up with the following rules, to be added to the existing features:

- Assign `PROWH` if the sentence ends with a question mark, there is no content word prior to *que*, and que is followed by an indicative or infinitive verb, or the clitics *en* or *se* followed by an indicative or infinitive verb. Sample: *Et que/PROWH se passera-t-il si un seul syndicat signe un accord de ce type contre l'avis des autres ?*
- Assign `CS` to *que* in any variant of *attendre à ce que*, *veiller à ce que*, *n'empêche que*, *avoir honte à ce que*, *le/du/au fait que*, *une fois que*.
- Assign `CS` for any expression of the type *être ADJ que*, such as "*il est probable que*", unless the expression is preceded by a *ne*.
- Assign `PROREL` in *quoi/qui/quel/quelle que ce soit* and *ceux/celui/celle/celles que*.
- Assign `ADV` for any verbal expression where *ne* directly precedes the verb, *que* directly follows the verb, and there is no instance of *rien, personne, aucun, aucune, ni* or *jamais* earlier in the sentence.
- Do not assign `PROREL` if *que* is the first word in the sentence.
- Do not assign `PROREL` if *que* follows a verb either directly, or after a comment clause surrounded by commas or dashes.
- Do not assign `PROREL` if *que* follows the word *reprises*, as in *"Il a dit a plusieurs reprises que. . . "* - this rule could of course be extended to other words and/or expressions, and replace the similar targeted feature.
- Do not assign `ADV` unless there is a preceding *ne*, or *que* is the first word in the sentence.

Unsurprisingly, the results of the rules on the training corpus are all positive, and we gain another 17 corrections for 0 new errors when compared to the targeted features only. When comparing our targeted features + rules version to the baseline version, accuracy for *que* has increased from 86.08% to 92.89%, and we have 49% decrease in the number of remaining errors, down from 190 to 97. The final confusion table for *que* looks like this, with numbers in parentheses giving the additional gains with respect to the version with targeted features only.

	ADV	**CS**	**PROREL**	**PROWH**
ADV	93 (+2)	7 (-2)	0	0
CS	11 (-1)	945 (+11)	31 (-10)	0
PROREL	0	46 (-2)	208 (+2)	0
PROWH	0	0 (-2)	2	22 (+2)

Table 6.13: Confusion matrix for *que* with targeted features and rules

We tested the rules on unannotated corpora as well, comparing the version analysed by targeted features for *que* only, with a version including both targeted features and rules, to ensure that their effect was positive on a much wider span of text. We get about 1 change per 10,000 words. For the first 47 changes, 40 are corrections and 7 are errors. Of the errors, 3 could be corrected by further rule refinement, 3 result from incorrect pos-tagging of the surroundings (nouns pos-tagged as verbs), and 1 results from less formal text in a literary corpus, in which a negative *que* is used without a preceding *ne*:

Example 6.18 *J'en ai connu que des mâles bighorns...*

6.5.3 Parser rules: prohibiting duplicate subjects

We now turn to parser rules, as presented in section 3.5, page 95. These are similar to pos-tag rules, except that they allow us to impose or prohibit a certain transition, rather than a certain pos-tag, in a certain context. They are, however, much more difficult to code than pos-tag rules, since one must place oneself within the transition based parsing algorithm to imagine the state of the stack and buffer in a given context, and to select the correct transitions.

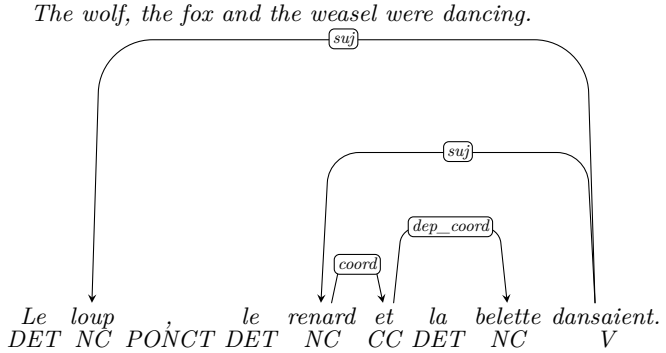
The first rule we tackle is one which attempts to prohibit the creation of multiple subjects for the same verb. It was coded in response to certain team members noticing duplicate subjects, which is not allowable in French outside of certain very specific cases. It was initially formalised as follows:

No duplicate subjects: Do not allow the creation of more than one `suj` dependency per governor, except in the case of one clitic and one non-clitic. The last exception allows for the case of clitic inversion in questions: *"Jean a-t-il mangé la pomme ?"*

This initial coding of the duplicate subject rule led to some surprises. The majority of dual subject errors result from an unrecognised coordination which uses a comma instead of a coordinating conjunction. This is most often the case when there are three or more conjuncts. The reason behind this is as follows: recall that left dependents are added from the inside out. When Talismane encounters such structures, it will reach the first conjunct, leave it on the stack, and then tackle the second conjunct. At this point, it generally does not recognise it as a conjunct despite the intervening comma, and therefore does not create the dependency. Talismane then reaches the coordinating conjunction and the 3rd conjunct, both of which are coordinated correctly to the 2nd conjunct and removed from the stack. We now find ourselves in a position with the 2nd conjunct at σ_0, and the verb at β_0. Naturally, Talismane adds this conjunct as a subject and removes it from the stack. When it next reaches the 1st conjunct, it is too late to recognise the coordination: so Talismane adds it as another subject. With the new rule in place, Talismane can no longer do so (there already is a subject), and so it typically adds it as a modifier of the verb instead.

Therefore, without the duplicate subject rule in place, we get the following result:

Example 6.19 *Le loup, le renard et la belette dansaient.*
The wolf, the fox and the weasel were dancing.



With the rule, we get the identical result, except that the `suj` of the first conjunct (*loup*) is replaced by a `mod`. Since in both cases, Talismane fails to recognise the coordination between the first two conjuncts, the analysis with two subjects seems preferable, since at least we recognise *loup* as a subject of *dansaient*.

However, the rule also corrects several cases of incorrect subjects, especially when Talismane erroneously marked the antecedent as the subject of a relative clause as in the following example (the governor for *France* is shown in bold without the rule, and underlined with the rule):

Example 6.20 *La* ***France****, qui a* ***développé*** *dans le cadre d'une politique de défense une industrie structurée et performante, n'*<u>échappe</u> *pas à cette tendance lourde...*

In this case, Talismane originally assigned *France* as the subject of *développé*. With the rule in place, it correctly finds it as the subject of *échappe*. The word *qui* remains the subject of *développé* in both cases.

Another common correction is one where a time adjunct preceding the verb is erroneously marked as a subject, since these often appear outside of a prepositional phrase:

Example 6.21 *Cet* ***après-midi****, au niveau 1 du centre commercial Les Nations, la MJC* ***accueillera*** *les enfants.*

In this case, Talismane originally marked both *après-midi* and *MJC* as subjects for *accueillera*. With the rule, *après-midi* was corrected to an adjunct, and *MJC* remained the subject.

In the case of the Europarl corpus, there are multiple corrections for the vocative noun phrases *Monsieur le Président* and *Madame la Présidente* surrounded by commas, as in the following examples:

Example 6.22 ***Madame*** *la Présidente, cette résolution me* ***pose*** *un problème majeur.*

Example 6.23 *Et comme vous le savez,* ***Madame*** *la Présidente, chers collègues, l'Union européene en est* ***arrivée*** *là...*

In both these cases, the rule corrected *Madame* to become an adjunct of the main verb, instead of its subject. The correct subject (respectively *résolution* and *Union*) remained annotated as such.

We therefore rewrote the rule to remove the majority of errors: the rule does not apply if the existing subject is a governor of a coordination. This excludes all of the cases where the subject consists of three coordinated conjuncts. However, even with this new modification in place, the rule adds 42 new corrections for 48 new errors. The vast majority of errors concerns coordination marked only by a comma, or in some cases clarification/specification, where an item is first presented and then clarified between commas, without any other marker, as in the following example (both annotated subjects marked in bold):

Example 6.24 *De 24 dollars en moyenne en 1990, le* ***prix*** *du baril de Brent, le* ***brut*** *de référence en mer du Nord, est revenu à 20 dollars en 1991...*

In many cases, the parser, unable to attach a noun as a subject, performed a `shift` transition instead, bringing the verb to the top of the stack. Note that in the arc-eager transition system, a `shift` transition is only ever needed when the governor of β_0 is further to the right. This is necessary if we want to attach the noun to another verb further down the buffer, but if there is no such verb, it prevents us from attaching the noun to anything at all, and, with the noun stuck between the central verb and the root, prevents us from attaching the central verb to the root. Thus, the single error in subject attachment provokes multiple errors downstream.

We also turned our attention to unannotated corpora, to see if the net effect of the parser rules on these corpora is positive. The results here were mixed, with one change every 250 words, and, for the first 57 differences, 22 induced corrections and 27 induced errors. Indeed, in the Frantext literary corpus, we often find numerous coordinated subjects without a conjunction, as in the following example:

Example 6.25 *La* ***couleur*** *des érables, la* ***plainte*** *de l'orignal, le* ***vol*** *des oiseaux dans le ciel, tout* ***cela*** *entrait en moi, devenait moi, et moi tout cela.*

When the duplicate subject rule worked correctly, most of the time it concerned time adjuncts not preceded by a preposition, including noun phrases governed by a month name, a weekday name, or other time-related nouns such as *semaine*, *veille*, *soir*, etc.

In our final version of the rule, we thus broke it up into three distinct, highly specific cases:

- **Time expression subjects:** do not allow the nominal head of a time expression to be a subject of a later verb if the two are separated by a comma or a dash. Time expression nouns here include any noun that typically modifies a verb as an adjunct outside of a prepositional phrase: time quantities such as *semaine*, month names, weekday names, times of day (e.g. *après-midi*), and relative day referrers (e.g. *veille*, *lendemain*). When projected onto the training corpus, this rule excluded one genuine subject, but also a whole slew of possible errors.

- **Antecedent subjects:** do not allow a noun to be the subject of a verb if it already has a relative pronoun subject.

- **Monsieur le Président:** do not allow *Monsieur* or *Madame* to be the subject of a verb if it is in the expression *Monsieur le Président* or *Madame la Présidente* followed by a comma or dash. This is an example of a targeted corpus-specific rule for the Europarl corpus.

With these three rules in place, we get much cleaner results on the evaluation corpora. We now have 20 new corrections for 3 new errors. Certainly, we have lost 22 of the original 42 corrections, but we have also reduced the errors from 48 to 3.

When reviewing changes in the unannotated corpora, we now found about 0.7 changes per 1000 words. The vast majority concerned time expressions, with a few for the antecedent subjects, and, not surprisingly, not a single case of *Monsieur le Président*. Among the 54 changes reviewed (approximately 10 per corpus), there were 45 corrections and 8 errors. Most of the corrections were simple time expression changes, where the time expression was correctly marked as an adjunct of the following verb instead of as the subject, as in the following example, emblematic of the wide variety of subject matter found in the Wikipedia corpus:

Example 6.26 *Un **jour**, en se levant du siège des toilettes, il se **rend** compte que ses cuisses ont grossies et saisi de tristesse, se met à pleurer.*

There is one rather elegant correction for a relative clause in the Frantext literary corpus, with three items corrected in one fell swoop:

Example 6.27 *Et **celui-ci**, qui **vole** à la lisière, c'est le redwins black bird.*

In the original annotation, we had both *celui-ci* and *qui* as the subjects of *vole*, and the main verb *est* as a modifier for the subordiante verb *vole*. With the rule in place, *celui-ci* is correctly marked as the subject of *est* and the antecendent of *vole*, and *est* is correctly marked as a conjunct of the opening *et*. However, most corrections for antecedents simply marked the antecedent as an adjunct of the subordinate verb instead of as its subject. This rule could probably be replaced by a feature for more effect, since in the training corpus, 100% of the cases concern those where σ_0 indeed governs β_0 as an antecedent.

Note in passing that the group of time-related nouns could probably be incorporated into features as well, since its occurrences are numerous in the training corpus, and its behaviour must be unusual in other contexts as well.

The remaining errors are a mixed bag: one that caught our attention, provoking 3 errors, was the use of *seconde* as a pronoun ("the second one") rather than a time expression. Such ambiguous words should probably be removed from the rule, to avoid this sort of situation.

6.5.4 Parser rules: prohibiting relations across parentheses

Another error that we noticed when reviewing duplicate subject errors was the case of a subject being erroneously marked within parentheses. Take the example below, with both subjects and the verb marked in bold:

Example 6.28 *Après huit réunions infructueuses avec le patronat de la sidérurgie à propos des augmentations salariales de 1992 dans la Ruhr (le **Monde** du 15 janvier), le **syndicat** de la métallurgie IG Metall a **décidé** [...].*

This led us to imagine a rule where items in parentheses cannot have certain relations with items outside of the parentheses, formalised as follows:

Relations across parentheses: Do not allow an item inside parentheses to govern an

item outside the parentheses, and do not allow an item inside parentheses to be a verbal argument of any type for a verb outside the parentheses, or to be a conjunct, unless the coordinating conjunction is inside the parentheses as well.

Regarding the parentheses rules, once we modify the rule to exclude numbers in parentheses that indicate enumeration, the results are positive on the whole, with 51 new corrections for 23 new errors, 4 of which are actually annotation errors. Most of the remaining errors result from Talismane applying an erroneous `shift` transition inside the parentheses, after which it is unable to perform attachments across the parentheses. It may be possible to rework the rule to exclude certain `shift` transitions, but this has to be done with great care and is likely to add considerable complexity, since `shift` transitions are still required inside the parentheses anytime a word has a right-hand governor.

6.6 Discussion

In this chapter, we presented methods for injecting linguistic knowledge into the parsing process intrinsically, that is, by defining targeted features and rules that describe typical linguistic structures within the training corpus to help the parser make certain decisions. In terms of features, targeted pos-tagger features provided very promising results, especially after refinement against the training corpus. More complex features, such as the parser coordination features, only managed to correct a small portion of remaining errors, even after considerable refinement. Whether further feature refinement and/or training corpus annotation corrections would allow them to be more effective is an open question. Our experience, based on the more successful case of *que*, is that further refinement can indeed eventually lead to clearly positive results. First of all, it takes a considerable imbalance in the labels for a feature to have a preponderant effect. Secondly, cases where the feature incorrectly grouped certain occurrences with a different equivalence class are particularly dangerous, since during analysis such occurrences will be bunched together with the wrong class and induce the parser to take the wrong decision. In such cases, adding the feature does more harm than leaving the baseline features only.

Note that the actual importance assigned by the classifier to the features is outside of our control in the present study. It often happens that even the most carefully designed features are drowned out by the mass of other features, so that they do not affect probabilities sufficiently to tip the balance in favor of the correct decision. This feature "drowning" is somewhat expected. The vast majority of features in the model are baseline features with very low information content. They nudge the results in a certain direction, favouring certain transitions slightly over others, but it's only through the combination of a myriad of such features that we get a final result that is fairly accurate. There is an inherent conflict between this huge mass of low-information content features, necessary for getting fairly accurate results on the mass of data without needing to explore the data itself deeply, and a few high-information content features, capable of correctly determining a few specific cases.

Features thus allow us to play the game of machine learning as it was meant to be played: the linguist attempts to give the classifier useful knowledge, but it is up to the classifier to decide what to do with this knowledge based on the training data it has at its disposal. On the other hand, rules allow us to tweak the system by bypassing the statistical model and

giving our own version of the truth. This may seem futile to a purist, but they do allow us to target and correct unambiguous surface phenomena, especially when they are too rare to be correctly handled by a statistical system. Moreover, in some cases, rules are successfully applied with much wider coverage, e.g. in the case of constraints surrounding closed classes.

One advantage of rules is that the decision as to apply them or not is taken when analysing, without requiring a model to be retrained. Thus, when analysing less formal text, the rule for not assigning *que* as an adverb if it is not preceded by *ne* can be discarded.

The main difficulties with rules are:

- The need to define very specific unambiguous cases—by following the methodology defined here, and refining the rules after examining results.
- Long-term maintenance is more difficult, as we could have hundreds of rules for various specific cases.
- Rules can only cover a very small percentage of errors, where idiomatic usage guarantees there will be little or no ambiguity.

Overall, parser rules were far more difficult to implement than pos-tagger rules, and correcting a phenomenon locally often provoked other errors at a greater distance. The difficulty of properly visualising the shift-reduce algorithm's mechanism makes it difficult to write rules correctly. Even rules that seemed obviously true, such as not allowing a verb to have an object inside parentheses, often created problems, when the $\texttt{leftarc}_{\text{obj}}$ transition was replaced by a `shift` transition, which blocked further dependencies across the parentheses. Still, the net results for some of the rules are clearly positive, especially in the case of very simple rules such as time expressions followed by commas, and lead us to believe that rules can be applied successfully with a sufficient understanding of the shift-reduce algorithm, if enough testing is performed to avoid undesirable side effects.

We do not pretend in this chapter to capture any sort of deep linguistic knowledge. The rules are almost guaranteed to be close to the surface, in that delving any deeper into the sentence structure is likely to bring about too much complexity to make unambiguous rules possible. Our coordination features delve somewhat deeper than the others in this chapter, by looking deeper into the stack, which is a reflection of the dependencies that have already been constructed (and more specifically of words that have been eliminated as having no more dependents). Still notions such as "surface" and "deep" are relative: we have indeed delved far deeper than the baseline features, which are limited to n-grams and some top-level partial dependency structures. In the next and final chapter, we attempt to extend the scope of the exploration begun here, not by delving any deeper, but by introducing wide-coverage external resources using similar mechanisms to the ones explored in this chapter, and seeing if we achieve a more wide-ranging positive effect.

Chapter 7

Improving parsing through external resources

In the previous chapter, we attempted to identify ways to improve the analysis results intrinsically, by defining better features or rules which attempted to capture more complex regularities within the corpus itself than those captured by the baseline model. In this chapter, we step outside of the training corpus, and see to what extent we can improve the analysis through the use of external resources. In many ways, this chapter is a natural extension of the previous chapter: the same methods are used to inject knowledge, but the knowledge is gathered from far more extensive bases.

There are two primary reasons for the inclusion of external resources, each of which requires a different type of resource, and especially a different methodology for constructing it. The first reason is **generalisation**: an attempt to generalise our system's applicability beyond the specific data found in the training corpus, but without aiming at a particular domain or genre. The second reason is **domain adaptation**: an attempt to improve a system's performance when applied to a specific domain that is clearly different from the training corpus domain. In the first case, we will tend to use comprehensive resources for "general" usage of the language, and to find ways to describe our training material in terms of these comprehensive resources. In the second case, we need to find ways to cross the domain bridge: somehow to create links between patterns found in the training material and equivalent patterns found in the out-of-domain material.

In the first two sections of this chapter, we tackle the generalisation problem. In section 7.1, we examine the question of incorporating knowledge from specialised structured lexical resources, in which entries are grouped together by certain syntactic behaviour they share. In section 7.2, we investigate whether improvements are attainable by retraining Talismane on a wider coverage lexicon.

Although we do not tackle domain adaptation directly in this thesis, there have been numerous recent experiments in this area, and given our interest in terminology extraction as an application for parsed text, experiments in a similar vein are one our primary future perspectives. Furthermore, most domain adaptation methods make novel uses of semi-supervised approaches, i.e. approaches which gather information from automatically parsed corpora and somehow feed it back into the parser to attempt to improve its out-of-domain results. Many of the methods used can be applied to the generalisation problem as well, as long as the corpora used are general language corpora. We therefore include a short state-of-the-art for

semi-supervised domain adaptation in section 7.3, before attempting our own experiment using a semi-supervised approach.

This experiment is described in section 7.3.3, presenting the construction of a distributional semantic resource from Talismane's analyses, and using this resource to improve parsing accuracy for coordination via targeted features.

7.1 Incorporating specialised lexical resources

The attempt to generalise typically comes down to assigning a word form in the training corpus to some larger class of word forms that display similar syntactic behavior.

When defining the targeted features in chapter 6, we naturally attempted to make such generalisations on a small scale. For example, in the case of *que*, we were led to create a list of verbs which sub-categorise with *que* as a direct object, such as *dire*, *penser*, etc. Similarly, we created another list of nouns which tend to govern *que* as a subordinating conjunction directly, such as *fait* in the expression *"le fait que..."*. Many other such possibilities exist in the corpus and are yet to be explored: we already mentioned the need to redefine our list of verbs governing *que* to include more complex verbal expressions, such as *se rendre compte que*, since the expression *rendre que* does not exist. Similarly, when discussing parser rules in section 6.5.3, our final version of the duplicate subject rule included a list of nominal time expressions that are not introduced by a preposition, and yet typically act as adjuncts, such as *"la semaine dernière"*.

In the previous chapter, the resources were gathered directly from the training corpus by reviewing the occurrences in it, and generally augmented or restricted by our own knowlege of the language. They were directly incorporated into features (and rules) by constructing the feature around a true/false value indicating whether a given token belonged to the group or not—instead of constructing the feature directly around the token's word form or lemma.

Talismane provides three basic methods for representing the group within a feature. The first is a boolean function checking if a particular item is in a particular set of words, all of which are directly written into the declarative feature. This can be used for short lists, e.g. for checking if the lemma of the head-of-buffer is in the set {*être, avoir, faire*}. The second is a function which takes an external file in a specific format, containing a mapping from a set of keys to a set of classes. For example, this file could be used to map a large number of pos-tag/lemma pairs to semantic cluster identifiers. Thus, we could map NC/*pomme* and NC/*pain* to the class *food*, or NC/*voiture* and NC/*vélo* to the class *vehicle*. This is then incorporated into the feature as a string which replaces the original word, representing an equivalence class for syntactic purposes. The final method is a file used for lexical affinity or semantic similarity purposes: this file maps pairs of words to a numeric score. The score can then be used within features, either as a threshold, or for comparison with the score of another pair.

The latter two methods can be used with much larger resources. Indeed, there have been several fairly systematic efforts in constructing resources that compile structured syntactic information for French. One is Dicovalence[1] [Van Den Eynde and Mertens, 2006], a dictionary of verbal subcategorisation frames. It would be worthwhile to try to replace or augment the list for *que* that we constructed from our training corpus with entries from Dicovalence. Dicovalence does not include fixed nominal or adverbial complements, but does include some

[1] `http://bach.arts.kuleuven.be/dicovalence/`

fixed clitics (e.g. reflexive verbs). It has been used by others successfully for other features as well, such as prepositional phrase attachment [Anguiano, 2013].

Another interesting resource to explore would be Verbaction[2] [Tanguy and Hathout, 2002], giving verb/noun couples where the noun is both morphologically related to the verb, and indicates the action taken by the verb, such as *zapper/zappage* and *zapper/zapping*. An interesting question here would be: do these nouns tend to take the same prepositional and/or semantic arguments as the verbs? For example, the verb *lutter* and the noun *lutte* both take an argument introduced by the preposition *contre*. In the case where one member of the pair is much more common or has a known prepositional argument (e.g. from Dicovalence), we could use this information to predict prepositional attachment on the related member.

The possibilities offered by such lists and resources are enormous in terms of targeted features and rules, and are probably the most immediately promising prospects for improving the linguistically motivated features/rules presented in the previous chapter.

7.2 Augmenting the lexicon with GLÀFF

Another opening Talismane gives for generalisation is through the lexicon interface, described in section 4.3.2, page 114. We make considerable use of the lexicon within our baseline features, both in the pos-tagger, when trying to guess the pos-tag of a word, and in the parser, for various features based on the word's lemma and others checking for matches in the morphological aspects (gender, number, tense, etc.) of the two words currently being considered for dependency.

Up until now, we only presented experiments using the LeFFF lexicon (section 4.3.3, page 115). With the recent availability of an alternative wide-coverage lexicon in the CLLE-ERSS laboratory, GLÀFF [Sajous et al., 2013], we became interested in the question of the importance of the lexicon in our evaluations. GLÀFF (Gros Lexique À tout Faire du Français) contains 1,425,848 inflected forms, as opposed to 404,483 in LeFFF, and is unusual in that it was derived entirely from the crowd-sourced Wiktionary website for French[3]. It contains detailed morpho-syntactic characteristics, lemmas, and phonological characteristics (which are absent from LeFFF, but are not useful in the present thesis). With respect to LeFFF, it is missing subcategorisation frames. We therefore decided to run an experiment in which we alternatively replace and augment LeFFF with GLÀFF.

In addition to the difference in size, there is a difference in philosophy: LeFFF was built by a small team semi-automatically, by analysing corpora for missing entries and applying a common methodology to ensure quality, whereas the source material for GLÀFF was presumably built by a large, unknown team, and there is no official methodology. One advantage to GLÀFF is that the crowd-sourced resource is constantly being improved, so that new versions can automatically be generated by downloading the resource and running the same derivation rules.

Although it seems reasonable that a larger lexicon, if the quality is good enough, would give better results, the quality of a crowd-sourced resource is yet to be proven. Moreover, the size of the lexicon may not be very important after a certain critical mass has been reached, covering the vast majority of words in most standard corpora.

[2] `http://redac.univ-tlse2.fr/lexiques/verbaction.html`
[3] `http://fr.wiktionary.org`

Because of the critical nature of closed-class words to syntax analysis, we created our own customised list of closed-class words, which replaced those of both LeFFF and GLÀFF in our experiments. We are therefore only concerned with differences in open-class words: common and proper nouns, adjectives, adverbs, verbs and, to a far lesser extent, interjections. Our initial tests of the pos-tagger output showed interesting updates by GLÀFF, but many errors when it came to geographical names, which are entirely missing from this lexicon. We therefore extracted the large list of geographical names from the LeFFF (approximately 53,000 names), and ran a second test run of GLÀFF augmented by these names.

Table 7.1 below shows the lexicon coverage for each of the evaluation corpora, where the three columns on the left show the % of unknown words among all token occurrences in the corpus, and the three columns on the right show the % of unknown words in a lexicon compiled from unique forms found in each corpus. The GLÀFF columns include the LeFFF geographical names. Unexpectedly, LeFFF has better coverage for all corpora, except for the two crowd-sourced Wikipedia corpora: articles (FrWiki) and discussions (FrWikiDisc), perhaps because of a similarity in lexical usage between the Wiktionary and Wikipedia communities!

	% unknown occurrences			**% unknown word forms**		
	LeFFF	**GLÀFF**	**Both**	**LeFFF**	**GLÀFF**	**Both**
FTBDep dev	3.98	4.77	3.63	11.40	12.07	10.59
FTBDep test	3.36	4.00	3.02	9.79	10.24	8.69
EstRépu	3.97	4.35	3.55	9.94	10.52	8.85
Europarl	2.74	3.96	2.38	5.53	6.71	4.87
FrWiki	8.64	8.20	7.79	18.10	17.88	16.76
FrWikiDisc	5.12	4.50	4.31	12.79	11.15	10.54
EMEA dev	7.41	8.34	6.66	13.59	14.62	12.03
EMEA test	5.67	6.14	5.02	9.30	10.17	8.29

Table 7.1: Coverage by lexicon (percentage of unknown words)

When looking into the details for the FTBDep dev corpus, some of the words present in GLÀFF but missing from LeFFF include a number of rare words or neologisms: *herbagers*, *privatisables*, *tripalium*, *eurosceptiques*, *hyperinflation*, *popularisation*, *ruralité*, *zappeurs*, *concèderont* and *préfèreront*. Words present in LeFFF but missing from GLÀFF include some much more common words: *nouvel*, *orthodoxes*, *terrestres*, *vraies*, *après-midi*, *assurance-chômage*, *baby-sitter*, *cul-de-sac*, *coeur* and *manoeuvre* (though the latter two exist with the "official" spelling *cœur* and *manœuvre*, containing the ligature "œ").

Despite these differences in coverage for the evaluation corpora, there are no significant differences in the overall pos-tagging accuracy for the full set of corpora. Indeed, in individual corpora, the coverage is not necessarily correlated to the scores. There are only four corpora where the difference between LeFFF and GLÀFF are significant: EstRépu, EMEA dev, EMEA test and FrWikiDisc. Among these, GLÀFF has better accuracy for EMEA dev and EMEA test, where its coverage is lower, and worse accuracy for FrWikiDisc, where its coverage is higher. In the EMEA corpora, the corrections concern mostly proper vs. common noun differences, where the common nouns are highly specialised medical terms missing from both lexicons. In FrWikiDisc, there are a majority of past participle/adjective differences for known words, and proper/common noun differences for unknown words. These differences

are among the least important for parsing, since they rarely result in a different governor or label.

Although both lexicons contain compound words separated by either spaces or by dashes, LeFFF coverage seems better. In our evaluation corpora, and especially FTBDep, there are many compound words inherited from the FTB which are missing from both lexicons. In analyses of unannotated corpora, there will be far fewer compounds, since Talismane only recognises a small set of mostly closed-class compounds, all of which have been added to both lexicons for these tests.

In terms of analysis results, the differences between LeFFF and GLÀFF are fairly orthogonal, with 693 new corrections for GLÀFF against 732 new errors. This is ≈0.5% of all dependencies: a big proportion when we consider accuracy is already at around 96.5%. This gives some hope for improvement when combining LeFFF and GLÀFF together, but unfortunately the differences are even less significant and remain orthogonal, with the combined lexicons introducing 530 corrections for 505 errors. Given the high level of ambiguity in the typical categories for the differences (e.g. past participle vs. adjective), this is not surprising.

We then turned to a parsing result evaluation after applying a pos-tagger and parser pair trained on the same lexicon. In this test, LeFFF significantly outperformed GLÀFF on its own, with highly orthogonal results: GLÀFF on its own introduces 1,763 corrections for 2,553 errors, out of a total of 135,024 dependencies. Thus, new corrections account for 1.3% of all dependencies. If we could somehow combine the two resources successfully, this represents a huge gain. Unfortunately, when we combine LeFFF and GLÀFF, there is no significant difference: a total of 723 new corrections for 774 new errors (p-value > 0.15). The only corpus for which results vary significantly when combining the two lexicons is FrWiki, with 167 new corrections for 100 new errors. A much more detailed analysis of the errors would be required to see if we can somehow combine these resources usefully—but this is rendered difficult by the fact that both errors and corrections often concern words unknown in both lexicons.

Overall, the tests with GLÀFF have not produced significant results. Although some of the evaluation corpora contain a large percentage of unknown words, these words tend to be unknown in both lexicons. Moreover, corpus coverage is not directly related to pos-tagging accuracy. In parsing, it would seem that the crowd-sourced resource on its own does not provide sufficient quality: however, an analysis of the specific errors is necessary to see if the problems are truly related to lexicon quality, or rather to internal assumptions made by Talismane, which was constructed for several years using LeFFF, and only tested with GLÀFF near thesis completion. We have not yet been able to combine the two lexicons in a way that takes advantage of the considerable orthogonality between the correct answers provided by each lexicon.

7.3 Injecting resources built by semi-supervised methods

In the previous sections, we reviewed resources created by some external process, typically involving some degree of manual input or supervision. We now turn to the injection of information gathered directly from automatically parsed corpora. Using such information to improve parsing is known as a semi-supervised approach.

As mentioned in the introduction, a lot of work with semi-supervised approaches concerns domain adaptation, when we try to improve parser results for a corpus whose domain is

clearly different from that of the training corpus. We therefore begin with a state of the art for semi-supervised methods in domain adaptation.

7.3.1 Semi-supervised methods for domain adaptation

Domain adaptation was first tackled at a fairly large scale in the CoNLL 2007 shared task [Nivre et al., 2007a], which evaluated a corpus from the biomedical and chemical domains, in a scenario with no available annotated resources. The results confirmed the difficulty of cross-domain parsing: while unlabeled accuracy for general language text is often higher than 90%, the best CoNLL 2007 unlabeled accuracy for out-of-domain texts was 83.58%, with labeled accuracy as low as 81.06%. This score was achieved by Sagae and Tsujii [2007], who used a semi-supervised approach in which two different parsers parse unannotated out-of-domain texts, and identical parses are extracted and added to the training set for out-of-domain parsing.

Using the same data from CoNLL 2007, Kawahara and Uchimoto [2008] used a similar semi-supervised approach, but with only a single parser. They built a binary classifier for unannotated out-of-domain parses which separated them into reliable and unreliable parses. After parsing a large amount of unannotated out-of-domain text, they extracted all reliable parses and added them to the training set. They managed to achieve an unlabeled accuracy of 84.12%.

Yoshida et al. [2007] use a different approach to domain adaptation, based on the observation that it is cheaper to build a pos-tag training set than to build a parser training set. They compared approaches which provided a single pos-tag to the parser and multiple ambiguous pos-tags to the parser (with probabilities). They also compared pos-tags provided by in-domain training vs. pos-tags provided by out-of-domain training. They concluded that providing multiple ambiguous out-of-domain pos-tags significantly increases out-of-domain parsing scores, coming very close to the use of the gold pos-tags found in the out-of-domain parse evaluation corpus.

For genre adaptation, McClosky et al. [2006] and McClosky et al. [2008] describe a semi-supervised approach in which both "re-ranking" and "self-training" are applied. Re-ranking is a method by which the n best parses are examined in a post-processing phase and re-ranked based on a rich set of features unavailable at parse-time. "Self-training" is a technique in which unannotated data is parsed and added *unfiltered* to the original training data, after which a new statistical model is trained. Their experiments use various parts of the Penn Treebank. The original in-genre training data is the journalistic Wall-Street Journal corpus (WSJ), whereas the out-of-genre evaluation data is the annotated portion of the Brown corpus, consisting mostly of works of fiction. They report significant gains: the baseline version in which a WSJ-trained parser is evaluated against Brown gives an accuracy of 83.9%, with 85.8% after re-ranking. Adding 2500k sentences from an unannotated news corpus raises the post re-ranking score to 87.7%. The choice of an unannotated news corpus seems rather odd—why not add results from an unannotated fiction corpus? Nevertheless, the results are convincing.

Sagae [2010] worked on the same set of data for initial training and evaluation. There are a few major differences: first, they apply simple self-training without any re-ranking. Second, they self-train using data from fictional novels instead of news sources. Finally, they don't give any additional weight to the original, manually annotated data. Their intuition was that correct parses would be systematic enough to help performance, whereas parse errors would be

distributed more or less randomly. Their results after self-training show a slight decrease in f-score on the original WSJ corpus (from 89.13% down to 88.06%), but a significant increase for the fictional genre (from 83.56% up to 85.42%). Perhaps more significantly, when evaluating semantic role labeling task using the parse output as input, their parses outperform McClosky et al's parses despite a lower parsing accuracy. Both of these studies used constituent rather than dependency approaches.

Candito and Crabbé [2009] and Candito et al. [2011a] discuss a word clustering approach for tackling domain adaptation. They construct word clusters that span both the source and target domain using an unsupervised clustering algorithm [Brown et al., 1992]. The clustering is constrained to minimize loss of likelihood on bigrams contained in the training corpus. They then conduct experiments in which tokens are replaced by their cluster id, and combine this with unfiltered self-training from the target domain. Their results show maximal gain from self-training, with a more limited gain from word clustering. In Candito et al. [2012], the Sequoia corpus, already presented in section 4.2.1 (page 109) and used for evaluation throughout this thesis, is parsed using the same word clustering techniques. They report significant parsing improvement.

Petrov et al. [2010] suggest a method similar in some ways to Sagae and Tsujii [2007], except that they place the parsers in sequence, using a high-accuracy parser with $O(n^3)$ [Petrov et al., 2006] to provide the in-domain training material to a transition-based linear-time parser (a method they call "uptraining"). In their experiment, they evaluate performance on the QuestionBank [Judge et al., 2006], and show initial loss of 20% accuracy for the transition-based parser as compared to the WSJ corpus. After uptraining on 100K unannotated questions, transition-based parser LAS improves from 60.06% to 76.94%, and UAS improves from 72.23% to 86.02%. If in addition to 100K automatically annotated questions, the transition-based parser is given 2K manually annotated questions, LAS improves from 78.27% to 84.02%, and UAS improves from 83.70% to 88.38%.

Mirroshandel et al. [2012] suggest a semi-supervised approach for generalisation, rather than domain adaptation. In their method, frequent lexical combinations from predicted syntactic dependencies are automatically learned from a large parsed corpus, and used to measure a "lexical affinity" score for word pairs. They then use the lexical affinity matrix to help graph-based parsing of the FTBDep corpus via three different methods: traditional training features, post-processing corrections, and double parsing where the combination is re-introduced as a constraint and the sentence re-parsed. Rather than relying directly on parser confidence to select high-quality lexical combinations, they rely on an ambiguity measure, which checks whether the same dependency exists in all of the n-best parses produced by the parser. They report a 7.1% relative decrease in the LAS error rate. A similar method was tested in Mirroshandel et al. [2013] to enforce verb sub-categorisation frame constraints.

Anguiano [2013] uses a similar approach using lexical affinity, but applied to a transition-based parser, and reports statistically significant gains for prepositional phrase attachment using a combination of sub-categorisation frames from Dicovalence [Van Den Eynde and Mertens, 2006] and lexical affinity, as well as a new metric for calculating the affinity which takes the context into account.

This variety of methods, using both raw and parsed corpora to attempt to improve parsing, enables us to envisage parsing improvement without having to annotate new material in expensive annotation campaigns. As Alexis Nasr quipped in a presentation he gave in Toulouse in 2013, "Now we finally know what syntax analysers are good for: making better syntax analysers."

As already mentioned, many of the methods for using information in a semi-supervised approach need not apply to domain adaptation only. We now turn to an experiment in a similar vein, using automatically constructed distributional semantic resources to attempt to learn generalisations from the training corpus data with respect to the semantic similarity of conjuncts.

7.3.2 Distributional semantic resources

> A word is known by the company it keeps. [Firth, 1957]

The quote above, taken from a linguistic article, echos Aesop's fable: "I know what kind of donkey he is because of the company he keeps". It also happens to be the standard English translation of a legal canon in Latin: *Noscitur a sociis* (literally "It is known from its associates")—which indicates that when a word in a legal document is ambiguous, its meaning is derived from the surrounding words. This traditional legal canon was first articulated as a linguistic theory by Harris [1954]. His distributional hypothesis states that similarity between two words is proportional to the number of contexts that the two words share.

When applied to corpus linguistics, this theory is put into practice by constructing a similarity matrix between the words of a corpus, based on the contexts they share. The contexts can either be collocations (e.g. n words to the right and left, same paragraph, same document) or identical syntactic dependency arcs gathered from automatic syntax analysis of the corpus.

Following the method described in Turney et al. [2010], the first step in constructing the similarity matrix is the construction of a large $n \times m$ matrix in which rows are words (or, more often, the guessed pos-tag/lemma pair corresponding to the word), and columns are contexts. In this thesis, the contexts are always taken to be syntactic dependency arcs. Thus, each word is represented by a sparse vector, into which we enter the counts of its occurrences in the analysed corpus as the governor or dependent of a particular syntactic triplet of the form (governor, relation, dependent). Due to this vectorial representation of words, distributional semantic resources are also known as vector space models.

For example, the word *pomme* might appear 20 times as the object of *manger*, 10 times as the object of *éplucher*, 5 times as a dependent in *tarte aux* ..., 3 times as the governor of the phrase *d'amour*, and once as the subject of *tomber*. Each of these syntactic contexts would have a column of its own, we would place the counts above into the corresponding columns for the row associated with the word *pomme*. Other words with non-zero counts in the column *obj/manger* might be *pain* and *orange*, and other words in the column *obj/éplucher* might be *orange* and *banane*.

The next step is typically to weight each entry in the matrix by the amount of information it contains. This is an attempt to give more weight to unusual combinations, which imply a tendency to collocation, than to common ones, which could occur by mere chance. There are a myriad of weighting methods, the most common of which is PMI (pointwise mutual information), which takes into account the marginal counts of both the word and the context, giving a higher weighted score to rare words (e.g. words with few contexts) occurring with rare contexts (e.g. contexts that are not shared by many other words).

Finally, we measure the similarity between each different pair of word vectors, using a variety of similarity measurements, the most common being the cosine, giving the cosine of the angle between the two n-dimensional vectors. Each similarity measurement is entered

into a large symmetric $n \times n$ matrix with the same list of words as both the rows and the columns.

This information can then be used in place of a manually constructed semantic resource. It has the advantage of giving a principled numeric measurement of word similarity, and of being directly derived from actual word occurrences in a given corpus. On the down side, since the similarity is entirely derived from shared contexts, it confuses literal and figurative usage, does not take into account the different polysemic uses of a given word, and incorporates parser analysis errors into the model. On the assumption that these errors are not systematic, they are assumed to have minimal effect on the final resource.

Within the CLLE-ERSS laboratory, considerable work has been performed on distributional semantic resources, constructed using the Upéry software [Bourigault, 2002]. The history of this work is presented in Fabre [2010], including the construction and use of the resources, and various linguistic applications. Among these, Fabre uses measurements derived from the original word-context matrix to attempt to correctly identify prepositional phrases as verbal arguments or adjuncts. This is a problem directly applicable to the current thesis, and it would be interesting to attempt to replicate this work within Talismane's statistical framework. She also describes attempts to use the distributional semantic resources to help segment a text thematically. This task is further expanded in Adam [2012], in which filtered semantic similarity links are shown to significantly improve thematic text segmentation. Adam also uses distributional semantics to help distinguish between pairs of discourse relationships having similar surface markers.

There is an inherent difficulty in evaluating the quality of distributional semantic resources, outside of a given application. Given their method of construction, while they do contain synonyms, antonyms, co-hyponyms, etc., they also contain a large number of non-traditional relations that are more difficult to classify, and are therefore not directly comparable to traditional semantic resources. In Morlane-Hondère [2013], an attempt is made to compare distributional semantic resources to dictionaries of synonymy and other, less traditional crowd-sourced resources, and to highlight the type of relations that are likely to appear. Unfortunately, the specific cases of co-hyponyms and co-meronyms are not evaluated in this work, since these two relations are not treated as a separate category in the reference resources. These relations interest us in particular, as they seem to be the relations most likely to be found between conjuncts.

7.3.3 Using the similarity between conjuncts to improve parsing for coordination

In this section, we attempt a semi-supervised improvement of Talismane's analysis by re-injecting a distributional semantic resource into the training model. The first question is how to effectively incorporate such resources, giving the pairwise similarity of a large lexicon, into the parsing process. The most straightforward approach is to group individual word forms into semantic clusters, emulating a resource such as WordNet [Fellbaum, 1998]. We could then design a feature which returns, for example, the lemma of a verb and the semantic cluster of the noun which is a candidate for attachment to this verb. If the verb is *manger* and the noun is found in a semantic cluster containing various types of food, we would expect a direct object dependency.

However, the vocabulary of our training corpus lexicon is entirely in the journalistic genre and largely limited to the domains of politics, economics and finance. If we take a distri-

butional semantic resource constructed from a large general-purpose corpus, and attempt to construct semantic equivalence classes via clustering, we will end up with many classes having no occurrences at all in the training corpus and, if they do, the occurrences may well be in metaphors or fixed expressions with a skewed distribution. For example, the word "*cheval*" appears more often in the FTB inside the expression "*cheval de bataille*" (a politician's main theme or argument) than as an actual animal. Most of the clusters will have so few occurrences as to not have the statistical weight required to affect the parsing correctly. One corrective approach is to seed the clustering process with clusters built from the training corpus alone, and then to add the words found in the larger corpus to the training corpus clusters only. This is similar to the approach by Candito and Crabbé [2009], although they use n-gram based clusters learned from raw corpora using the method described in Brown et al. [1992], rather than distributional resources from parsed corpora. However, our experiments attempting to construct classes from distributional resources by training class seeding have so-far proved unsuccessful in improving accuracy significantly.

Another question of the clustering approach is how to handle polysemy: should a word only belong to a single cluster, or should we allow it to belong to multiple clusters? If the word is allowed to belong to multiple clusters, should the word's membership in a cluster be weighted to indicate its distance from the cluster's centroid? Although our robust statistical methods all represent features as numeric values, and could somehow incorporate weight into the features, we have been unable to improve accuracy with either monosemic or polysemic clustering approach

And yet, when examining the nearest neighbours of any given word subjectively, it is quite clear that the resource contains a lot of semantic information, since the nearest neighbours are all clearly semantically related. We therefore decided to turn to methods for integrating this resource which rely purely on the distance between two given words, rather than on effective clustering. This excludes most dependency types, since they combine different categories, such as nouns and verbs, which would not be found to be similar in a distributional semantic resource constructed from syntactic dependencies. The one exception is coordination. While *eat* and *apple* may not be semantically similar in the sentence "I'm eating an apple", *apple* and *banana* may well be semantically similar in "I'm eating an apple and a banana".

We thus make the hypothesis that coordinated words tend to be semantically closer to each other than to other words of the same category in the sentence. Following the discussion in section 6.4.1, page 175, we are particularly interested in comparing the correct coordinated pair with other words which would be considered for coordination during transition-based parsing: typically the series of nouns in a nested set of prepositional phrases prior to the coordinating conjunction. Take, for example, the following example sentence from the FTBDep dev corpus:

Example 7.1 *Tokyo envisage différentes mesures destinées à faciliter les importations d'automobiles : une révision de la procédure d'homologation des véhicules (acceptation des* ***tests*** *américains pour les freins et des* ***normes*** *internationales pour les appuis-tête).*

Structurally, it is clear that the second conjunct is the prepostion *des* governing the noun *normes*. For the first conjunct, we have to select between the preposition governing *tests* and the one governing *freins*. Ignoring other strong clues in the sentence (the use of identical prepositions and their parallel structure *des X1 pour Y1 et des X2 pour Y2*), we would expect *tests* to be semantically closer to *normes* than to *freins*.

To perform our test, we first constructed a set of distributional semantic resources from medium-sized corpora. The corpora selected are:

- **Le Monde:** all of the articles from the daily French national newspaper Le Monde over a period of 10 years (1991-2000). 150 million words.
- **Wikipedia:** all of the articles from a snapshot of the French wikipedia taken in 2008. 250 million words.

We analysed the corpora using the baseline version of Talismane with a beam width of 2 and beam propagation activated between the pos-tagger and parser. In Talismane's output, we included the parser confidence in each dependency. We then constructed multiple distributional semantic resources as follows:

- From the corpus Le Monde, Wikipedia, and both combined.
- Applying a parser confidence cutoff (see section 5.3, page 142) of 70%, 80%, 90%, 95% and 99%, as well as no cutoff at all

The resources were constructed by Franck Sajous (CLLE-ERSS laboratory) using the Upéry software. We used the Lin score [Lin, 1998] to measure similarity, rather than the more common cosine measure. The filters used to construct the resources were as follows: each pos-tag/lemma pair must appear at least twice in the corpus, the item on each side of the predicate must combine with at least 3 different items in the corpus, and the Lin score must be ≥ 0.1. The table below shows the number of word pairs retrieved from each corpus at each different confidence cutoff, using the above thresholds.

Cutoff	**LM10**	**Wikipedia**	**Both**
None	10,042,268	5,881,040	19,630,673
70%	8,875,324	5,191,657	17,295,300
80%	7,803,098	4,565,720	15,103,758
90%	5,931,717	3,508,455	11,314,324
95%	4,349,563	2,631,865	8,198,631
99%	1,050,686	362,950	1,994,874

Table 7.2: Number of word-pairs per corpus and confidence cutoff

In order to test our hypothesis, we first checked similarity scores for all coordination parsing errors in the FTBDep development corpus. In the case of prepositions, we took the object of the preposition as the basis for comparison, rather than the preposition itself. In this initial test, we used four different semantic resources: Wikipedia 70 and 99, and Le Monde 70 and 99. Resource coverage for the correct pair ranged from 15% (Wikipedia 99) to 52% (Le Monde 70). For the incorrect pair it ranged from 9% (Wikipedia 99) to 39% (Le Monde 70). Cases were both pairs were covered ranged from 3% (Wikipedia 99) to 30% (Le Monde 70). We then looked at the difference between the similarity scores for the correct pair and the incorrect pair, assuming any pair absent from the resource was given a score of 0, and counted the number of times the difference in Lin score was greater than n in favor of the correct pair, versus the number of times the difference was greater than n in favor of the incorrect pair, with n in $\{0.05, 0.1, 0.2, 0.3\}$. This gave us a ratio between two counts:

the best ratio of 1.5 was achieved for Le Monde 70 with a difference of 0.1—but all other comparisons yielded ratios between 1.2 and 1.5, except for Wikipedia 70 at a threshold > 0.1 (ratio = 1.0). This indicated that there is a perceptible difference in similarity between correct guesses and incorrect guesses, but the difference is not systematic enough to use in a rule: rather, we decided to construct a targeted feature, which would favor a dependency between similar conjunct candidates, without imposing it.

When designing this feature, we used the same heuristic for guessing the 2nd conjunct as was presented in section 6.4.1. We then setup the following feature: for two coordination candidates, check if an earlier 1st conjunct candidate exists with a higher similarity than the current 1st conjunct candidate. Answers where the difference in score was below a certain threshold were ignored. We tested this feature on all distributional semantic resources, and for thresholds in {0.001, 0.05, 0.1, 0.2}. Again, for pairs absent from the distributional semantic resource, we assumed a score of 0.

The results show a small, but significant gain for the use of distributional semantic features, over the use of targeted features alone. We show the results for all evaluation corpora (dev + test) on the label `coord`, using the mixed resource (Le Monde + Wikipedia) at a confidence cutoff of 90%, and a score difference of 0.1 :

`coord`	**baseline**	**targeted**	**distSem**
true+	2230	2262	2281
false+	25	27	27
false-	1687	1655	1636
precision	98.89%	98.82%	98.83%
recall	56.93%	57.75%	58.23%
f-score	72.26%	72.90%	73.29%

Table 7.3: Parsing improvement for coordination using distributional semantic features

When comparing the model with both targeted features distributional semantic features to model with targeted coordination features only, we have 41 new corrections for 22 new errors (p-value < 0.02). Thus, with targeted features and distributional semantic features combined, we have reduced the error rate for `coord` by 3.0%, as opposed to 1.9% with targeted features only.

The choice of corpus turned out to be critical, with results often descending even below the baseline for the Wikipedia resource, and results for the Le Monde resource alone slightly lower than for the two combined. Confidence threshold differences were significant for the Wikipedia corpus on its own, with best results at a confidence threshold of 90%, but did not significantly affect results for the Le Monde corpus or the two combined.

When reviewing the results in detail, we see, as usual for robust statistical methods, a number of inexplicable corrections and errors, entirely unrelated to coordination. There are also a number of corrections for part-of-speech mismatches. These are understandable, since tokens with different parts-of-speech will generally have a similarity score of zero, whereas those with the same part-of-speech are more likely to have a positive score. However, they are not particularly interesting, as they ought to have been corrected by the targeted features. Finally, we also see a number of interesting cases that clearly illustrate the expected behaviour. For some of these cases, we show the results before the addition of semantic distributional features (underlined) and after their addition (in bold).

In the EstRépu corpus, these include:

Example 7.2 *Si les réformes envisagées par le gouvernement pour les* ***européennes*** *et les* ***régionales*** *sont adoptées telles qu'envisagées [...]*

Example 7.3 (...) *la loi favorise l'égal accès des femmes et des hommes aux* ***mandats*** *et* ***fonctions*** *électives.*

In the EMEA corpus, we have one surprising example, since it is highly unlikely that *mutagène* and *clastogène* were found as semantic neighbors.

Example 7.4 *Néanmoins, la bivalirudine ne s'est pas avérée* ***mutagène*** *ni* ***clastogène*** *dans les tests standards.*

We also have a much more reasonable example:

Example 7.5 *Ne pas utiliser Aclasta après la date de péremption mentionnée sur la* ***boîte*** *et le* ***flacon****.*

The following are cases from the Europarl corpus:

Example 7.6 *C'est pourquoi nous considérons que certaines* ***questions*** *soulevées par la baronne Ludford et certains* ***problèmes*** *soulevés par Mme Boumediene-Thiery [...]*

We also have a case of correction due to verbal similarity between *comprendre* and *marquer*, although the reasons behind this similarity are not easy to comprehend:

Example 7.7 *Ce pays n'est pas au centre de notre intérêt, mais il y a extrême urgence : ce Parlement l'a* ***compris*** *et a* ***marqué*** *son soutien.*

In the Wikipedia corpus, we have the following examples:

Example 7.8 *Il fut l'objet de plus de 500 millions de dollars de* ***commissions*** *et* ***rétrocommissions*** *qui ont nourri beaucoup de fantasmes [...]*

In the FTBDep dev corpus, we have the following clearly understandable error, where *crime* was considered more similar to *complices* than to *drogue*:

Example 7.9 (...) *aussi désarmé qu'on l'était au XIX siècle devant l'imagination des chevaliers d'industrie, l'entregent des* ***complices*** *de la drogue et du* ***crime*** *?*

We also have the following correction:

Example 7.10 *80000 francs (****indemnités*** *de changement de résidence et* ***allocation*** *de mobilité du conjoint comprises) .*

We close with the following example from the FTBDep test corpus:

Example 7.11 *En revanche, l'analyse des enchaînements cycliques nous a montré l'importance du* ***volume*** *d'épargne disponible et de son* ***degré*** *de liquidité dans les retournements cycliques.*

Although the results above are statistically significant, they are quite modest. Certain possibilities for improving them could be:

- Use a larger parsed corpus, such as FrWAC [Baroni et al., 2009]
- Construct the resources using different thresholds in terms of token count, productivity, and minimum similarity.
- Evaluate with different weighting methods, which can affect the similarity scores considerably. As already mentioned, the most common option is PMI (pointwise mutual information) which can be normalised in various manners [Bouma, 2009]. Anguiano [2013] attains better results for prepositional phrase attachment using a more context-dependent measurement he calls neighborhood-based relative frequency NRF. Mirroshandel et al. [2013] use a function that, like PMI, compares the count to the marginal counts of both the word and the predicate, but uses the direct ratios rather than the logarithm of the ratio.
- Evaluate with other similarity measures (e.g. cosine)
- Create a single feature combining the distributional similarity measures explicitly with other, stronger indicators, such as pos-tag matching (to avoid noun/preposition matches) and preposition matching (to avoid matching two different prepositions when the same preposition exists), so that similarity is only compared when stronger indicators do not exclude the comparison.
- Study the typical shift-reduce transition sequences for coordination carefully, and inject other features at critical decision points, e.g. if the system tends to reduce the correct 1st conjunct too early, so that only the wrong candidate is actually presented for coordination.

7.4 Discussion

In this chapter we have discussed several different methods for incorporating external resources into the parsing process. Three main areas were discussed: the use of specialised structured lexical resources, which group words together by their syntactic behavior; the use of large coverage lexicons giving a part-of-speech and morphological details for each word; and the incorporation of data attained directly from the parser's own analyses in semi-supervised approaches.

Of these, the first method is both the simplest and perhaps the most promising, since it allows us to construct word lists based on the observation of any particular behavior within our training corpus, to augment these lists with larger coverage resources built by others, and to apply them directly to a particular parsing problem. These have already been shown to be effective in the previous chapter, but we have only scratched the surface in terms of the available possibilities.

Experiments with replacing or extending the large-coverage lexicon have not been conclusive. Indeed, although the two lexicons reviewed were very different in size, their coverage of our evaluation corpora was quite similar. Since we used the identical lists of closed-class words in all tests, differences in the accuracy attained seem likely to depend more on the set of categories for each open class word form, and possibly on the listing of closed class word forms under open classes as well (e.g. *moi* listed as a common noun from its rare usage as

"*le moi*" in psychological contexts). Further exploration is required to understand the differences between the results between the two lexicons, in cases where they are significant, and especially to combine them effectively, since the corrections by each lexicon are orthogonal.

Both of the previously described methods do not take into account the relative frequency of each equivalence class or entry in actual corpora. This is one of the advantages of semi-supervised methods based on automatic corpus analysis: both in the case of unfiltered self-training, and in the case of automatically constructed resources, the frequency in the corpus is somehow brought to bear on the information fed back into the parser. However, while self-training has been shown to allow considerable gains for out-of-domain corpora, our own experiment with distributional semantic resources brought only modest gains. So far, we have only applied these resources to a very particular phenomenon: the semantic similarity (or lack thereof) between conjunct candidates in coordination. We have suggested various methods for attempting to improve on the results of this experiment. It would also be interesting to find ways to successfully use semantic clusters from distributional semantics to help parsing, which would have a much wider scope of applicability—but then we have to overcome the problem of cluster representativity within the training corpus.

Conclusion and perspectives

Overview

In this thesis, we have explored the robust statistical syntax analysis of French. Our goal, in so doing, has been to see to what extent a user can open up the black box of robust statistical methods to guide and hopefully improve the decision making process.

Another goal was to create a robust open-source tool capable of filling the gap left by Syntex in the Toulouse NLP landscape. From this point of view, Talismane has been undeniably successful: it is already being used in the CLLE-ERSS laboratory as a critical analysis component in a wide variety of daily NLP tasks. Its open source philosophy allows us to hope that it will open the door to collaboration with other laboratories in France and abroad.

Targeted users

Our stated goal is thus to enable the user to open up the black box. Ignoring for a moment the success or failure of our results, a first key question is, who is this user? Our initial aim was to open the system up to linguists, but is the profile of a typical linguist really adequate for performing the type of manipulation presented here? What type of knowledge is the user expected to master before he can safely proceed?

The answer may be found to some extent in the earlier chapters, in which we tried to provide all of the keys necessary to begin fiddling. These included, in chapter 1, a description of the dependency parsing scheme for French; in chapter 2, a high level description of robust statistical classification and supervised machine learning; in chapter 3, a description of how the two are combined in order to apply statistical machine learning classification to syntax analysis; and in chapter 4, a description of the baseline features and external resources needed to accomplish this task. At this point, it was assumed the reader would have an understanding of the core concepts needed to design new features and/or rules, and to imagine new ways in which external resources could be brought to bear on the syntax analysis process, be they corpora, lexicons or automatically constructed syntactic or semantic resources.

However, there are many basic skills assumed here that are not in the palette of the typical linguist, not the least of which is a thorough understanding of algorithms, and a sufficient understanding of probability and statistics to get by. Perhaps there are really three different types of users that need to be considered, each with their own needs: the machine learning guru, who is unabashed in the face of mindbogglingly complex statistics and mathematics, and for whom this thesis provides almost nothing of interest, except for an understanding of the particular case study of French syntax analysis. Outside this small circle of users stands the larger circle of computational linguists, who are eager to tweak the system in order to inject their knowledge and/or resources or test their hypotheses, but need sufficient understanding

of the underlying mechanics to do so: this is the user the current thesis most often aims to help. And finally, in the outermost circle, are the users of applications: users who want an out-of-the-box system to give them an adequate result for their specific need, without delving into how it works or why it doesn't work in a specific case. Although our thesis does not concentrate on evaluating Talismane within the context of applications, we have ensured that Talismane is made available as an easy-to-use out-of-the-box open source system, with a clear and detailed user manual. Our hope is that these users will now be able to take the matter into their own hands directly, and test Talismane in a variety of applicative contexts.

So, concentrating on the computational linguists in the middle circle, one of our key guiding principles in designing Talismane was to make the system as configurable as possible through command-line options and declarative text files, enabling these linguists to incorporate their knowledge and resources into the system without the help of a software engineer. In particular, we provided a feature definition syntax for defining declarative features and rules in configuration files. In hindsight, the goal of opening the system to linguists has not been entirely successful. The feature definition syntax, while highly expressive, becomes far too complex for anybody other than than a programmer as soon as we tackle complex features, while lacking the power of a true compiler. In fact, a programmer could write features far more efficiently in a compiled language such as Java, debug them more easily using modern development environments, and generate thousands of features in a single function, rather than having to analyse similar information thousands of time through declarative features. A better option would probably be to continue to support declarative features/rules, but also to allow an opening for compiled features and rules, improving the system's maintainability and its analysis speed. Indeed, feature analysis at a beam width of 1 takes up over half of the analysis time. At higher beams, it increases proportionally to the beam width. Pre-compiling very complex features may well help to improve this.

Furthermore, especially in the case of parser rules, it has proven difficult to anticipate the long-distance effects of a particular forced correction given the relative complexity of the shift-reduce algorithm's behaviour. Even when adding very specific pos-tagger features for a single function word, we observe seemingly random changes to other, completely unrelated words in sentences where the function word does not even appear. In some manners, robust statistical systems seem to behave in a chaotic manner, a bit like the "butterfly effect" coined by Edward Lorenz [Lorenz, 1963] in the context of atmospheric science, where a butterfly flapping its wings one way or another can determine whether or not a hurricane forms thousands of kilometers away and several weeks later. This is a direct result of the huge mathematical complexity of weights being assigned to thousands of interdependent features based on thousands of training occurrences. Nevertheless, we have shown that a systematic methodology to feature and rule testing can indeed result in some degree of improvement in the expected direction. Perhaps one of the most promising approaches would be to work in collaboration with linguists: the linguists would imagine various dependency attachment indicators, and myself (or another software engineer with sufficient mastery of the open source code and interfaces) would code these indicators as compiled features and test them.

In relation to this question, one area we didn't insist on enough in this thesis was the importance of manually annotating our own small evaluation corpus: FrWikiDisc, the French Wikipedia discussion pages. We did not, to our regret, attempt to capitalise on the differences between this genre and the journalistic training corpus in our tests using rules or through more advanced genre adaptation techniques. On the other hand, the very fact of manually annotating a corpus raised many dependency attachment questions, and forced us

to formalise the answers into dependency attachment indicators to be added to the annotation guide. Moreover, placing ourselves in a similar position to a machine learning system trying to annotate a text allowed us to gain a much better understanding of the nature of the ambiguities encountered by this system. A very long sentence is all the more easy to interpret if it contains surface level clues to guide the reader: clues such as parallel structures and appropriate punctuation. It is no wonder that the most successful features and rules capitalise on the same surface indicators to understand the structure of otherwise unwieldy sentences. I personally do not believe that quality work is possible in a machine learning task without first "delving into the data" and attempting manual annotation oneself.

Now, having discussed the difficulty of using declarative features and rules successfully, I do not wish to intimate that such configurable features and rules are completely useless. The toolkit provided seems perfectly adequate to enable a computational linguist to define surface rules. These would include, for example, lists of idiomatic expressions that need to be handled a certain way 99.9% of the time. For the tokeniser and pos-tagger, rules are fairly simple to write and understand, and straightforward to incorporate. The current syntax, however, gets very difficult to follow as soon as we tackle the transition-based parser, with a fairly simple surface rule stretching over 2 or 3 lines with a confusion of logical operators and parentheses. Research would be required into defining a more abstract, user-friendly syntax, allowing the user to define rules at a higher level of abstraction than the individual transitions of a shift-reduce parser. Perhaps trying to incorporate "deep" linguistic knowledge into a statistical system is doomed to failure from the outset, because the deeper we delve into the sentence structure, the more complicated it is to describe this structure in terms of surface manifestations, and the more likely it is for its surface manifestations to overlap with other phenomena, requiring semantic knowledge to differentiate them. In the latter case, the statistical system is unlikely to draw any definite conclusions favoring one decision over another: the information, if introduced as a feature, would almost certainly be drowned by the mass of low-information-content features; if introduced as a rule, it would most certainly be sufficiently complex as to introduce at least as many errors as it corrected.

It is important in any case to keep in mind the limits of robust statistical systems, and use these systems where they are most effective. With a labeled accuracy hovering between 87 and 93%, such systems cannot be expected to deal well with rare or complex phenomena. Keep in mind also that given the current state-of-the-art in annotation, the inter-annotator agreement scores often hover around the same level, so that it is not clear how much farther these systems can really be improved. They are best suited for applications which require a large critical mass of parsed data, and where the quantity of the data at a reasonable quality can drown out any noise introduced by the inevitable parsing errors. There are certain tricks of the trade for improving the quality of the data produced: we introduced one such possibility through the use of parser confidence cutoffs to include only those links where the parser is fairly sure of its decision, and attempted to apply it to the automatic building of semantic resources.

Targeted applications

This brings us to the types of applications to which statistical syntax analysis is indeed well suited: the construction of distributional semantic resources being one such example, where data mass is critical for success, and parse errors are hidden unless the identical error is repeated often. Another application which was mentioned earlier in the thesis is authorship

attribution: trying to recognise an author's style from the type of constructs he uses more often, among other information. Yet another application where robust statistical parsing can be useful was the original driving force for this thesis: terminology extraction.

Recall from the introduction that I first embarked upon this thesis after the French Space Agency had asked me for a terminology extraction tool. As I write now, the Talismane terminology extractor is built and is being used by space terminologists to construct their knowledge bases. The use of robust statistical parsing seems justified, because it allows us to analyse very large corpora, and only extract those terms which occur a sufficient number of times—in other words, while parsing errors may well remove a few instances of a candidate term here and there or introduce an incorrect candidate term elsewhere, they are unlikely to do this systematically. Initially, I had hoped to include an evaluation of the Talismane terminology extractor in the present thesis: but time constraints have forced me to lessen my ambitions. This is definitely one of the immediate future perspectives. Many other questions require exploration around this task: how well will a parser trained on a journalistic corpus bear when evaluated against a technical corpus for space studies? And what methods can be used to improve parsing for a specific corpus, genre or application? I have mentioned some studies in domain adaptation in chapter 7, but have not yet explored the key issues of domain adaptation in the present thesis, despite the availability of evaluation corpora including a corpus from the medical domain, thanks to the Sequoia project [Candito et al., 2012]. I have not at all mentioned work in application adaptation, but there are some very interesting recent advances which would be worth exploring. For example, Hall et al. [2011] use global learning through perceptrons in a way that makes it possible to optimise in parallel against a training corpus (the intrinsic objective) and against any external measurement of success (the extrinsic objective). The only constraint on the extrinsic objective is that, given some sentence of its choice, it must be able to identify the single best parse among the k currently most probable parses of this sentence. Could such a method be used to train a parser specifically for terminology extraction?

Results and methodology

I return to the results in the present thesis. One of the goals of chapter 5 was to get the choice of machine learning algorithm and configuration out of the way, so that we could concentrate on more interesting linguistic questions. However, as it turned out, the improvements presented in this chapter far outweigh results presented in the later chapters in terms of overall accuracy gain. The choice of classifier, for example, turned out to be critical in terms of the parser, with a correctly configured linear SVM classifier consistently outperforming the other classifiers by approximately 1%. The beam width and beam propagation between the pos-tagger and parser also have a considerable global effect, with considerable global gains in the lower beams: 0.5% or so at beam 2, and an additional 0.3% or so at beam 5, at the cost of slower parsing speed.

The linguistic methods presented in later chapters provide, in contrast, a much more modest gain. However, these methods have to be placed in perspective: our goal was not to achieve global gains in all areas of parsing, but rather to prove the concept of targeted features and rules for one particular phenomenon. For example, in chapter 6, when tackling the word *que*, a combination of pos-tagger features and rules allowed us to decrease the remaining error count by 49%. It can be hoped that similar efforts for other function words would yield similar results. On the other hand, the more complex features, especially those regarding

coordination, although their coding is extremely satisfying from an intellectual point of view, yield little improvement in parsing accuracy, reducing the initial number of coordination parse errors of 1687 by 33 only (a 1.9% improvement). The distributional semantic method for attempting to recognise conjuncts on the assumption that they are semantically closer than other candidates also produced fairly disappointing results: given the immense mechanics required to parse large corpora, construct the semantic resources, and apply the data during the parsing process in the form of complex features, the paltry improvement of only 40 new corrections for 20 new errors has to be taken with a bit of humor. We are not yet, for all that, ready to abandon ship: further tests are required to see if targeted coordination features, or else coordination rules, with or without the help of semantics can somehow be made more effective. Given the amount of room there is for improvement, we are confident that better results could be attained with a bit of concentrated effort. Perhaps we need to rethink our philosophy on rules, and rather than trying to find rules that work systematically, try to find rules that work often enough to improve considerably on the unassisted parsing performance.

Indeed, there is only so much improvement that can be attained through purely statistical methods, without generating new reliable external resources, or annotating much larger training corpora, and possibly embarking on a correction campaign for existing training material. For instance, early on in the thesis, considerable work was put into constructing a probabilistic tokeniser. The amount of effort required seems unjustified in view of the results: given that ambiguous tokens typically have one preponderant solution and another very rare solution, a probabilistic system is unlikely to favor the rare solution except on very strong indicators, and we were unable to find these with the limited amount of information available to the tokeniser. Here again, a better solution might have been to use a deterministic tokeniser, possibly augmented with surface rules. But in so saying, we are moving away from a purely statistical approach, and more towards a mixed statistical and symbolic approach: and considerable work has already been put into symbolic approaches for French and other languages through parsers such as Fips [Wehrli, 2007] or FRMG [De La Clergerie et al., 2009]. The advantages of symbolic approaches are obvious in their ability to capture knowledge of the language in a formal manner, but with this comes a lot of baggage: the difficulty of maintaining a huge number of symbolic rules, the tendency to code only "correct" usage as opposed to real usage, and lower overall robustness as it was defined for this thesis, in terms of speed and the ability to cope with unexpected input. Perhaps systems such as Talismane open the way to a middle ground: a strong statistical base that can grow stronger as more training material and higher quality/coverage external resources becomes available, combined with symbolic tweaks to help the system where empirical results show it to be weakest. But can these tweaks ever hope to do more than a minuscule correction of a very specific error type in very specific circumstances?

The question of reliable external resources is a critical one. In French, we are lacking freely available high-quality and high coverage resources for semantics. The existence of resources such as Dicovalence, giving verbal sub-categorisation framse, are certainly interesting, and further experimentation is required to attempt to integrate them with Talismane. However, more quality resources are required around selectional preferences of verbs especially, both in terms of lexical and semantic preferences. I personally feel that such resources would go far into enabling us to write better statistical parsers, incorporated either as features or as some variant of rules. We attempted some experiments with lists of verbs that sub-categorise with *que* as a direct object in section 6.2, with mitigated success. We only looked at the verb itself, ignoring clitics and fixed noun complements (e.g. "*se rendre compte que*"). It

would be interesting to experiment with a much more detailed list for this type of preference to see if better results can be achieved, either in features or in rules. Still, even the most carefully constructed manual resource cannot take into account critical information regarding frequencies: information that is only possible in resources automatically constructed from a given corpus. To give a trivial example from pos-tagging: a lexicon may tell us that *cela* is the simple past of the verb *celer*, but if we don't accompany this knowledge by the fact that 99.9% of the time it is a demonstrative pronoun, the information is only partially useful, and could easily induce errors.

So, in terms of the overall results, the methods we have presented here—and especially the linguistic methods—admittedly provide only a very modest gain. On the positive side, the methodology presented for testing targeted features and rules has already provided interesting results: both in terms of highlighting targeted errors in the training or development corpora, and in terms of enabling us to view the specific cases where an indicator which we thought would be a strong disambiguating mark for a given decision resulted in a different decision than the one expected. This allows us to explore the richness of the language, as we hone in on the information that isolates a target decision. We have found Talismane to be a system that begs to be explored further: our hope is that, with time, we will manage to improve the methodology to the point where we improve results more radically in the final evaluation. The intellectual curiosity and satisfaction made possible by our methods is one of the most positive aspects of this thesis, and indeed, inspires us to imagine many new, different scenarios to explore. This brings us to the our initial list of future perspectives.

Future perspectives

Remaining within the paradigm presented in the current thesis, we would like to push the methodology presented even further, and see if we cannot get better results for several target problems, including function word pos-tagging for many different ambiguous function words, and targeted features/rules for coordination as well as prespositional phrase attachment. For prespositional phrase attachment, we would like to test existing external resources for subcategorisation frames such as Dicovalence, as well as building richer resources that include nominal or adverbial complements, combined with automatically constructed lexical affinity scores (for nouns) or verbal attachment propensity scores (for prepositions). Similarly, there are several groups of words (e.g. time expressions) which display very specific behavior in terms of dependency attachment labels: we would like to construct such groups and see to what extent we can correct parsing for these groups through features and/or rules.

In terms of the transition-based parsing mechanism, there are several points we would like to explore more deeply. One of the first is a new way of applying rules through constrained dependencies. We have seen above that trying to write rules around specific transitions is cumbersome. Moreover, the rules only kick in if the parse configuration follows a path that leads to the situation envisaged. If we want to compare two items on the stack to see which is more likely to attach with a preposition on the buffer, we are assuming both items are still on the stack—but the system may well have reduced one of the items prior to reaching the preposition, so that the rule cannot be applied. An alternative approach would be to analyse certain aspects of the sentence ahead of time, and add constraints around the inclusion or exclusion of certain dependencies. We then need to develop an alternate transition system that can take these constraints into account, and refuse to apply a transition that violates a

constraint or makes it impossible to meet it at a later stage. Certainly, we lose out on the information relating to partial dependency trees already constructed—but this information is not required for all rules, especially not for surface rules. This type of system would also allow us to apply a similar experiment to Mirroshandel et al. [2012] to the case of transition-based parsing, with linear complexity.

The idea of automatically constructed word clusters has been explored by Candito and Crabbé [2009] for domain adaptation with mitigated results: the gains shown are equivalent to those achieved by self-training. The idea of using automatically generated semantic clusters is tempting, but there are several problems from a semantic point of view with such clusters: how do we handle polysemy, and how do we handle figurative versus literal uses of a word, especially when it may be used with figurative preponderance in some corpora, and literal preponderance in others. We have attempted one experiment in this thesis using distributional semantic resources. However, we have not yet been successful in semantic word clustering approaches based on these resources, and so we have only tackled an area where the resource could be used independently of semantic clusters: the semantic proximity of potential conjuncts. However, we feel there is a lot of potential in these automatically constructed resources, both for semantics and lexical affinity/selectional preferences, especially since they can take into account frequency in their scores. We plan to continue attempts at successfully incorporating these resources into transition-based parsing, especially in the area of out-of-domain parsing, where rules/constraints may well be much more applicable than features, the latter being particularly difficult to apply out-of-domain, since they are tied closely to the training corpus material.

Beyond these perspectives for annotated corpus evaluation presented in the present thesis, one of the main areas we have not yet explored is the evaluation and tuning of our parser in view of its results for specific applications. We are especially interested in terminology extraction: both inasmuch as it concerns out-of-genre and out-of-domain parsing, and inasmuch as the constructs which interest us for terminology extraction are not the same as those used for, say, question answering. We are rarely if ever interested in long-distance relationships or in non-contiguous terms. Thus, a transition-based parser, which is fairly good at guessing short-distance dependencies, may be the ideal candidate for such a task. Given the corpora already analysed via Talismane at the French Space Agency, and the terms that have been selected after automatic extraction and manual revision, it should be fairly straightforward to construct an evaluation set.

Finally, we would like to see to what extent our targeted features and rules can work for languages other than French. We are interested in minority languages, such as Occitan and Yiddish, through our work with OCR on the Jochre project. It would be interesting to see to what extent Talismane can be adapated to languages with even less resources than French. Then of course, there is the case of English: it would be useful to adapt Talismane to English not only for comparison with the vast body of work concerning the parsing of the Penn Treebank corpus, but also to begin to experiment with methods for bilingual terminology extraction, and see if parsing for the two languages in parallel can help extract better terms.

Closing remarks

This thesis was as much a question of personal career development as of scientific research. As a freelance software engineer, I wished to develop expertise in areas which would allow

me to combine my interest in languages and algorithms, and veer my working life towards areas that would use my mental faculties to their full capacity. This has to a large extent already been accomplished: whereas my contracts at the start of the thesis were mostly in project management for large intranet web sites managing company databases, the vast majority of my contracts nowadays are in natural language processing. My thesis being entirely unfinanced, it has been a constant struggle to find enough time for the research and development directly related to the thesis, all the while having to earn a living and keep customers satisfied. Bringing my contrasts closer in line with my thesis work has helped immensely.

Nevertheless, the questions raised by this thesis, and in particular those related to incorporating linguistic knowledge into robust statistical methods for parsing, have become central to my thinking, far beyond the concerns of career development. The prospects opened by my plunge into the world of research are fascinating, and it is my hope that I will find the opportunity to continue to collaborate with research laboratories in the future around the various future perspectives discussed here, as well as many other, unimagined, areas of research surrounding syntax analysis. The improved state-of-the-art in robust parsing is undeniable, made possible by the availability of annotated corpora such as the French Treebank, high-quality external resources such as the LeFFF or GLÀFF, and improved parsing technology and methodology. It is only a question of time before parsed results will be fully usable in applications such as statistical translation or statistical multilingual terminology extraction. It has been a privilege to be able to play a small part in this adventure.

Appendix A

Evaluation graphs

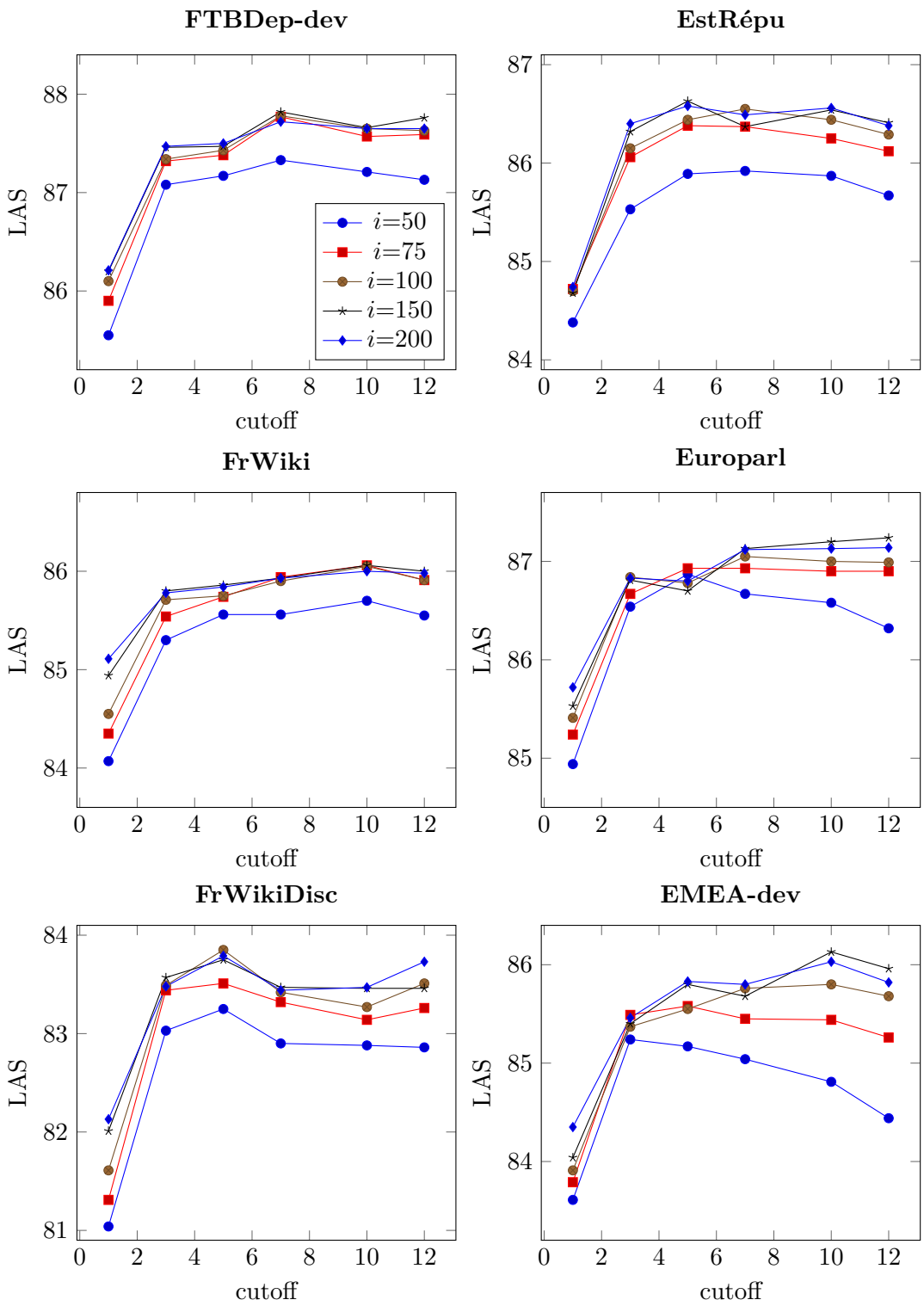


Figure A.1: Parser evaluation corpora LAS for a MaxEnt classifier using different values for iterations i and cutoff

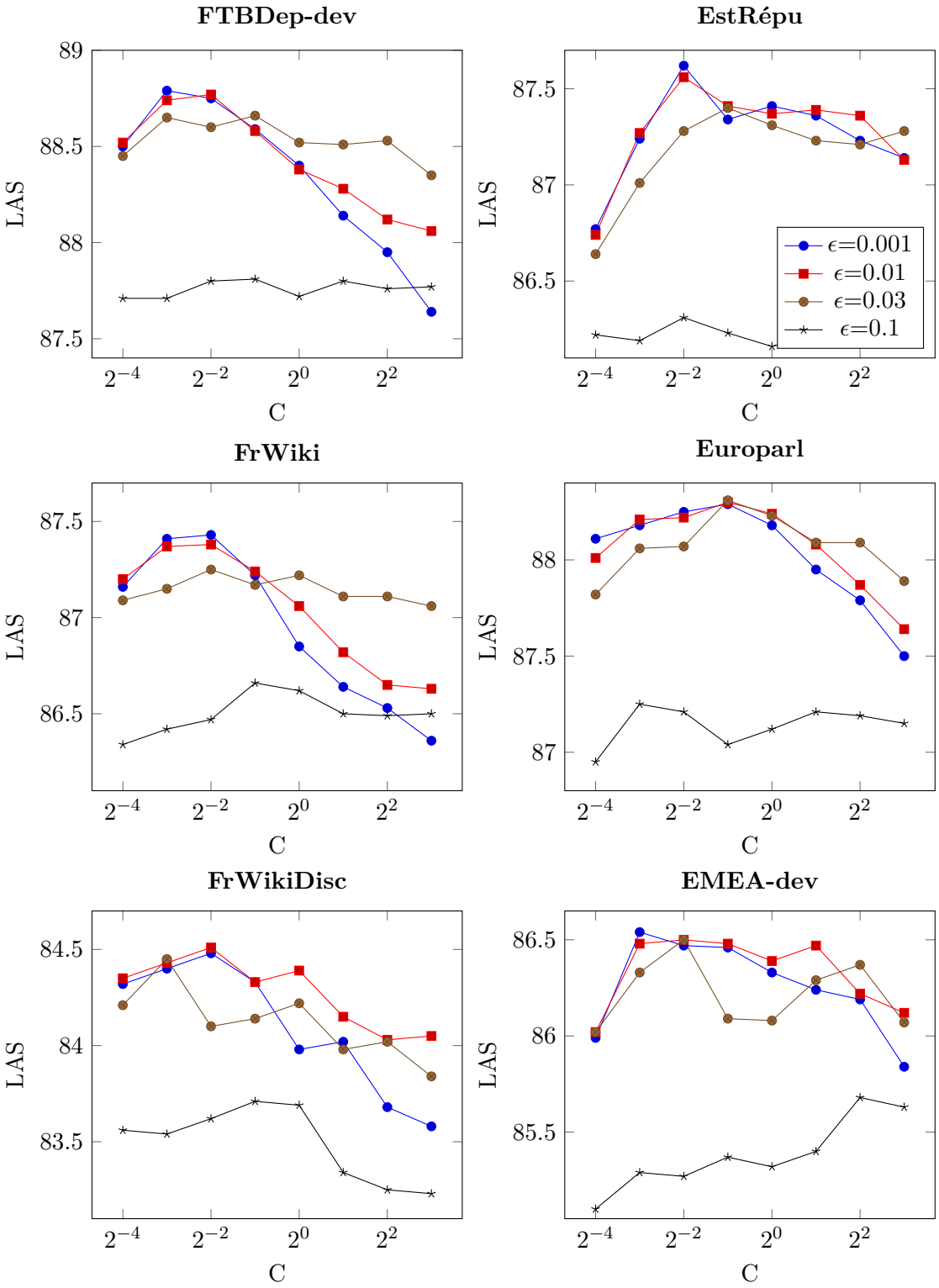


Figure A.2: Parser evaluation corpora LAS for a linear SVM using different values of C and ϵ

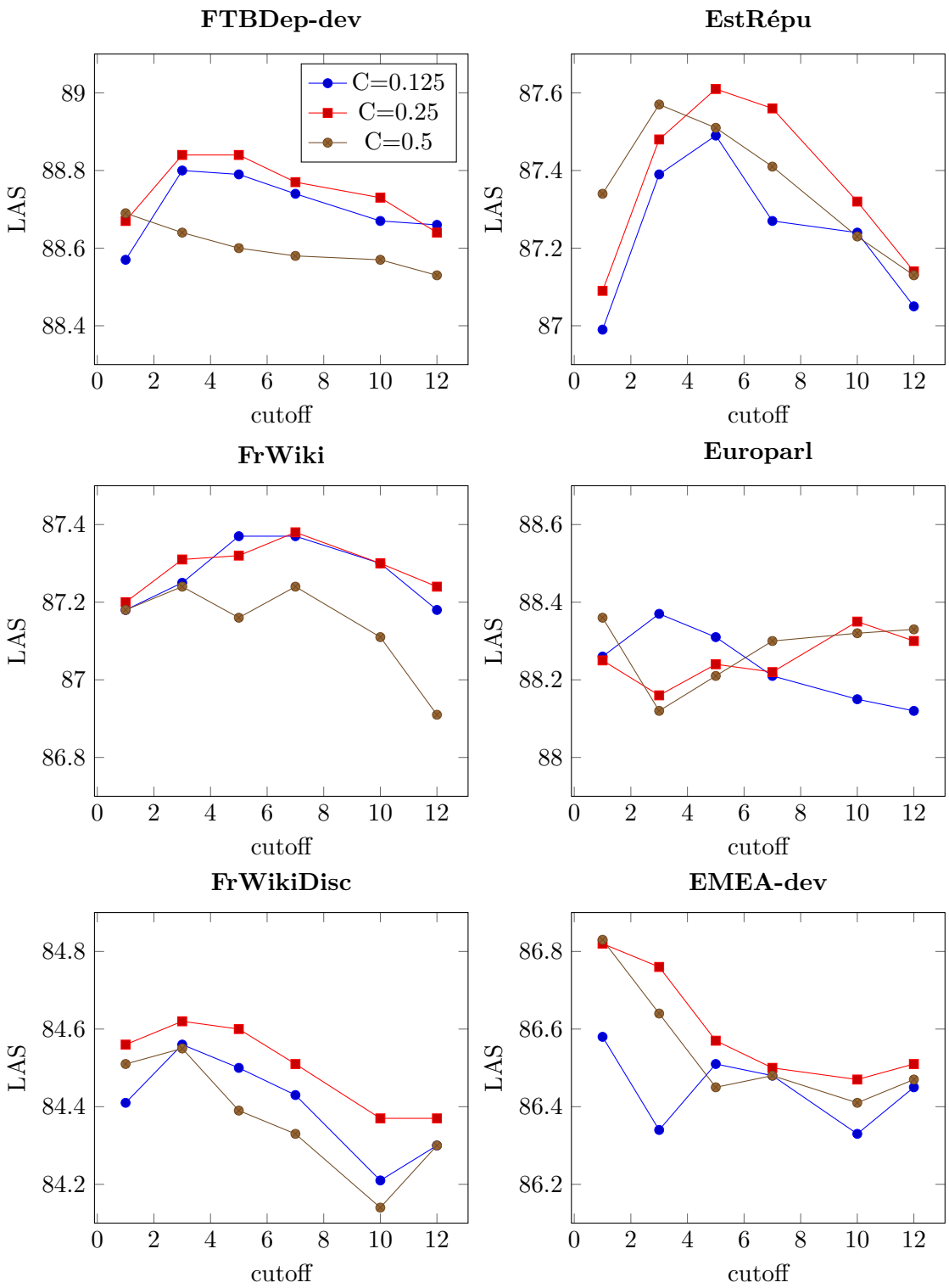


Figure A.3: Parser evaluation corpora LAS for a linear SVM using different values of C and cutoff

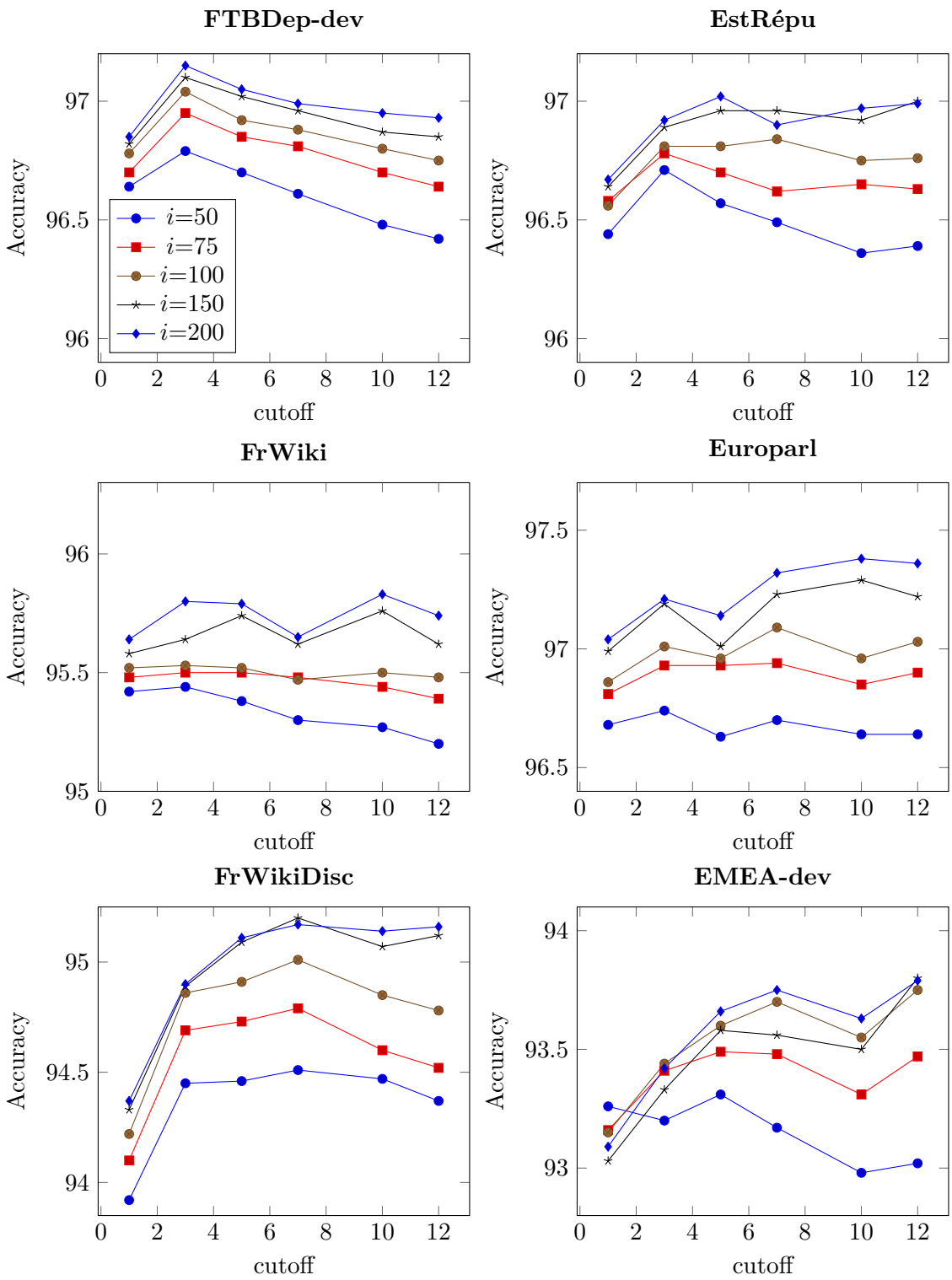


Figure A.4: Pos-tagger evaluation corpora accuracy for a MaxEnt classifier using different values for iterations i and cutoff

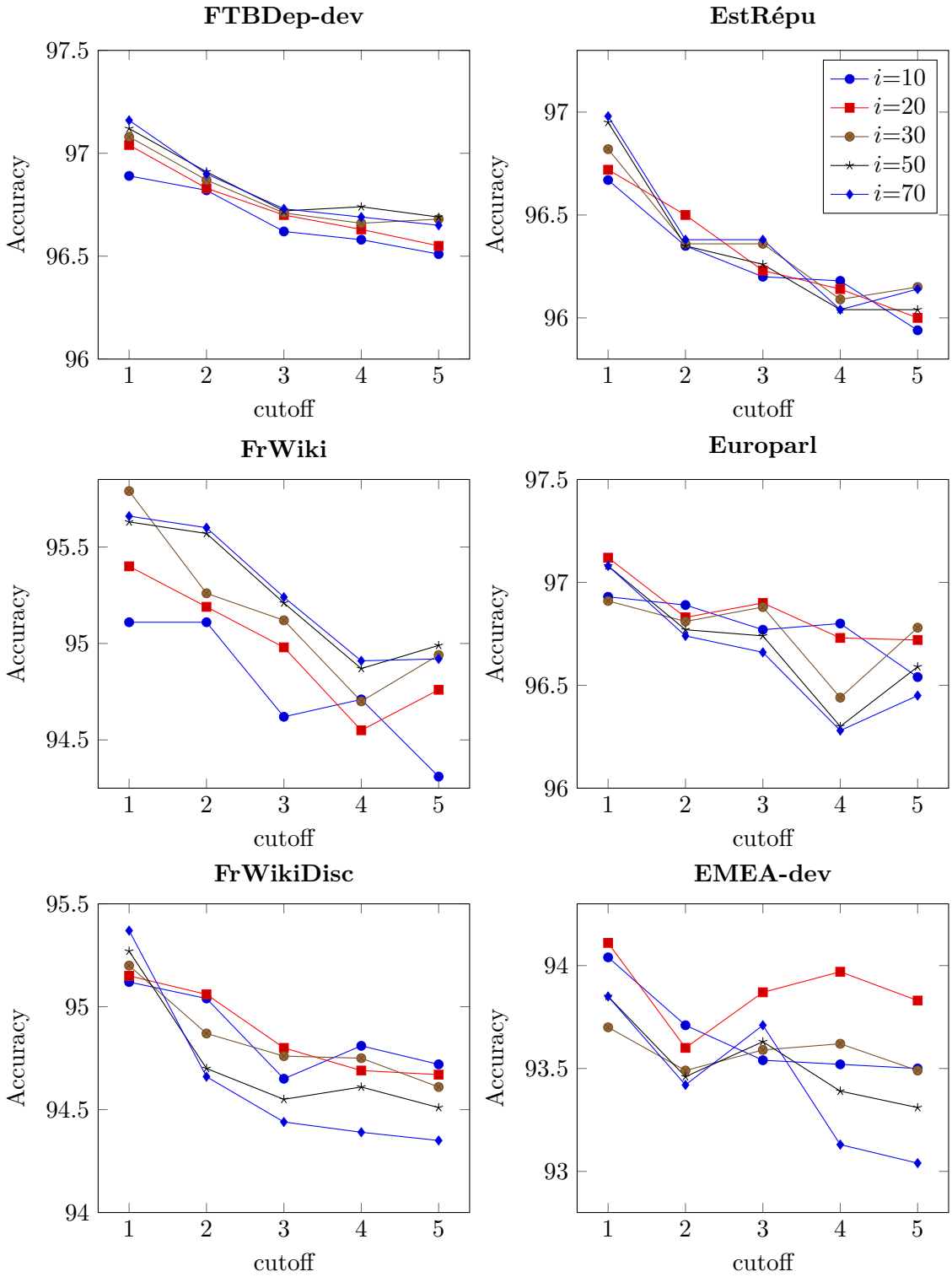


Figure A.5: Pos-tagger evaluation corpora accuracy for a perceptron classifier using different values for iterations i and cutoff

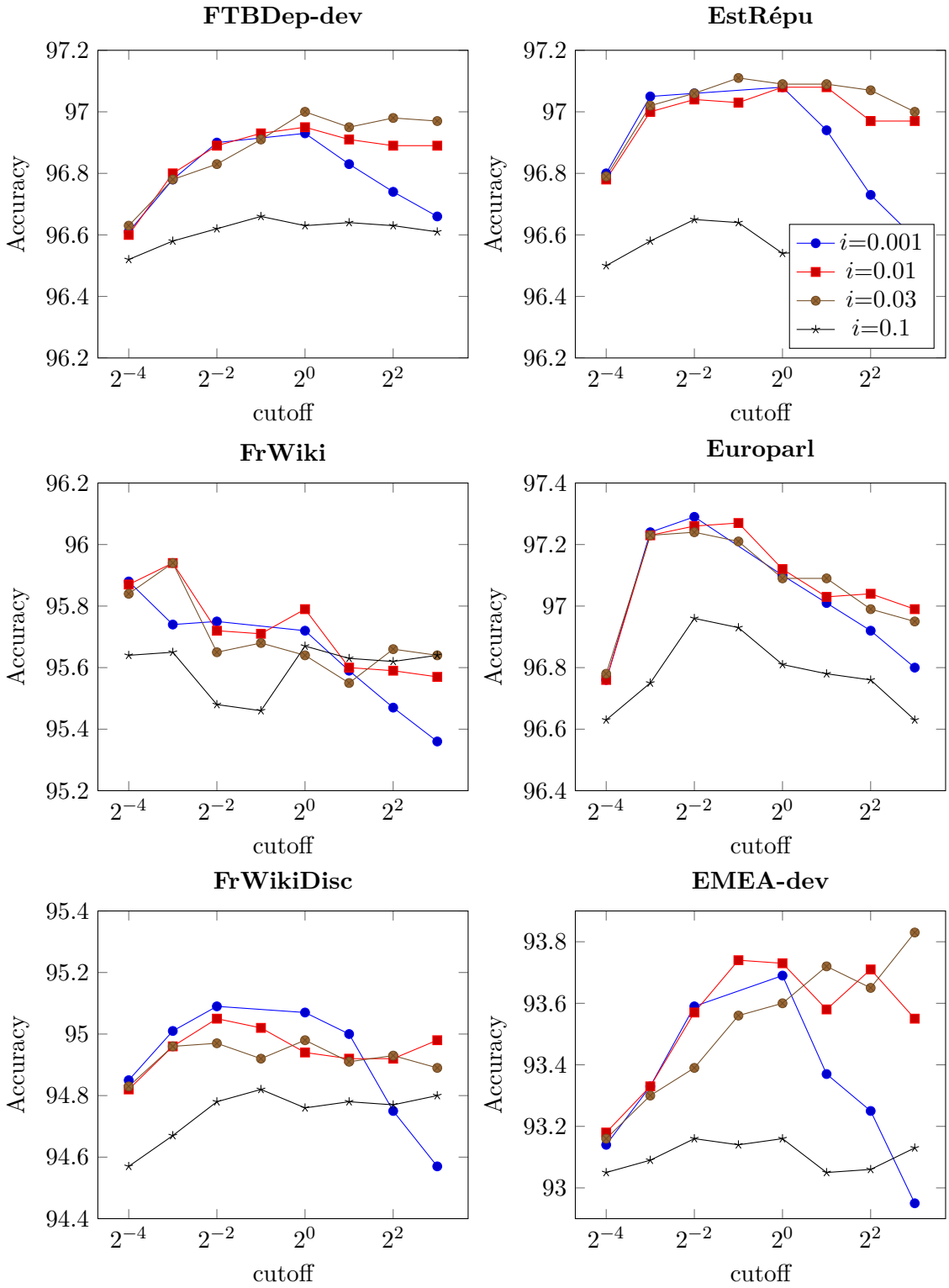


Figure A.6: Pos-tagger evaluation corpora LAS for a linear SVM using different values of C and ϵ

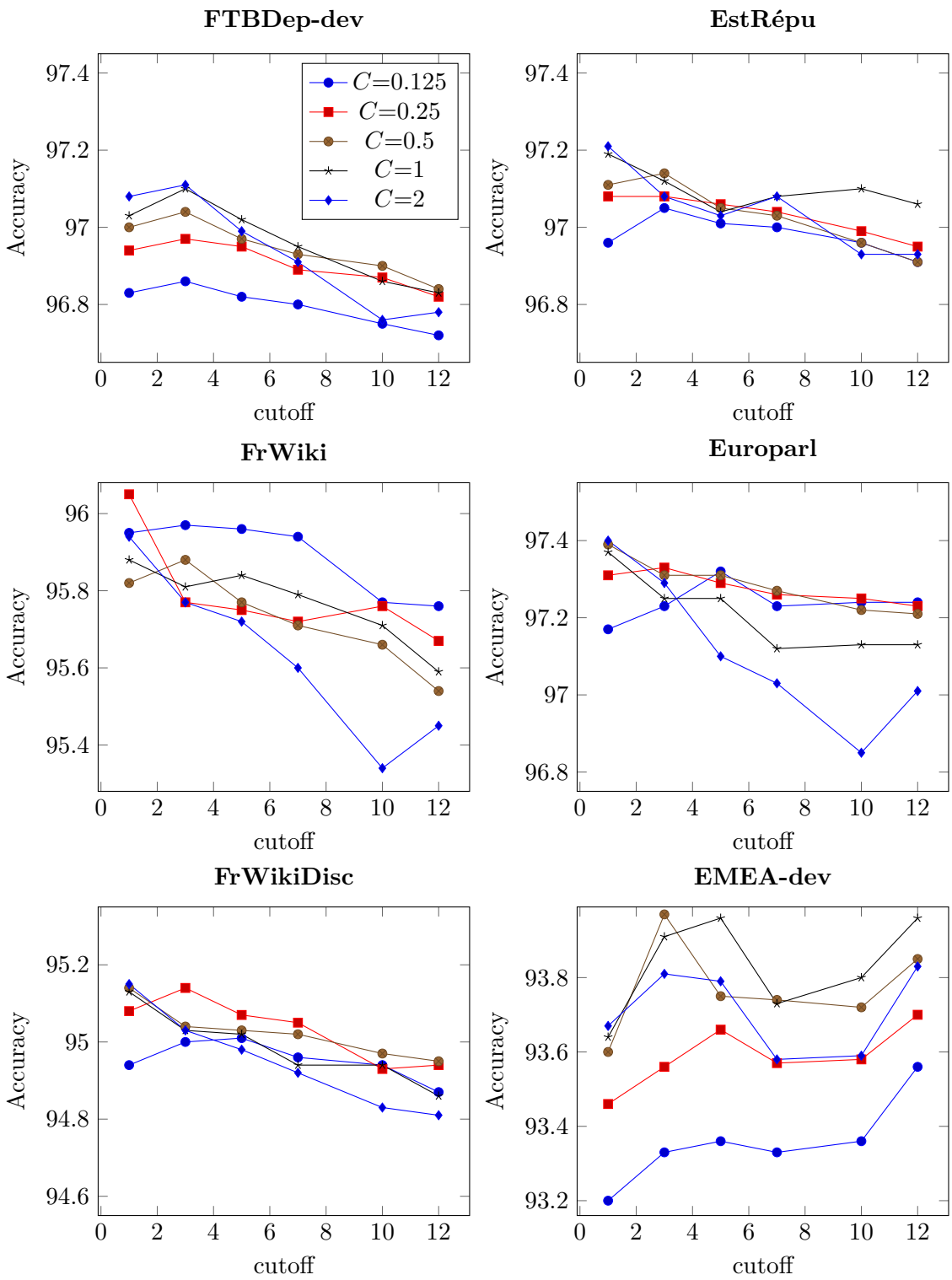


Figure A.7: Pos-tagger evaluation corpora LAS for a linear SVM using different values of C and cutff

Bibliography

Anne Abeillé. Guide des annotateurs - annotation fonctionnelle, March 2004. URL `http://www.llf.cnrs.fr/Gens/Abeille/guide-fonctions.new.pdf`.

Anne Abeillé, Lionel Clément, and François Toussenel. Building a treebank for French. In Anne Abeillé, editor, *Treebanks*. Kluwer, 2003.

Anne Abeillé, François Toussenel, and Martine Chéradame. Corpus le monde - annotations en constituants - guide pour les correcteurs, March 2004. URL `http://www.llf.cnrs.fr/Gens/Abeille/guide-annot.pdf`.

Anne Abeillé and Nicolas Barrier. Enriching a french treebank. In *LREC*, Lisbon, Portugal, 2004.

Anne Abeillé and Lionel Clément. Annotation morpho-syntaxique, 2006. URL `http://llf.linguist.jussieu.fr/llf/Gens/Abeille/guide-morpho-synt.06.pdf`.

Clémentine Adam. *Voisinage lexical pour l'analyse du discours*. PhD thesis, Université Toulouse le Mirail-Toulouse II, 2012.

Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2004.

Enrique Henestroza Anguiano. *Efficient Large-Context Dependency Parsing and Correction with Distributional Lexical Resources*. PhD thesis, Université Paris Diderot, 2013.

Marco Baroni, Silvia Bernardini, Adriano Ferraresi, and Eros Zanchetta. The wacky wide web: a collection of very large linguistically processed web-crawled corpora. *Language resources and evaluation*, 43(3):209–226, 2009.

Solon Beinfeld and Harry Bochner. *Comprehensive Yiddish-English Dictionary*. Indiana University Press, 2013.

Roberto Bisiani. Beam search. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence, 2nd edition*, pages 1467–1468. Wiley-Interscience, 1987.

Bernd Bohnet and Joakim Nivre. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465. Association for Computational Linguistics, 2012.

Gerlof Bouma. Normalized (pointwise) mutual information in collocation extraction. In *Proceedings of the Biennial GSCL Conference*, pages 31–40, 2009.

Didier Bourigault. Upery: un outil d'analyse distributionnelle étendue pour la construction d'ontologies à partir de corpus. In *Actes de la 9ème conférence annuelle sur le Traitement Automatique des Langues (TALN 2002), Nancy*, pages 75–84. ATALA, 2002.

Didier Bourigault. *Un analyseur syntaxique opérationnel: SYNTEX*. Habilitation à diriger des recherches en linguistique, Université de Toulouse le Mirail-Toulouse II, 2007.

Myriam Bras. Le projet teloc: construction d'une base textuelle occitane. *Langues et Cité: bulletin de l'observation des pratiques linguistiques*, 8(9), 2006.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

Marie Candito and Benoît Crabbé. Improving generative statistical parsing with semi-supervised word clustering. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 138–141. Association for Computational Linguistics, 2009.

Marie Candito and Djamé Seddah. Effectively long-distance dependencies in french: annotation and parsing evaluation. In *TLT 11-The 11th International Workshop on Treebanks and Linguistic Theories*, 2012.

Marie Candito, Benoît Crabbé, and Pascal Denis. Statistical french dependency parsing: treebank conversion and first results. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010)*, pages 1840–1847, 2010a.

Marie Candito, Joakim Nivre, Pascal Denis, and Enrique Henestroza Anguiano. Benchmarking of statistical dependency parsers for french. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 108–116. Association for Computational Linguistics, 2010b.

Marie Candito, Enrique Henestroza Anguiano, and Djamé Seddah. A word clustering approach to domain adaptation: Effective parsing of biomedical texts. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 37–42. Association for Computational Linguistics, 2011a.

Marie Candito, Benoît Crabbé, and Mathieu Falco. Dépendances syntaxiques de surface pour le français, May 2011b. URL `http://alpage.inria.fr/statgram/frdep/Publications/FTB-GuideDepSurface.pdf`.

Marie Candito, Djamé Seddah, et al. Le corpus sequoia: annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical. In *TALN 2012-19e conférence sur le Traitement Automatique des Langues Naturelles*, 2012.

Jean Carletta. Assessing agreement on classification tasks: the kappa statistic. *Computational linguistics*, 22(2):249–254, 1996.

Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics, 2000.

Noam Chomsky. *Syntactic structures.* Mouton & co., The Hague/Paris, 1957.

Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

Kenneth Church. A pendulum swung too far. *Linguistic Issues in Language Technology*, 6, 2011.

Michael Collins. *Head-driven statistical methods for natural language parsing.* PhD thesis, University of Pennsylvannia, 1999.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

Michael Collins. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.

Marcel Cori and Jacqueline Léon. La constitution du TAL : Étude historique des dénominations et des concepts. *TAL*, 43(3):21–55, 2002.

Benoit Crabbé and Marie Candito. Expériences d'analyses syntaxique statistique du français. In *TALN 2008- conférence sur le Traitement Automatique des Langues Naturelles.* ATALA, 2008.

J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480, 1972.

Éric De La Clergerie, Benoît Sagot, Lionel Nicolas, Marie-Laure Guénot, et al. Frmg: évolutions d'un analyseur syntaxique tag du français. In *Journée de l'ATALA sur: Quels analyseurs syntaxiques pour le français?* ATALA, 2009.

Pascal Denis and Benoît Sagot. Coupling an annotated corpus and a lexicon for state-of-the-art pos tagging. *Language Resources and Evaluation*, 46(4):721–736, 2012.

Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

Jason M Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics, 1996.

Cécile Fabre. *Affinités syntaxiques et sémantiques entre mots: apports mutuels de la linguistique et du TAL.* Habilitation à diriger des recherches en linguistique, Université de Toulouse le Mirail-Toulouse II, 2010.

Cécile Fabre, Josette Rebeyrolle, Lydia-Mai Ho-Dac, et al. Examen du statut des syntagmes prépositionnels à la lumière de données issues de corpus annotés. *CMLF 2008*, pages 2484–2494, 2008.

C. Fellbaum, editor. *WordNet: An Electronic Lexical Database.* The MIT Press, 1998.

John Rupert Firth. A synopsis of linguistic theory, 1930-1955. In *Studies in Linguistic Analysis*, pages 1–32. Blackwell, Oxford, 1957.

Yoav Goldberg and Jon Orwant. A dataset of syntactic-ngrams over time from a very large corpus of english books. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task*, pages 241—-247. Association for Computational Linguistics, June 2013.

Keith Hall, Ryan McDonald, Jason Katz-Brown, and Michael Ringgaard. Training dependency parsers by jointly optimizing multiple objectives. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1489–1499. Association for Computational Linguistics, 2011.

Zellig Harris. Distributional structure. *Word*, 10(23):146–162, 1954.

Chia-Hua Ho and Chih-Jen Lin. Large-scale linear support vector regression. *Journal of Machine Learning Research*, 13:3323–3348, 2012.

Marie-Paule Jacques. Que : la valse des étiquettes. In *Actes de la 12ème conférence sur le Traitement Automatique des Langues Naturelles (TALN'2005)*, pages 133–142, Dourdan, France, 2005.

Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

Richard Johansson and Pierre Nugues. Investigating multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 206–210. Association for Computational Linguistics, 2006.

Richard Johansson and Pierre Nugues. Incremental dependency parsing using online learning. *Proceedings of the CoNLL/EMNLP*, pages 1134–1138, 2007.

John Judge, Aoife Cahill, and Josef Van Genabith. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 497–504. Association for Computational Linguistics, 2006.

Daisuke Kawahara and Kiyotaka Uchimoto. Learning reliability of parses for domain adaptation of dependency parsing. *IJCNLP'08*, 2008.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. *Dependency parsing*. Morgan & Claypool Publishers, 2009.

Dekang Lin. An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304, 1998.

Edward N Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20 (2):130–141, 1963.

David McClosky, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 337–344. Association for Computational Linguistics, 2006.

David McClosky, Eugene Charniak, and Mark Johnson. When is self-training effective for parsing? In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 561–568. Association for Computational Linguistics, 2008.

Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, volume 6, pages 81–88, 2006.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Marvin Minsky and Papert Seymour. Perceptrons. 1969.

Seyed Abolghasem Mirroshandel, Alexis Nasr, and Joseph Le Roux. Semi-supervised dependency parsing using lexical affinities. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 777–785. Association for Computational Linguistics, 2012.

Seyed Abolghasem Mirroshandel, Alexis Nasr, and Benoıt Sagot. Enforcing subcategorization constraints in a parser using sub-parses recombining. In *Proceedings of NAACL-HLT*, pages 239–247, 2013.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. The MIT Press, 2012.

François Morlane-Hondère. *Une approche linguistique de l'évaluation des ressources extraites par analyse distributionnelle automatique*. PhD thesis, Université Toulouse le Mirail-Toulouse II, 2013.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

Yitskhok Niborski and Bernard Vaisbrot. *Yidish–frantseyzish verterbukh / Dictionnaire yiddish–français*. Bibliothèque Medem, Paris, 2002.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan T. McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The conll 2007 shared task on dependency parsing. In *EMNLP-CoNLL*, pages 915–932. Association for Computational Linguistics, 2007a.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kubler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95, 2007b.

Patrick Paroubek, Isabelle Robba, Anne Vilnat, and Christelle Ayache. Data, annotations and measures in easy, the evaluation campaign for parsers of french. In *In proceedings of the fifth international conference on Language Resources and Evaluation (LREC 2006)*, pages 315–320, 2006.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics, 2006.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 705–713. Association for Computational Linguistics, 2010.

Adwait Ratnaparkhi. *Maximum entropy models for natural language ambiguity resolution.* PhD thesis, University of Pennsylvania, 1998.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Kenji Sagae. Self-training without reranking for parser domain adaptation and its impact on semantic role labeling. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 37–44. Association for Computational Linguistics, 2010.

Kenji Sagae and Alon Lavie. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 691–698. Association for Computational Linguistics, 2006.

Kenji Sagae and Jun'ichi Tsujii. Dependency parsing and domain adaptation with lr models and parser ensembles. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1044–1050, 2007.

Benoît Sagot. The lefff, a freely available and large-coverage morphological and syntactic lexicon for french. In *7th international conference on Language Resources and Evaluation (LREC 2010)*, 2010.

Benoît Sagot, Lionel Clément, Eric de La Clergerie, Pierre Boullier, et al. The lefff 2 syntactic lexicon for french: architecture, acquisition, use. In *LREC*, pages 1–4, 2006.

Franck Sajous, Nabil Hathout, and Basilio Calderone. Glàff, un gros lexique à tout faire du français. In *Actes de la 20e conférence sur le Traitement Automatique des Langues Naturelles (TALN'2013)*, pages 285–298, Les Sables d'Olonne, France, 2013.

John Sinclair. Chapter 1: Corpus and text — basic principles. In Martin Wynne, editor, *Developing linguistic corpora: a guide to good practice.* Oxbow Books, 2005.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. Brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association*

for Computational Linguistics, pages 102–107. Association for Computational Linguistics, 2012.

Ludovic Tanguy. *Complexification des données et des techniques en linguistique: contributions du TAL aux solutions et aux problèmes.* Habilitation à diriger des recherches en linguistique, Université de Toulouse le Mirail-Toulouse II, 2012.

Ludovic Tanguy and Nabil Hathout. Webaffix: un outil d'acquisition morphologique dérivationnelle à partir du web. *Actes de TALN'02*, 2002.

Ludovic Tanguy, Assaf Urieli, Basilio Calderone, Nabil Hathout, and Franck Sajous. A multitude of linguistically-rich features for authorship attribution. In *Notebook for PAN at CLEF 2011*, Amsterdam, 2011.

Robert F Tate. Correlation between a discrete and a continuous variable. point-biserial correlation. *The Annals of mathematical statistics*, 25(3):603–607, 1954.

Lucien Tesnière. *Eléments de syntaxe structurale.* Editions Klincksieck, Paris, 1959.

Ivan Titov and James Henderson. A latent variable model for generative dependency parsing. In *Trends in Parsing Technology*, pages 35–55. Springer, 2010.

Peter D Turney, Patrick Pantel, et al. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188, 2010.

Assaf Urieli and Ludovic Tanguy. L'apport du faisceau dans l'analyse syntaxique en dépendances par transitions : études de cas avec l'analyseur Talismane. In *Actes de la 20e conférence sur le Traitement Automatique des Langues Naturelles (TALN'2013)*, pages 188–201, Les Sables d'Olonne, France, 2013.

Assaf Urieli and Marianne Vergez-Couret. Jochre, océrisation par apprentissage automatique : étude comparée sur le yiddish et l'occitan. In *Actes de TALARE 2013 : Traitement Automatique des Langues Régionales de France et d'Europe*, pages 221–234, Les Sables d'Olonne, France, 2013.

Karel Van Den Eynde and Piet Mertens. Le dictionnaire de valence dicovalence: manuel d'utilisation. *Manuscript, Leuven*, 2006.

Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

Eric Wehrli. Fips, a deep linguistic multilingual parser. In *Proceedings of the Workshop on Deep Linguistic Processing*, pages 120–127. Association for Computational Linguistics, 2007.

Kazuhiro Yoshida, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. Ambiguous part-of-speech tagging for improving accuracy and domain portability of syntactic parsers. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.

Daniel H Younger. Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2):189–208, 1967.

Yue Zhang and Stephen Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics, 2008.

Yue Zhang and Stephen Clark. Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 162–171. Association for Computational Linguistics, 2009.

Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *ACL (Short Papers)*, pages 188–193, 2011.

Yue Zhang and Joakim Nivre. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *COLING (Posters)*, pages 1391–1400, 2012.