Université de Montréal

L'atténuation statistique des surdétections d'un correcteur grammatical symbolique

par

Fabrizio Gotti

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences

en vue de l'obtention du grade de

Maître ès sciences (M. Sc.)

en informatique

Septembre 2011

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

L'atténuation statistique des surdétections d'un correcteur grammatical symbolique

présenté par

Fabrizio Gotti

a été évalué par un jury composé des personnes suivantes :

Yoshua Bengio,	président-rapporteur
Philippe Langlais,	directeur de recherche
Éric Brunelle,	membre du jury

Mémoire accepté le 6 février 2012

Résumé

Les logiciels de correction grammaticale commettent parfois des détections illégitimes (fausses alertes), que nous appelons ici *surdétections*. La présente étude décrit les expériences de mise au point d'un système créé pour identifier et mettre en sourdine les surdétections produites par le correcteur du français conçu par la société Druide informatique. Plusieurs classificateurs ont été entraînés de manière supervisée sur 14 types de détections faites par le correcteur, en employant des traits couvrant diverses informations linguistiques (dépendances et catégories syntaxiques, exploration du contexte des mots, etc.) extraites de phrases avec et sans surdétections. Huit des 14 classificateurs développés sont maintenant intégrés à la nouvelle version d'un correcteur commercial très populaire. Nos expériences ont aussi montré que les modèles de langue probabilistes, les SVM et la désambiguïsation sémantique améliorent la qualité de ces classificateurs. Ce travail est un exemple réussi de déploiement d'une approche d'apprentissage machine au service d'une application langagière grand public robuste.

Mots-clés : correction grammaticale, analyse syntaxique robuste, apprentissage machine supervisé, repérage de fausses alertes, modèles de langue probabilistes, désambiguïsation sémantique

Abstract

Grammar checking software sometimes erroneously flags a correct word sequence as an error, a problem we call *overdetection* in the present study. We describe the development of a system for identifying and filtering out the overdetections produced by the French grammar checker designed by the firm Druide Informatique. Various families of classifiers have been trained in a supervised way for 14 types of detections flagged by the grammar checker, using features that capture diverse linguistic phenomena (syntactic dependency links, POS tags, word context exploration, etc.), extracted from sentences with and without overdetections. Eight of the 14 classifiers we trained are now part of the latest version of a very popular commercial grammar checker. Moreover, our experiments have shown that statistical language models, SVMs and word sense disambiguation can all contribute to the improvement of these classifiers. This project is a striking illustration of a machine learning component successfully integrated within a robust, commercial natural language processing application.

Keywords: grammar checking, robust parsing, supervised machine learning, false alarm identification, statistical language models, word sense disambiguation

Table des matières

Liste des tableaux

Liste des figures

Remerciements

Puisqu'il faut bien commencer quelque part, je tiens à remercier mes parents d'avoir assemblé mon génome et d'en avoir assuré le service après-vente.

Je remercie également mon directeur de recherche, Philippe Langlais, chez qui la générosité n'a d'égale que la sagacité.

Ce mémoire présente un projet alliant les mondes industriel et universitaire; cela n'est jamais l'œuvre d'une seule personne. Je suis ainsi redevable à Guy Lapalme, professeur à l'Université, dont l'aide décisive a permis le succès de ce projet, et à Éric Brunelle, président et cofondateur de Druide informatique, qui nous a confié une tâche délicate et a tout fait pour sa réussite. Chez Druide, je tiens aussi à remercier Simon Charest, informaticien, et Mala Bergevin, linguiste-informaticienne, dont la grande méticulosité a rendu possibles les délicats échanges technologiques entre mon laboratoire de recherche et l'industrie.

Je tiens à saluer mes collègues du RALI, notamment Elliott Macklovitch, Alexandre Patry et Pierre-Etienne Genest, qui m'ont secouru quelques fois par leurs perspicaces remarques. Yoshua Bengio, professeur au DIRO, a diligemment relu ce texte et m'a suggéré des corrections.

Je remercie enfin Serge Parent pour ses conseils toujours érudits.

Chapitre 1
Introduction

1.1 La correction grammaticale, parent pauvre de la linguistique informatique ?

Depuis son avènement au début des années 1950, la linguistique informatique n'a cessé de multiplier ses axes de recherche. D'abord exclusivement intéressés par la « traduction mécanique », les chercheurs du domaine se sont attaqués dans les décennies qui ont suivi aux problèmes d'extraction d'information, de résumé automatique, de repérage d'information, etc. Au-delà de leurs retombées purement théoriques, ces efforts ont conduit à bon nombre d'applications concrètes dans ces champs de recherche.

Cependant, pour bien des utilisateurs de systèmes informatiques actuels, c'est avant tout la *correction grammaticale*[1] qui s'avère la manifestation la plus tangible des découvertes en traitement automatique des langues naturelles (TALN). Intégrée de façon transparente à des *outils d'aide à la rédaction*, la correction grammaticale consiste généralement à détecter et à signaler les fautes commises par l'utilisateur. Habituellement, des rectifications sont également proposées afin de remédier aux problèmes repérés. Les suites logicielles populaires, comme Microsoft Office, ajoutent

[1] Nous reportons à la section 1.2.1 la définition formelle de la correction grammaticale, afin d'alléger cette entrée en matière.

cette fonctionnalité à leurs programmes, du traitement de texte au tableur, en passant par l'incontournable logiciel de courriel.

Il n'en demeure pas moins que la recherche sur la correction grammaticale fait comparativement pauvre figure dans le domaine de l'ingénierie linguistique, comme le souligne Thierry Fontenelle (2005), du *Microsoft Speech & Natural Language Group*. Ce dernier déplore que le sujet n'ait plus la cote en TALN, ce qui, selon lui, est regrettable « vu le niveau orthographique de la population générale ». À notre avis, c'est d'autant plus fâcheux puisque ces correcteurs ont une vocation pédagogique, notamment dans l'apprentissage des langues secondes (voir par exemple TechWriter [Napolitano, 2009] pour l'anglais, Granska [Knutsson, 2003] pour le suédois ou encore le Correcteur 101 pour le français [Morin, 1995]).

Thierry Fontenelle nuance néanmoins ces constats en rappelant (Fontenelle, 2005) que des sommes considérables sont investies annuellement par plusieurs éditeurs de logiciels dans le développement des outils de correction grammaticale. Si ces fonds permettent sans doute l'amélioration de logiciels propriétaires, ils participent rarement à la publication de résultats de recherche dans leur champ d'étude, comme l'explique Agnès Souque (2008). Cette dernière, dans la même étude, rappelle que les outils de correction font « cruellement défaut à la communauté des utilisateurs de logiciels libres » qui, eux, ne jouissent pas de sources de financement comparables.

Les causes de ce désintérêt apparent sont nombreuses, et Clément *et al.* (2009) en esquissent quelques-unes. Les systèmes de correction grammaticale sont relativement dépendants de la langue à corriger et du niveau de connaissance du locuteur, et exigent le recueil patient des fautes probables par une équipe de linguistes. Cette tâche est difficile à automatiser, donc longue et coûteuse. Par ailleurs, l'utilité d'un tel système dépend de son intégration avec un traitement de texte, ce qui est une entreprise audacieuse en elle-même. L'avènement de suites de logiciels libres comme OpenOffice.org[1] ou LibreOffice[2] mitige toutefois cette dernière limitation.

Quelles que soient ces causes, nous partageons le point de vue de Clément *et al.* et souhaitons nous aussi que, comme pour les autres domaines du TALN, une commu-

[1] http://www.openoffice.org
[2] http://www.libreoffice.org

nauté de chercheurs se forme autour de la recherche en correction grammaticale et diffuse ses résultats. Nous espérons que le présent travail s'inscrira dans ce mouvement scientifique.

1.2 Inhibition des « surdétections » en correction grammaticale

Ce travail porte sur la mise en sourdine des surdétections (fausses alertes) lors du processus de correction grammaticale. Avant d'expliquer les surdétections et leur importance en analyse grammaticale, nous offrons quelques définitions afin de clarifier notre propos, et pour dissiper toute ambiguïté terminologique dans un domaine où règne une certaine confusion.

1.2.1 Quelques définitions

La *correction grammaticale* est le procédé qui consiste à détecter les fautes commises par un *scripteur* lors de la rédaction d'un texte et à aider ce dernier à les éliminer. Le logiciel qui effectue cette tâche s'appelle *correcteur grammatical* (en anglais *grammar checker*). Nous préférons éviter ici l'expression « correction grammaticale automatique », car la correction dont on parle dans ce travail requiert justement l'intervention d'un être humain, qui accepte ou rejette les propositions du logiciel. Dans ce mémoire, la correction grammaticale signifie exclusivement le travail effectué par le logiciel.

Les *fautes* commises par le scripteur peuvent être de plusieurs *types*. Les typologies de fautes sont légion dans la littérature du domaine. Le lecteur pourra consulter par exemple (Foster, 2005) pour l'anglais, (Brunelle, 1987) pour le français ou (Wedbjer Rambell, 2000) pour le suédois, entre autres langues. Nous présentons ici les types de fautes qui nous apparaissent saillants dans ces travaux. Ce sont des fautes :

- d'orthographe, où une suite de symboles constitue un mot inconnu dans la langue écrite;
- de syntaxe, où l'on contrevient aux préceptes qui régissent l'agencement des mots en syntagmes, puis en propositions, et enfin en phrases;
- de sémantique, où l'on confond un mot pour un autre (p.ex. confusion d'homophones, impropriété), ou encore lorsque le sens du message est compromis;

- de typographie, lorsque le scripteur se trompe dans la casse d'un mot, dans l'utilisation des points, deux-points, tirets, etc.;
- de style, lorsque l'utilisateur utilise un registre de langue inadéquat, lorsqu'il est trop verbeux, maladroit, répétitif, etc.

Une *détection* se produit lorsque le correcteur grammatical détermine qu'un mot ou groupe de mots (le *site* de la détection) contient une faute d'un certain type. Un site peut renfermer plusieurs détections, lorsque le scripteur commet plusieurs fautes au même endroit, par exemple pour une de ces redoutées « doubles fautes », telle `les *démontration`.

Une *correction* est une suggestion faite par le correcteur grammatical pour remédier à la faute détectée. Selon le contexte, le correcteur peut n'avoir aucune correction à proposer, ou encore une ou plusieurs corrections pour aider le scripteur. Dans ce dernier cas, un correcteur bien conçu suggère d'abord les plus plausibles.

La figure 1.1 montre la détection faite par le correcteur de Microsoft Word 2007 et celle faite par le correcteur d'Antidote RX, pour la même phrase. La correction apparaît également dans les captures d'écran. C'est là une faute de syntaxe, nommément un problème d'accord en nombre. Une des corrections proposées par Word est douteuse (la forme au pluriel); celle d'Antidote est la bonne.

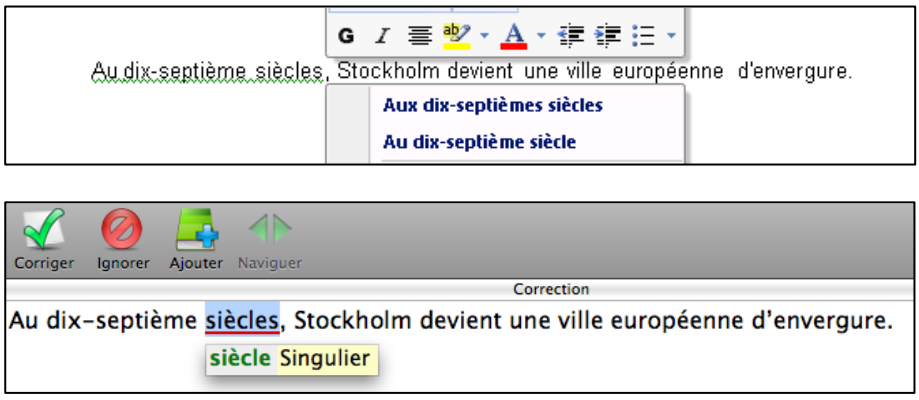


Figure 1.1 — Exemples de détection et de correction dans Word 2007 (haut) et Antidote RX (bas)

Uskoreit (1996)[1], cité dans (Sågvall Hein, 1998) raffine encore le domaine en introduisant deux termes supplémentaires, soit la *reconnaissance* (*recognition*), qui est l'identification des règles possiblement violées et le *diagnostic*, soit l'identification des sources potentielles de la faute.

1.2.2 Les surdétections et leur importance

La correction grammaticale est un processus qui s'appuie sur de nombreuses heuristiques, ainsi que sur des techniques et des ressources faillibles (voir chapitre 2). Par conséquent, elle est sujette à des erreurs. Nous distinguons deux grands types de problèmes en correction grammaticale : la surdétection et la sous-détection de fautes.

Les *surdétections* sont des détections qui sont faites erronément par le correcteur grammatical, à des sites où l'utilisateur n'a commis aucune faute. Elles sont appelées plus communément *fausses alertes* (voir notamment [Fontenelle, 2006] pour ce terme) ou encore *bruit*, mais le mot « surdétection » nous semble plus clair, et il a le mérite supplémentaire d'être celui employé par Druide informatique dans sa documentation et par d'autres professionnels de la langue (*cf.* [Morin, 1995] ou le manuel du Correcteur 101 [Machina Sapiens, 1999])[2].

On montre, à la figure 1.2, un exemple de surdétection dans Microsoft Word 2007 (sur le mot a) et un autre dans Antidote RX (sur le mot Où).

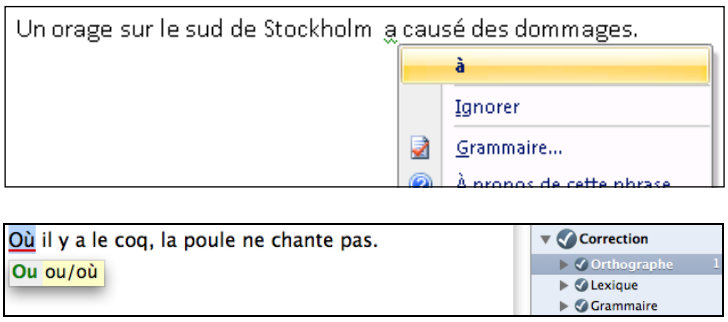


Figure 1.2 — Exemple de surdétection dans Word 2007 (haut) et Antidote RX (bas)

[1] Nous n'avons pu trouver la publication référencée par Sågvall Hein.
[2] Le monde anglophone utilise quant à lui les expressions *false flags*, *noise* ou encore *overflagging*.

Par opposition, les *sous-détections* sont des erreurs faites par le correcteur lorsqu'il omet de faire une détection sur le site d'une faute. On les appelle également *silences*. Nous ne les aborderons pas dans la présente étude. La figure 1.3 montre des sous-détections dans Microsoft Word[1]. La ligne de soulignement ondulée dans Word indique au moins une détection. Il en reste d'autres, qu'Antidote (partie inférieure de la figure) relève avec davantage de succès (mots soulignés).

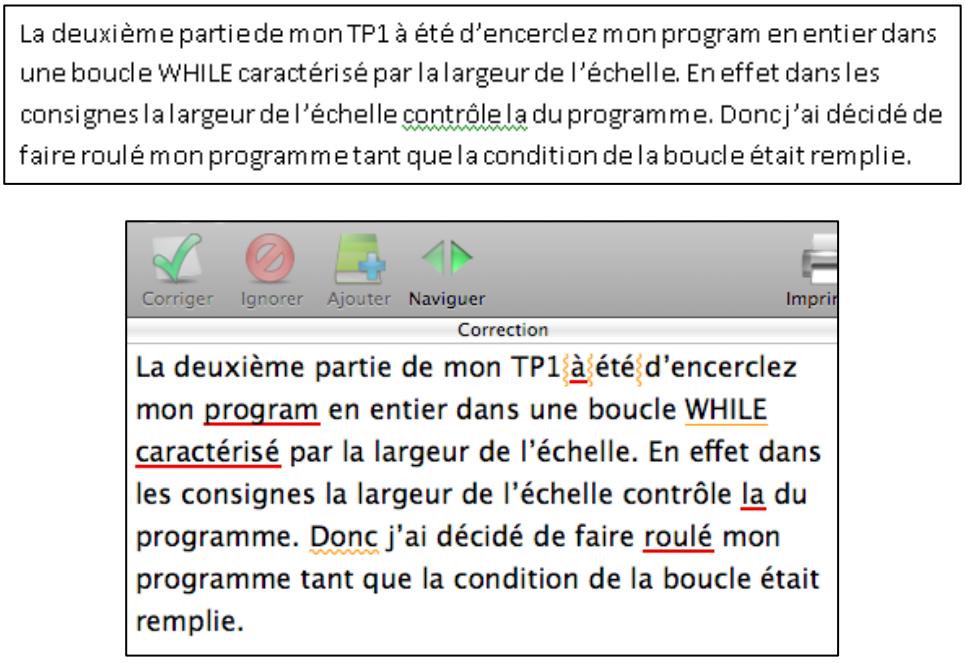


Figure 1.3 — Sous-détections dans Word 2007 (haut) et les détections d'Antidote RX (bas)

Ce sont les surdétections, et leur mise en sourdine, qui feront l'objet de ce mémoire. Ces fausses alertes sont d'autant plus gênantes qu'elles découlent d'une intention *a priori* louable, celle de l'exhaustivité dans les détections. En effet, un correcteur grammatical qui vise l'identification d'un maximum de fautes dans un texte donné court le risque de commettre des surdétections : c'est que son zèle exige qu'il abaisse ses seuils de tolérance. Il appartient donc aux concepteurs du correcteur de sacrifier l'exhaustivité (ou *rappel*) pour taire les surdétections, un problème typique de compromis coûts-bénéfices.

[1] Cet exemple est tiré des travaux remis à l'auteur de ce mémoire dans le cadre de son travail d'enseignant au collégial, regrettablement.

Pourquoi est-il si important d'atténuer ces fausses alertes ? C'est que leur présence dans un outil de correction grammaticale en diminue la fiabilité et l'utilité telles que perçues par les utilisateurs. Ainsi, lorsque Microsoft a effectué ses études de marché pour guider les efforts de développement de son outil de correction, elle a conclu (Helfrich & Music, 2000) que la diminution du taux de fausses alertes par page a préséance sur l'exhaustivité[1]. Leur correcteur grammatical se veut en fin de compte un outil d'aide à la productivité, et il ne doit pas « distraire, retarder ou embêter les gens »[2]. C'est là un constat récurrent, comme l'explique (Bernth, 1997), qui cite trois études indépendantes qui abondent dans ce sens.

Le problème des surdétections est amplifié lorsqu'un correcteur est conçu pour couvrir une grande variété de types de fautes. En effet, plus le correcteur cherche de types de fautes, plus il a de chances de commettre des surdétections. Au final, ce pari de l'exhaustivité peut donc être moins utile que prévu pour l'utilisateur.

1.3 Le projet Scorali

Ce mémoire explique les recherches que nous avons effectuées dans le cadre d'une collaboration entre la société Druide informatique et notre laboratoire(le RALI[3]), spécialisé en recherche appliquée en linguistique informatique. Ce projet, baptisé *Scorali*, consiste à réduire les surdétections produites par l'analyseur symbolique de Druide à l'aide de l'apprentissage machine. Cette idée, proposée par Druide, est un métissage de deux paradigmes souvent mis en opposition, soit l'approche symbolique, d'une part, et l'approche statistique, d'autre part.

L'auteur de ce mémoire s'est chargé de mener à bien cette entreprise durant la première moitié de l'année 2009, avec la participation de Druide.

1.4 Structure du mémoire

Au chapitre 2, nous brossons un tableau des stratégies de conception et d'amélioration des correcteurs grammaticaux puis, au chapitre 3, nous présentons le

---

[1] Les auteurs relatent une anecdote éloquente : même si des règles claires gouvernent l'emploi de la majuscule pour marquer la différence entre une langue et un peuple (par exemple, « parler le suédois » et « les Suédois »), leurs études de marché montrent que les utilisateurs *ne souhaitent pas* que ces détections soient faites.

[2] Notre traduction de « We don't want to distract, delay or bother people… », tiré de l'article de Helfrich et Music (2000).

[3] http://rali.iro.umontreal.ca/

projet Scorali en détail. Ensuite, au chapitre 4, nous expliquons l'approche que nous avons privilégiée pour réduire les surdétections. Nous présentons nos résultats au chapitre 5 et en discutons au chapitre 6. Dans cette dernière section, nous soulignons également les (nombreuses) pistes de recherche qui se sont dévoilées au fur et à mesure de notre travail. Une conclusion suit au chapitre 7. L'annexe A consigne l'entièreté des résultats obtenus. Enfin, à l'annexe B, le lecteur trouvera une copie de l'article *Reducing Overdetections in a French Symbolic Grammar Checker by Classification*, soumis et accepté à la conférence CICLing 2011. Cet article, rédigé par nos collaborateurs et nous, résume une partie des résultats obtenus dans ce travail.

Chapitre 2
Pistes d'amélioration des correcteurs grammaticaux

Notons d'emblée que ce n'est pas le propos de ce mémoire de décrire exhaustivement un domaine aussi vaste que le fonctionnement des correcteurs grammaticaux, *a fortiori* lorsque la quasi-totalité de ceux-ci sont des logiciels propriétaires dont les mécanismes sont protégés par le secret industriel.

Notre objectif dans cette section est plutôt de présenter certaines des techniques les plus saillantes de correction grammaticale et de montrer quelques-unes des pistes qu'ont empruntées les chercheurs du domaine pour améliorer ces correcteurs. Il sera ensuite plus aisé de situer nos efforts dans ce paysage.

2.1 La grammaire du correcteur grammatical

Dans la littérature, un analyseur (« parseur ») est l'outil privilégié pour la détection de fautes. L'analyseur s'appuie sur une *grammaire*, qui est en mesure d'accepter ou de rejeter des phrases ou des fragments de phrases, pour une langue donnée, et pour un domaine donné.

Deux grandes familles de grammaires se retrouvent dans les articles du domaine : les grammaires de surface et les grammaires profondes. Les premières ne s'intéressent

qu'aux phénomènes de surface de la langue écrite, comme la séquence des mots. Elles s'apparentent aux langages informatiques dits réguliers. Les secondes tentent une analyse plus profonde et bâtissent des arbres d'analyse syntaxique. Plus sophistiquées, elles s'apparentent aux langages informatiques dits hors contexte.

Chaque approche a ses forces et ses faiblesses, et se prête à des améliorations très différentes, le cas échéant.

2.2 Correcteurs avec grammaire de surface

Les approches les plus simples de correction grammaticale s'appuient sur des grammaires régulières s'intéressant aux séquences de mots ou de parties du discours du texte à corriger. Ce dernier est d'abord segmenté en phrases, puis en mots[1]. On peut ensuite procéder à l'identification des parties du discours (étiquetage morphosyntaxique), souvent à l'aide de stratégies statistiques éprouvées.

Il devient ensuite possible de construire une *grammaire négative*, c'est-à-dire un ensemble de règles de reconnaissance de fautes dans les textes. Dans ce cas, lorsqu'une séquence est retrouvée dans le texte, une faute est signalée à l'utilisateur. Comme le souligne (Souque, 2008), tous les outils de correction libres qu'elle a étudiés fonctionnent ainsi. Parmi eux, citons An Gramadóir[2] (Scannell, 2011), LanguageTool[3] (Naber, 2003) et After the Deadline[4] (Mudge, 2009). Les deux derniers peuvent s'intégrer à OpenOffice.org.

Un exemple de règles embarquées dans la version anglaise d'After the Deadline est illustré à la figure 2.1. Les 3 règles illustrées, relativement complexes à créer, corrigent les propositions infinitives comme `to *is or not to *is`. La première règle traite le cas spécial `to *is`. La seconde règle stipule que le mot `to` suivi d'un verbe à la 3e personne du singulier, au présent, lui-même suivi d'un déterminant et d'un nom, constitue une détection. Elle contient également une suggestion de correction. Elle s'appliquerait dans le cas de la séquence `to *greets a Swede.` qui serait corrigée

[1] D'un point de vue purement informatique, ce sont plutôt des *tokens* qui sont dégagés lors de la segmentation, puisque le traitement par l'ordinateur considère bien souvent sur un pied d'égalité la chaîne `fur` (dans la locution `au fur et à mesure`) et la virgule `,`. Au confluent de la linguistique et de l'informatique, la distinction entre mot et *token* est périlleuse, et mériterait une section à elle seule, ce qui serait excessif ici.
[2] http://borel.slu.edu/gramadoir/index.html
[3] http://www.languagetool.org
[4] http://open.afterthedeadline.com

par `to greet a Swede.` Les règles sont hautement lexicalisées (dans notre exemple, il y a une règle pour `to` et une autre pour `To`) et spécifient manuellement les corrections à proposer (ici, le retour à la forme infinitive du verbe).

```
to is::filter=kill
to .*/VBZ .*/DT|NN::word= \1:base \2::pivots=\1,\1:base
To .*/VBZ .*/DT|NN::word= \1:base \2::pivots=\1,\1:base
```

Figure 2.1 — Trois règles du logiciel After the Deadline (Mudge, 2009) pour l'anglais

En fin de compte, ce que les auteurs de ces règles tentent de capturer (consciemment ou pas) sont les cas de figures où l'*unification* échoue entre les mots. L'unification est un concept crucial en analyse syntaxique et, partant, en correction grammaticale. On peut le définir, comme le font Sag *et al.* (1986), comme le processus de fusion du contenu informationnel de deux *structures de traits* (*feature structures*). Ces structures de traits sont tout simplement des caractéristiques linguistiques attribuées aux mots étudiés.

La figure 2.2 montre un exemple de cette unification, pour les mots `Il` et `vient` dans la phrase `Il vient de Stockholm.` Toutes les structures de traits potentielles ne sont pas représentées. On voit que `Il` et `vient` ont des structures de traits compatibles, soit les couples attribut-valeur de nombre et de personne. L'unification (notée $A \sqcup B$) est donc possible, et il en résulte la structure maximale qui est à la fois compatible avec A et B, soit le terme de droite dans la figure 2.2.

$$\underset{\text{Il (pronom)}}{\begin{bmatrix} genre = masculin \\ nombre = singulier \\ personne = 3 \end{bmatrix}} \sqcup \underset{\text{vient (verbe)}}{\begin{bmatrix} mode = infinitif \\ nombre = singulier \\ personne = 3 \end{bmatrix}} = \begin{bmatrix} nombre = singulier \\ personne = 3 \end{bmatrix}$$

Figure 2.2 — Unification de deux structures de traits

Dans le cadre de la correction grammaticale, l'échec lors de l'unification (noté $A \perp B$) nous intéresse tout particulièrement. En effet, cet échec d'unification signale des

structures de traits irréconciliables, comme dans la phrase `*Ils vient de Stockholm.`, entre `Ils` et `vient`. Les règles des grammaires négatives mentionnées plus tôt tentent donc d'énumérer tous les patrons où ces échecs d'unification peuvent survenir, afin de les signaler.

Or, ces règles sont le plus souvent construites à la main, et lexicalisées, ce qui requiert un effort de développement considérable. Si ce n'était que de cette dernière limitation, il serait peut-être envisageable de compter sur l'inépuisable et charitable main-d'œuvre de la communauté du logiciel libre pour bonifier ces correcteurs.

Cependant, c'est sans compter l'explosion combinatoire à laquelle fait face ce genre d'approche : prévoir toutes les positions, variantes et types de fautes au sein d'un simple groupe nominal de quelques mots confine à l'impossible. Qui plus est, l'approche souffre de sa difficulté à capturer des relations entre mots séparés par quelques mots-écrans. Par conséquent, une excellente couverture des cas de figure utiles est difficile à atteindre avec cette approche.

Souque (2008) explique en détail ces limitations et propose de remédier au problème en découpant d'abord en syntagmes le texte à corriger. Un *syntagme* est un mot de tête et ses subordonnés immédiats. La littérature les désigne parfois « chunks » ou segments. Une fois ces syntagmes identifiés, tous les éléments qu'ils contiennent doivent s'accorder, et il est possible, une fois cela vérifié, d'attribuer à chaque syntagme les structures de traits de leur tête lexicale, pour ensuite tenter d'unifier ces syntagmes entre eux. Cette façon de « diviser pour régner » permet à Souque de tenter un contrôle de l'explosion combinatoire des règles.

Aït-Mokhtar *et al.* (2002) appellent fort joliment ces syntagmes « îlots de certitude », dans leur exposé sur les analyseurs syntaxiques qui procèdent eux aussi en plusieurs passes, en prenant des décisions en cascade. Il n'en demeure pas moins que la construction de listes de patrons de détection nous semble peu pratique.

Certains chercheurs se sont plutôt tournés vers les *grammaires positives*, soit celles qui décrivent les phrases correctes d'une langue donnée. Les plus répandues des grammaires de ce type font appel à la statistique et essaient de modéliser les séquences légitimes de la langue. Wagner *et al.* (2007) en font l'historique et citent en pionnier Atwell (1987) qui utilise un étiqueteur de parties du discours (*POS tagger*)

pour détecter les couples d'étiquettes (bigrammes) improbables dans un texte à corriger, en les comparant à ceux d'un corpus de référence écrit dans un anglais châtié.

Plusieurs autres stratégies sont possibles avec ces outils statistiques. Par exemple, Sofkova Hashemi (2001) fait usage de grammaires régulières (positives) sur les parties du discours pour détecter les fautes grammaticales. Elle utilise le concept de soustraction des automates qu'elle crédite à Chanod. L'idée est que, si une grammaire à large grain (par exemple ne faisant pas usage du genre et nombre associés aux étiquettes) permet d'analyser une séquence, mais pas une grammaire plus précise, alors il est probable qu'une faute est présente dans cette séquence de mots.

Napolitano *et al.* (2009) font flèche de tout bois et modélisent les séquences de 2, 3, 4 mots[1] ainsi que les séquences de 2, 3, 4 parties du discours, pour TechWriter, un aide logiciel d'aide à l'apprentissage de l'anglais. Toute séquence de mots ou de parties du discours qui survient dans le texte du scripteur avec une fréquence relative plus grande que celle du modèle devient le site d'une détection. Puisque leur outil s'adresse aux locuteurs non natifs de l'anglais aux prises avec la rédaction d'un article scientifique dans la langue des Beatles, le corpus de référence qu'elles utilisent est composé de 426 articles principalement techniques de l'*American National Corpus*[2].

C'est bien là que la difficulté surgit avec ce type d'approche : elles font usage d'un corpus de textes pour faire l'acquisition des séquences de mots, lemmes ou étiquettes morphosyntaxiques, qui sont stockées puis comparées aux séquences du texte à corriger. Or, peu importe le soin qu'apportent les chercheurs à leur sélection de textes « étalons », la représentativité du corpus d'entraînement n'est jamais parfaite. Plus grave encore, étant donnée l'extraordinaire liberté dont jouit celui qui rédige un texte, il n'est jamais certain qu'une séquence de mots légitime lui venant à l'esprit fera partie du modèle de langue « correcte », ce qui générera alors une surdétection. Même les corpus d'entraînement les plus volumineux (extraits par exemple de millions de documents de la toile), ne permettent pas d'éliminer ce risque. De même, une panoplie d'outils statistiques visant le lissage (*smoothing*) de distributions de séquences de mots peut atténuer le problème, sans pour autant le faire disparaître.

---

[1] On appelle ces séquences des *n*-grammes, où *n* est la longueur de la séquence étudiée. Nous reportons à la section 4.4.3 l'explication plus détaillée de ceci.
[2] http://www.americannationalcorpus.org

2.3 Correcteurs avec analyse grammaticale profonde

Il est clair à ce point que les correcteurs se basant sur une analyse grammaticale de surface sont limités, et qu'ils souffrent notamment de leur incapacité à analyser des dépendances grammaticales à longue distance.

Pour tenter de remédier à ce problème, une analyse syntaxique plus en profondeur est préconisée par plusieurs auteurs, notamment Bustamante et León (1996) pour le correcteur de l'espagnol GramCheck, Sågvall Hein (1998) pour le correcteur du suédois ScarCheck et Clément *et al.* (2009) pour un correcteur du français. Les correcteurs propriétaires, tels WhiteSmoke[1], Cordial[2] ou ceux de Microsoft Office[3], demeurent des boîtes noires, donc il serait hasardeux de les inclure ici. Pour Antidote, cependant, notre collaboration avec Druide Informatique nous a assuré que leur correcteur adoptait cette approche.

Dans leur érudite revue de la littérature du domaine, Bustamante et León (1996) relèvent que si, pendant les années 1970, les correcteurs grammaticaux étaient plus ou moins des sous-modules de systèmes de compréhension du langage (*natural language understanding*), ils sont devenus des projets à part entière dans les années 1980. C'est vers ce moment qu'ils ont été vus comme une extension naturelle des techniques existantes d'analyse syntaxique. En particulier, les grammaires basées sur des systèmes de contraintes (comme l'est l'unification des structures de traits, vue à la section 2.2) se prêtent bien à ce genre d'exercice. Une règle de grammaire utilisant des structures de traits, tirée de (Clément *et al.*, 2009) est illustrée à la figure 2.3 ci-dessous.

```
gn[nb=N;gen=G;pers=3] → det[nb=N;gen=G] sadj[nb=N;gen=G;type=anté]*
  nc[nb=N;gen=G] sadj[nb=N;gen=G;type=post]* gp[]? rel[nb=N;gen=G]?
```

Figure 2.3 — Exemple de règle grammaticale extraite de (Clément *et al.*, 2009)

On y voit le non-terminal `gn` (pour groupe nominal) et sa dérivation en une expression régulière intégrant d'autres non-terminaux (comme `det`, déterminant), qualifiés

[1] http://www.whitesmoke.com/
[2] Conçu par la société française Synapse Développement (http://www.synapse-fr.com/).
[3] La suite Microsoft Office (http://office.microsoft.com/) inclut des correcteurs grammaticaux pour plusieurs langues.

par un ensemble de couples attribut-valeur appartenant à une structure de traits plate, c'est-à-dire sans hiérarchie. Un des attributs est le nombre nb, dont la valeur est la variable N, qui doit avoir une valeur cohérente dans la dérivation si l'unification doit réussir.

Il est possible, pour « forcer » la grammaire à accepter certaines déviations du langage, de *relâcher* certaines de ces contraintes. En simplifiant, on peut alors garder la trace de ces relaxations et les considérer comme une faute, puis il est alors possible de proposer des corrections en tenant compte de la nature de ces relaxations.

Dans une grammaire hors contexte équipée d'un système de contraintes, comme le proposent Bustamante et León dans l'article précité, le désaccord des contraintes, on l'a vu, peut fort bien être géré par le relâchement de celles-ci. Les auteurs appellent ces fautes *non structurelles*, et donnent comme exemples les problèmes d'accord entre le sujet et le verbe ou d'accord en genre et nombre entre un déterminant et un nom.

Toutefois, ce ne sont pas toutes les fautes qui peuvent être capturées par le relâchement des contraintes. En effet, les mêmes chercheurs opposent ces fautes à celles qui sont *structurelles*, c'est-à-dire celles qui violeraient la structure hors-contexte. Un exemple de ce dernier type de faute est un mauvais rattachement prépositionnel.

Les fautes structurelles sont traitées à l'aide de règles spécifiques qui encodent les fautes typiques (donc par une grammaire négative). Ces règles sont soit ajoutées à la grammaire de la langue du correcteur, soit dans des grammaires satellites consultées sur demande, selon par exemple le type de texte à corriger. Ce genre de fautes rend beaucoup plus difficile la suggestion d'une correction.

Dans tous les cas, le développement d'un tel système est une entreprise complexe qui requiert l'ajustement d'heuristiques linguistiques fines intervenant à même le processus d'analyse syntaxique.

C'est donc dire que l'on peut apporter des améliorations décisives à ce genre de système par l'addition patiente de règles dans les grammaires (positives et éventuellement négatives) utilisées par le correcteur. Wagner *et al.* (2007) mentionnent ainsi que l'analyseur ParGram pour l'anglais a pris plusieurs années de développement. Pour le correcteur de Microsoft (Helfrich & Music, 2000), cet apport

se fait grâce à l'effort de linguistes locuteurs de la langue concernée, ayant accès entre autres ressources à des livres de référence, des corpus de fautes ainsi qu'aux rétroactions des utilisateurs du système.

Au cœur de ces efforts de perfectionnement se situe la question de la *robustesse* de l'analyseur syntaxique. Comme l'écrivent Foster et Vogel (2004), « un analyseur robuste doit pouvoir se comporter de façon raisonnable lorsqu'il est confronté à des entrées qui ne se conforment pas à l'idée qu'il s'est faite d'un langage donné »[1]. On comprendra aisément que, dans le cas de la correction grammaticale, cette caractéristique est hautement désirable. Un analyseur robuste doit être en mesure de détecter les déviances grammaticales, d'identifier des solutions de repli et de produire détections et corrections pour l'utilisateur.

Parmi les chercheurs qui ont étudié la non-grammaticalité (*ungrammaticality*), les travaux de Jennifer Foster attirent l'attention (Foster 2004; Foster & Vogel, 2004; Wagner *et al.*, 2007). Au cours de ses recherches, elle a, avec le concours de ses collaborateurs, créé et évalué la qualité des corpus de fautes de façon automatique ou manuelle puis étudié la typologie de celles-ci, soumis ces phrases déviantes à divers analyseurs (dont ceux de Charniak [2000] et Collins [2003]) et analysé leurs sorties. Elle a également tenté d'en améliorer la robustesse à la lumière de ces résultats.

Ce mémoire ne peut évidemment pas rendre justice à tous ces travaux, mais, par exemple, dans (Foster & Vogel, 2004), une grammaire négative est construite empiriquement, et entre en jeu lorsque l'analyse traditionnelle échoue. Cette grammaire est construite à partir des règles de la grammaire positive, de façon à ce que la relation entre une règle de la grammaire positive et la règle correspondante dans la grammaire négative renseigne à la fois sur la façon de secourir l'analyseur et sur la correction à apporter à la phrase initiale. La représentativité et la couverture du corpus utilisé pour la typologie des fautes commises sont à nouveau critiques, ici.

Dans sa thèse de doctorat (Foster, 2004), Foster présente notamment une forme plus restreinte de relaxation de l'unification de structures de traits pour les fautes d'accord, qui permet l'analyse de phrases déviantes grammaticalement.

[1] Notre traduction.

Elle cite également d'autres stratégies pour aborder le texte agrammatical, par exemple *l'assemblage d'analyses* (*parse fitting*), qui consiste à réunir les résultats fragmentaires d'une analyse qui a échoué, avec des résultats parfois douteux. Toutefois, l'analyseur n'est jamais muet ainsi et certains considèrent que c'est là l'essence de la robustesse. L'analyseur du néerlandais Alpino est un exemple d'analyseur robuste, dans la mesure où, en cas d'échec de l'analyse d'une phrase, il retourne un ensemble minimal d'analyses fragmentaires pour des séquences non superposées de la phrase en entrée (van Noord, 2004).

Foster explique aussi (Foster, 2004) l'analyse probabiliste, qui attribue à chaque dérivation de la grammaire une probabilité calculée à partir d'analyses d'entraînement (*treebank*[1]). Le mérite de cette dernière approche est qu'elle permet de générer plusieurs analyses candidates (c'est la *surgénération*) pour une phrase à analyser. Aux prises avec une phrase non grammaticale, cette approche se montre alors robuste en proposant au moins une liste de solutions possibles, chacune dotée d'une probabilité. Malheureusement, cela ne veut pas dire que ces analyses sont correctes… À notre connaissance, ces deux dernières techniques de renforcement ne sont pas associées à la correction grammaticale.

En dernière analyse, l'amélioration des analyseurs embarqués dans les correcteurs grammaticaux consiste soit à ajouter des ressources linguistiques au système (par l'ajout de règles), soit à procéder à de délicats réglages et chirurgies au sein de l'analyseur.

2.4 Améliorations guidées par la fouille d'erreurs

Gertjan van Noord (2004) a proposé, dans une étude abondamment citée, de s'attaquer à l'amélioration d'un analyseur en se plaçant *en aval* de celui-ci, c'est-à-dire en analysant ses sorties. Son approche ingénieuse consiste à faire l'analyse d'un grand nombre de phrases néerlandaises à l'aide d'Alpino[2], un analyseur de dépendances syntaxiques à large couverture, puis à comparer la fréquence des mots et séquences de mots dans les phrases qui font échouer l'analyseur avec la fréquence des mêmes mots et séquences dans les phrases qui ne posent pas de problèmes.

---

[1] Le plus célèbre de ces corpus d'entraînement est peut-être le Penn Treebank, disponible à http://www.cis.upenn.edu/~treebank/.

[2] http://www.let.rug.nl/vannoord/alp/Alpino/

Il définit ainsi l' « analysibilité » (*parsability*) d'une séquence de mots $w_i \dots w_j$ comme $R(w_i \dots w_j)$, selon la formule

$$R(w_i \dots w_j) = \frac{C(w_i \dots w_j|\text{OK})}{C(w_i \dots w_j)}$$

où $C(w_i \dots w_j|\text{OK})$ est le nombre de fois où la séquence $w_i \dots w_j$ se trouve dans une analyse réussie, et $C(w_i \dots w_j)$ est le nombre de fois où la séquence $w_i \dots w_j$ se trouve dans les phrases à analyser. Lorsque ce ratio est significativement plus bas que le ratio de couverture de l'analyseur, alors on peut considérer cette séquence comme suspecte. Il ne considère que les séquences pour lesquelles $C(w_i \dots w_j) \geq 5$ pour éviter le bruit dus aux mots trop rares.

L'approche est féconde, car elle a permis à van Noord d'identifier à la fois des erreurs dans les ressources linguistiques utilisées par Alpino et dans les règles de sa grammaire positive. Ainsi, il a isolé des erreurs intervenant lors de la segmentation en mots, dans le lexique (par exemple, des erreurs sur le genre d'un mot, ou une description incomplète pour une entrée), dans le traitement des expressions idiomatiques ou de certaines entités nommées. De plus, il a pu dégager des faiblesses dans les règles du parseur, pour certains phénomènes syntaxiques, telle une apposition modifiant un pronom. Ces constatations ont mené à des améliorations d'Alpino.

Sagot & Villemonte de la Clergerie (2009) s'inspirent de l'étude de van Noord et vont plus loin en tentant d'identifier, *pour chacune des phrases* pour lesquelles l'analyse a échoué, le mot qui est le plus susceptible d'avoir causé cet échec. Ils appellent ce mot *suspect principal*. Les auteurs étendent leur stratégie aux bigrammes de formes et de lemmes suspects. Leur étude exhaustive utilise non pas un mais deux analyseurs syntaxiques du français (FRMG et SxLFG-fr) utilisant le même lexique (le Le*fff* 2[1]) et la même chaîne de prétraitements (SxPipe[2]), sur plusieurs millions de phrases.

En utilisant un algorithme de point fixe, ils calculent le taux de suspicion moyen des mots et bigrammes, permettant d'identifier ceux qui sont vraisemblablement source d'erreurs. Ceux-ci agissent comme révélateurs de problèmes de segmentation, de

[1] http://alpage.inria.fr/~sagot/lefff.html
[2] http://alpage.inria.fr/~sagot/sxpipe.html

bogues informatiques divers ou encore d'erreurs dans les entrées du Le*fff*. Le fait d'avoir utilisé deux analyseurs partageant des ressources importantes leur permet également d'attribuer certaines erreurs à la partie commune du pipeline d'analyse lorsque les résultats sont les mêmes, pour un certain suspect, entre les deux chaînes. Leur étude a permis de surcroît l'augmentation de la couverture de FRMG lorsque confronté à un corpus technique botanique.

2.5 Complémentarité des approches symbolique et statistique

Les stratégies décrites à la section précédente montrent la fécondité du croisement d'une composante symbolique (un correcteur) avec une composante statistique placée en aval. Ce succès n'est pas anodin, car il révèle, du moins dans le cas qui nous intéresse, la complémentarité de deux paradigmes souvent considérés en porte-à-faux en traitement des langues : d'une part, l'approche à base de règles et, d'autre part, l'approche basée sur l'analyse statistique de corpus relativement volumineux.

Ces deux méthodes pourraient donc se réconcilier avantageusement dans notre cas, ce qui est assez raisonnable : dès 1972, Kanal & Chandrasekaran situaient les deux philosophies sur un seul et même continuum et prônaient leur combinaison plutôt que leur exclusion mutuelle. Leur étude de techniques de reconnaissance de formes (*pattern recognition*) a conduit les auteurs à recommander la subdivision hiérarchique d'un problème en sous-problèmes, chacun associé à une des deux approches.

Depuis lors, de nombreux autres terrains d'entente ont été découverts. Ainsi, en juillet 1994, l'atelier de l'ACL (*Association for Computational Linguistics*) intitulé *The Balancing Act*[1] (Klavans & Resnik, 1996), était tout entier dédié à la délicate combinaison des deux approches et montrait leur complémentarité. Sans entrer dans les détails, la quinzaine d'articles publiés traitaient de sujets aussi divers que l'extraction terminologique, l'analyse syntaxique, la compréhension du langage et l'étiquetage morphosyntaxique.

En 2007, lors de l'atelier de l'ACL sur la traduction automatique statistique (Callison-Burch *et al.*), deux expériences ont été menées indépendamment pour combiner statistiques et règles symboliques en traduction automatique. En effet, Dugast *et al.* (2007) ainsi que Simard *et al.* (2007) ont tous deux placé un module de post-édition

[1] Une traduction de ce titre pourrait être « Le juste équilibre ».

statistique *en aval* de SYSTRAN, un système de traduction commercial à base de règles. Les deux groupes de chercheurs observent ainsi une amélioration très significative de la qualité des traductions en ajoutant la post-édition statistique.

L'approche hybride semble donc prometteuse, pour autant que l'on parvienne à trouver « le juste équilibre » entre outils statistiques et symboliques. C'est ce que nous tenterons ici.

Chapitre 3
Le projet Scorali

Le projet Scorali, fruit d'un partenariat entre Druide informatique et le RALI, a pour but l'amélioration d'un système embarquant un analyseur, comme l'ont fait les chercheurs cités au chapitre précédent. Le problème que nous tâchons de résoudre diffère : si les erreurs que nous tentons de réduire ici sont bien, en partie, celles engendrées par des problèmes d'analyse syntaxique, notre projet porte sur un problème bien ciblé et quelque peu original, soit celui du repérage des surdétections dans les sorties d'un correcteur grammatical. Le but du travail présenté ici est de diminuer ces surdétections.

Similairement aux travaux de van Noord (2004) et de Sagot & Villemonte de la Clergerie (2009), pour l'analyse syntaxique, et de Dugast *et al.* (2007) et Simard *et al.* (2007), pour la traduction automatique, nous nous plaçons en aval de la chaîne de traitement et travaillons sur les sorties produites. Pour van Noord et Sagot & Villemonte de la Clergerie, c'est dans le but de repérer les formes suspectes, afin de bonifier le pipeline syntaxique en amont, presque toujours après inspection manuelle. Dans notre cas, cependant, nous n'avons pas accès à l'analyseur syntaxique et ne pouvons donc le modifier. Par ailleurs, notre ambition première est plutôt de fournir un module qui puisse automatiquement s'insérer en aval du système actuel pour mettre les surdétections en sourdine. On verra que, nonobstant cet objectif premier, nous

aurons quand même la possibilité de fournir une rétroaction intéressante sur le fonctionnement de l'analyseur.

En théorie, notre approche pourrait donc être adaptée facilement à un correcteur autre que celui utilisé dans notre étude, puisque nous nous situons après le correcteur. La tâche que nous étudions est également plus simple puisque nous n'avons pas à localiser les fautes ni même à proposer une correction, ceci étant pris en charge par le correcteur.

3.1 Druide informatique et Analytix

Druide informatique[1] est le partenaire commercial du RALI dans le projet Scorali. C'est une société montréalaise spécialisée en linguistique informatique, fondée en 1993. Elle offre divers produits grand public, notamment ceux de révision linguistique et d'aide à la rédaction.

Elle développe depuis des années une technologie d'analyse symbolique et de détection appelée Analytix, intégrant une grammaire de dépendances syntaxiques[2] à large couverture ainsi que plusieurs ressources linguistiques riches, pour la plupart élaborées et maintenues manuellement.

Les grammaires de dépendance s'inscrivent dans une théorie syntaxique développée par Lucien Tesnière[3], où la structure d'une phrase est décrite à l'aide de relations de dépendance entre un mot régissant (parent) et un mot subordonné (fils). Le subordonné dépend du régissant et ce dernier régit le subordonné. De plus, un subordonné n'a qu'un seul régissant, alors qu'un régissant peut avoir plusieurs subordonnés. Ce type d'analyse ne dépend pas de l'ordre des mots, si bien qu'un tronçon de phrase comme `les ruisseaux capricieux` et `les capricieux ruisseaux` donneront les mêmes dépendances, ce qui permet une certaine souplesse.

Analytix traite ainsi des phénomènes complexes et un soin particulier a été investi pour assurer sa robustesse face aux déviances grammaticales de tout acabit. Cette dernière qualité est hautement désirable pour un correcteur, comme on l'a vu à la

[1] http://www.druide.com/
[2] Nous montrons un exemple de ces dépendances syntaxiques à la section 3.3.2.
[3] Son ouvrage maître est *Éléments de syntaxe structurale*, publié de façon posthume en 1959. Nous avons consulté le document de Schwischay (2003) pour nous familiariser avec le travail de Tesnière.

section 2.2. Les détections sont faites automatiquement à partir de l'inspection des arbres d'analyse produits. Lorsque nous parlons du « correcteur » dans ce travail, nous faisons référence à Analytix.

3.2 Cahier des charges

Le projet Scorali, de par les circonstances de sa naissance, a la particularité d'être à la fois guidé par des impératifs commerciaux et par notre désir de chercheur de faire progresser le domaine. C'est dire qu'une partie de la difficulté du projet consiste à concilier ces deux élans.

Cela signifie également que Scorali obéit à un cahier des charges esquissé dès les débuts du projet, puis précisé rapidement dans les semaines qui ont suivi.

Ce cahier délimite le but de Scorali : le projet doit reconnaître les surdétections pour 14 types de fautes que traite Analytix. Druide a été en mesure de fournir, pour chacun de ces types de faute, des exemples de bonnes et de mauvaises détections (surdétections). Ces données sont détaillées à la section suivante. Nous n'avons cependant pas eu accès au code ou à une API d'Analytix : notre travail se situe exclusivement en post-traitement des sorties du correcteur.

De par sa nature, le projet se prête à merveille à l'entraînement supervisé de classificateurs, à partir d'exemples annotés. Nous verrons plus en détail au chapitre 4 pourquoi nous avons privilégié cette approche. Pour être considéré comme intéressant par Druide, un classificateur doit repérer au moins 66 % des surdétections et ne pas éliminer plus de 10 % des détections légitimes ce faisant. Ces seuils ont été établis tôt dans la chronologie du projet, à la suite de notre présentation de résultats préliminaires à Druide.

Le cahier des charges fait également mention que les classificateurs développés doivent être intégrables facilement dans la chaîne de traitement du correcteur. Bien que ces contraintes n'aient pas été chiffrées, il était hors de question de livrer un classificateur qui requière quelques mégaoctets de mémoire lors du chargement du moteur, qui ralentisse considérablement ce dernier, ou qui lise ou écrive une grande quantité de données. Nous verrons plus tard, à la section 4.2, comment nous avons composé avec ces contraintes, bien légitimes au demeurant.

3.3 Les données fournies par Druide

Pour chaque type de faute, Druide a préparé un ensemble d'environ 1000 exemples de détections et de corrections produites par leur correcteur (voir tableau 3.1 pour des statistiques détaillées). Ces exemples constituent notre *corpus d'entraînement*. Chaque détection a été annotée par Mala Bergevin, linguiste-informaticienne chez Druide, comme étant correcte ou pas (surdétection). Les exemples ont été choisis par notre partenaire commercial de manière à contenir des textes représentatifs de différents utilisateurs de l'application. L'interface d'annotation a été créée par Simon Charest, informaticien. Cette interface est présentée à la figure 3.1, pour la confusion entre les mots `que` et `dont`.

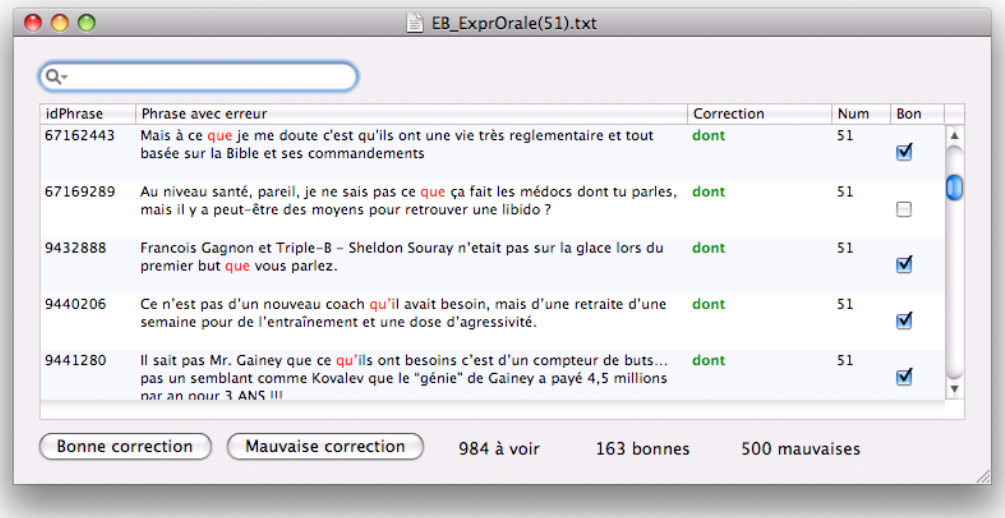


Figure 3.1 — Interface graphique d'annotation des détections d'Analytix

De plus, un document en format libre présentait pour chaque type de détection une analyse sommaire des contextes où l'annotatrice considérait que les surdétections se produisaient. Le minutieux travail de préparation des données a été décisif pour la bonne réalisation du projet.

Nous montrons ci-dessous, pour chacun des types de faute, une définition de celui-ci ainsi que des exemples de bonnes détections et de surdétections.

Dans ce mémoire, nous utiliserons systématiquement la convention qui suit pour distinguer les bonnes et les mauvaises détections. Le site de la détection est indiqué en **gras souligné**, et la correction, s'il y en a une, est proposée *[entre crochets]*, en italique, après la détection. Pour distinguer une bonne détection d'une surdétection, on utilisera la convention qui veut que l'astérisque dénote une incorrection linguistique, soit sur le site de la détection (si la détection est exacte), soit sur la correction proposée (si on a affaire à une surdétection).

Voici un exemple de détection légitime :

La Suède est le *__troisiemme__ *[troisième]* pays le plus grand d'Europe de l'Ouest.

Voici un exemple de surdétection :

Les comtés sont divisés en communes, ou __kommuner__ *[*commune]*.

Nous présentons à la section 3.3.1, pour chaque type de faute, une explication de la détection, ainsi que quelques exemples de détections légitimes ou pas, tirées des exemples d'entraînement fournis pas Druide. Nous présentons ces fautes en ordre alphabétique. Ces exemples numérotés seront utilisés tout au long de ce travail. Nous montrons ensuite, à la section 3.3.2, le format de ces données.

3.3.1 Types de détections

On présente ici les types de fautes étudiés. Lorsqu'un exemple contient plus d'une faute, seule la détection étudiée est soulignée.

ACCORD : faute d'accord

La détection ACCORD identifie les cas où le scripteur commet une faute d'accord (notamment en genre ou en nombre) entre deux entités linguistiques. C'est le cas classique d'échec d'unification, dont nous avons parlé à la section 2.2, qui se traduit par une détection.

ACCORD$_1$ Ils veulent un libéralisme VRAI, accepte __le__ *[*la]* marche mais refusent son truquage…

ACCORD$_2$ Cinq colonnes à la __une__ *[*un]* Pays d'origine

L'exemple ACCORD$_1$ contient plusieurs erreurs et mérite que nous en offrions une version corrigée pour faciliter la lecture : « `Ils veulent un libéralisme vrai, acceptent le marché, mais refusent son truquage…` ».

APOS : faute d'apostrophe

La détection APOS survient pour les fautes d'apostrophe. Elle signale soit l'omission d'une apostrophe ou encore sa présence erronée, par exemple dans une élision de mauvais aloi. La correction ajoutera ou supprimera donc l'apostrophe. APOS$_2$ est un exemple frappant de texte de très mauvaise qualité.

APOS$_1$ `Le gouvernement canadien est terriblement *`**`décu`**
`[*d'écu]` `et avec raison;`

APOS$_2$ *__salu__ *`[*s'alu]`* `bbcousin tu utilise koi comme stero et depuis quand?`

CONJUG : faute de conjugaison

La détection CONJUG signale les fautes de conjugaison des verbes. Elle concerne notamment les cas où le scripteur s'est trompé de nombre, de mode ou de temps. Pour CONJUG$_1$, le scripteur écrit `*inclue`, qui est en fait le subjonctif présent du verbe `inclure`, à la 3^e personne du singulier. Ce n'est manifestement pas son intention, et il y a fort à parier que c'est là le cas typique de confusion entre les formes `exclue` et `incluse`, où l'une interdit le `s` et pas l'autre.

CONJUG$_1$ `Il est clair que si l'on parle de mariage, la question de l'adoption est *`**`inclue`** *`[incluse]`* `!`

CONJUG$_2$ `Pour les majors, l'ennemi n'est plus le concurrent (peut-on encore parler de concurrence quand quatre sociétés se` **`partagent`** *`[*partageant]`* `le monde…`

Dans les données fournies, certaines fautes touchant la flexion du verbe sont signalées par les détections ER/EZ, MODE, PP/INF et PP/V.CONJ, permettant un degré plus fin de fonctionnement.

ÉLISION : faute d'élision

La détection ÉLISION montre au scripteur qu'il a mal utilisé l'élision. Celle-ci se définit comme le remplacement d'une voyelle finale (`a`, `e` ou `i`) par une apostrophe devant un mot qui commence par une voyelle, permettant l'euphonie. Cette détection n'est faite

dans nos exemples que lorsque l'élision est utilisée là où ce n'était pas nécessaire. Toutes les corrections proposent ainsi de restaurer la voyelle élidée.

ÉLISION$_1$ **M'bassidjé** *[*Me bassidjé]* François fut un Roi Abé.

ÉLISION$_2$ Ça ***s'** *[se]* peut vraiment.

ER/EZ : confusion entre les terminaisons `-er` et `-ez`

La détection ER/EZ révèle une confusion entre les terminaisons `-er` et `-ez`, pour les verbes dont la forme infinitive se termine par `-er`, une faute courante en français, puisque ces terminaisons sont phonétiquement équivalentes. Cette détection ne s'intéresse pas à la confusion entre des mots comme `cher` et `chez`, qui ne sont pas des verbes. Même si elle constitue une faute de conjugaison, elle a mérité une détection distincte de CONJUG, vue plus haut.

ER/EZ$_1$ **Regardez** *[*Regarder]* les yeux de Faber (interprété par Luchini), une histoire s'y déroule…

ER/EZ$_2$ Pour Fabrice **Ferrer** *[*Ferrez]* 4 octobre 2007 à 20:50 (CEST)

INVAR : changement erroné de forme pour un mot invariable

La détection INVAR décèle un changement de forme pour un mot qui devrait être invariable, ce qui est une incorrection. La correction, dans ce cas, est toujours le mot invariable, dépouillé de la flexion supposément incorrecte.

INVAR$_1$ Trois Casques bleus du Sri-Lanka avaient été blessés par ***balles** *[balle]* mercredi.

INVAR$_2$ deje pas beaucoup de marocain construise leur maison ici c vrai parce qu'il savent ce qui se passe et **la** *[*le]* plus part des maisons detruite sont a des francais

LA/LÀ : confusion entre `la` et `là`

La détection LA/LÀ identifie les cas où le scripteur a confondu l'article et pronom `la` avec l'adverbe et interjection `là`, sans doute à cause de leur homophonie. La correction est `la` lorsque `là` est détecté, et vice-versa. C'est donc un cas particulier des détections faites ici, puisqu'il n'y a que deux formes en jeu.

LA/LÀ$_1$ L'objectif est de rouvrir *__la__ *[là]* ou les coronaires bouchées ou en train de se boucher…

LA/LÀ$_2$ *__La__ *[Là]*, la france n'est pas mal lotie.

MAJ : faute de casse

La détection MAJ s'intéresse aux fautes dans la casse des mots. Elle impliquera donc souvent des entités nommées, ou des symboles.

MAJ$_1$ Tout le monde semble prendre le __pb__ *[*Pb]* au sérieux.

MAJ$_2$ 1574 : Tycho Brahé entreprend dans la petite île de __Ven__ *[*VEN]* (ou Hveen) près de Copenhague au Danemark, Uraniborg la construction du château…

MODE : faute de mode verbal

La détection MODE distingue les fautes d'usage des modes lors de la conjugaison d'un verbe. En d'autres termes, le correcteur rencontre une faute de conjugaison, mais confinée à une confusion sur le mode, comme dans la surdétection MODE$_1$, où la construction que je suis appelle correctement l'indicatif, et non le subjonctif suive, comme le propose le correcteur.

MODE$_1$ En résumé, et pour te dire Koko que je __suis__ *[*suive]* parfaitement en accords avec…

MODE$_2$ comment vont nolan et lucas? t'es tjrs op pour qu'on se *__voit__ *[voie]*?

OU/OÙ : confusion entre ou et où

La détection OU/OÙ identifie les fautes de confusion entre la conjonction de coordination ou et l'adverbe et pronom relatif où. Comme pour LA/LÀ, la faute procède de l'hésitation entre deux formes homophones.

OU/OÙ$_1$ Là *__ou__ *[où]* ça se gâte, c'est lorsqu'il commence à sauter…

OU/OÙ$_2$ __Où__ *[*Ou]* il y a le coq, la poule ne chante pas.

PP/INF : confusion entre le participe passé et l'infinitif

La détection PP/INF signale au scripteur les formes verbales pour lesquelles il a confondu l'infinitif d'un verbe en `-er` avec un participe passé dont la terminaison est phonétiquement identique, soit le son `/e/`.

PP/INF$_1$ `En ce sens, des engagements ont été pris, tels que` **développer** *[*développés]* `des technologies plus propres…`

PP/INF$_2$ `Elle nous apprend à ne plus *`**convoitée** *[convoiter]* `le bien d'autrui.`

PP INV : participe passé invariable

La détection PP INV dénote une confusion entre la forme fléchie et la forme invariable d'un participe passé. La correction consiste donc à restaurer la forme invariable du participe passé, soit à proposer une forme fléchie accordée correctement.

PP INV$_1$ `Par contre j'ai une préférence pour la mozza` **rapée** *[*rapé]*.

PP/V.CONJ : confusion entre un participe passé et le verbe autrement conjugué

La détection PP/V.CONJ souligne une faute où le scripteur a utilisé par erreur le participe passé d'un verbe là où il aurait plutôt dû utiliser une autre forme verbale du même verbe, ou vice-versa.

PP/V.CONJ$_1$ `Une fois installés,` **fini** *[*finirent]* `les soucis!`

PP/V.CONJ$_2$ `Par ce jeu de reflet, la forme devient évanescente, la matière` **confondue** *[*confondit]*.

QUE/DONT : confusion entre que et dont

La détection QUE/DONT identifie les cas où le scripteur aurait dû écrire `que` au lieu de `dont`, et vice-versa. La confusion est facile pour le scripteur inattentif : les deux mots sont des pronoms relatifs et la faute est répandue dans le langage parlé.

Cependant, le pronom relatif `dont` s'utilise exclusivement avec un verbe dont le complément est introduit par `de`. Comme pour LA/LÀ et OU/OÙ, il n'y a que deux formes en concurrence, mais elles ne sont pas homophones ici.

QUE/DONT$_1$	Je comprends ce que tu dis mais pas ce ***que** [dont]* tu parles.
QUE/DONT$_2$	Mais bon dieu **que** *[*dont]* les adultes s'amusent!
QUE/DONT$_3$	mange ce ***que** [dont]* tu as envie mais raisonablement prend 8 verres deau par jour bon resultat

3.3.2 Format des données d'entraînement et arbre d'analyse syntaxique

Chacune des phrases du corpus d'entraînement nous a été transmise sous la forme d'un arbre de dépendances syntaxiques sérialisé, accompagné d'une interface de programmation (API) C++ permettant sa désérialisation et son interprétation. Chaque arbre est annoté avec diverses informations, notamment les sites de détections et les corrections proposées. D'autres informations auxiliaires sont décrites à la section 4.4.

La figure 3.2 montre les dépendances syntaxiques isolées par le correcteur pour le tronçon de phrase `Ce fut très bien`. On constate que chaque dépendance entre deux mots est étiquetée avec la nature de cette dépendance syntaxique (par exemple, `fut` régissant `Ce` dans une relation de dépendance de type `sujet`).

Cet arbre d'analyse est accompagné de la détection et de sa correction, mais aussi d'éventuelles détections additionnelles, si plus d'une faute par phrase est signalée.

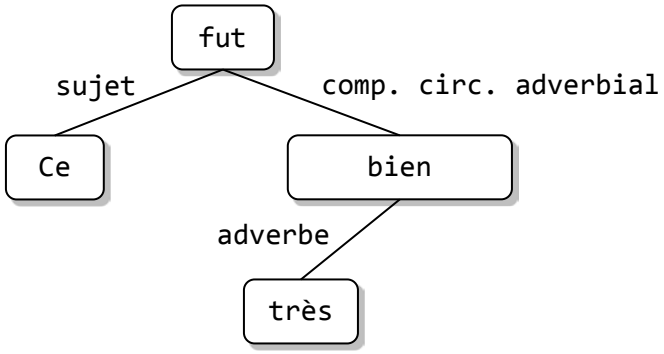


Figure 3.2 — Arbre de dépendances syntaxiques pour le tronçon `Ce fut très bien`

En fin de compte, l'arbre d'analyse contient les informations suivantes :

- La segmentation en mots et syntagmes (*tokenization*) : celle-ci peut regrouper plusieurs mots en une expression figée, telle que `plus ou moins`.

- La catégorie morphosyntaxique de chacun des mots : on en a dénombré 527 dans le corpus d'entraînement, ce qui est un nombre relativement grand. On y retrouve des catégories typiques, comme `NumOrd` (adjectif numéral ordinal), et certaines catégories très lexicalisées, par exemple `MoinsQuart`, comme dans `minuit moins quart`.
- Une ou plusieurs caractéristiques renseignant les entrées du lexique de l'analyseur. Il y en a près de 1000, touchant par exemple le régime d'un verbe. Certaines d'entre elles sont des attributs sémantiques des mots concernés.
- La relation de dépendance syntaxique que chaque mot subordonné entretient avec son régissant : on en a trouvé 290 types dans les exemples, ce qui est un grand nombre, ici aussi. Cette liste inclut le complément circonstanciel, le sujet, ou encore des relations plus sophistiquées et lexicalisées, comme la relation qui unit l'expression `ci-joint` au mot qu'elle qualifie. Lorsque l'analyseur a de la difficulté à faire une analyse complète d'une phrase, il lui est possible de se rabattre sur une relation spéciale « bidon » qui permet de regrouper des sous-arbres d'analyse sans pour autant se prononcer sur la relation les unissant.
- Une indication lorsqu'un mot est inconnu.
- Un score dénotant la qualité perçue du sous-arbre d'analyse dont la racine est le mot courant.

3.4 Portrait général des fautes étudiées

3.4.1 Observations générales sur les 14 types de fautes

Le corpus d'entraînement se caractérise avant toute chose par la grande hétérogénéité de ses exemples.

Primo, les fautes sont de natures très différentes, comme les exemples plus haut l'ont démontré. Certaines d'entre elles, très spécifiques, sont des confusions entre deux formes seulement (LA/LÀ , OU/OÙ, QUE/DONT) alors que d'autres sont très générales, comme ACCORD, qui pourrait survenir pour n'importe quel couple nom commun/adjectif du français, pour une faute de genre ou encore de nombre. CONJUG est également un type de faute aux manifestations extrêmement variables.

Secundo, les phrases soumises au correcteur afin de constituer le corpus d'entraînement appartiennent à des registres extrêmement variés. C'est la conséquence d'un effort stratégique de Druide d'y représenter les textes de leurs divers clients. On retrouve ainsi des phrases de Wikipédia dans un excellent français, aux côtés d'extraits typiques de la langue des messages SMS (voir les exemples INVAR$_2$ et MAJ$_1$), où les diacritiques manquent cruellement et où l'orthographe devient presque phonétique. D'autres extraits illustrent l'affligeante qualité du français employé sur les forums de discussion. D'autres encore sont des phrases tronquées, télégraphiques ou même des titres. Un exemple de la compétence douteuse du scripteur de certaines de ces phrases est INVAR$_2$, qui contient pas moins de douze fautes.

L'interaction entre les fautes est un problème en soi. Pour l'exemple ACCORD$_1$, la correction de `le` en `la` est liée au fait que le mot `marché` n'est pas accentué, confondant ainsi le correcteur.

Tertio, les fautes détectées sont caractéristiques de clientèles bien différentes. En effet, la plupart des francophones sont susceptibles de commettre une faute ACCORD ici et là, mais la faute QUE/DONT (p.ex. `la personne *que je parle`) est peu probable dans le texte d'un scripteur natif et compétent.

Cette grande variété est digne de mention, dans la mesure où des manifestations très variables de détections posent un défi à l'induction de règles prédictives : à l'extrême, si chaque exemple est un cas particulier sans rapport avec les autres, alors l'induction de règles couvrant d'autres cas est impossible, par définition.

3.4.2 Statistiques du corpus d'entraînement

Le tableau 3.1 présente notamment le nombre d'exemples, le nombre de surdétections et détections correctes pour chacune des fautes étudiées.

Druide nous a fourni un nombre d'exemples qui varie peu de faute en faute. Il s'élève en moyenne à 1250 phrases. Pour les détections QUE/DONT et PP/INF, nous avons produit nous-mêmes quelques centaines d'annotations supplémentaires, car des résultats préliminaires semblaient indiquer que leur nombre restreint diminuait la qualité de nos résultats.

Tableau 3.1 — Description du corpus d'entraînement utilisé

Type de faute	Nombre d'exemples	Nb moyen de mots par phrase	% surdétec-tions	Nb moyen de détections par phrase
ACCORD	1506	35,5	45%	3,8
APOS	1086	35,3	46%	6,3
CONJUG	1325	25,6	34%	2,7
ÉLISION	997	27,9	63%	3,4
ER/EZ	1600	22,0	49%	2,2
INVAR	1119	33,9	45%	4,4
LA/LÀ	1249	19,3	50%	2,2
MAJ	1012	31,0	50%	3,7
MODE	1517	29,7	62%	2,6
OU/OÙ	1241	25,8	40%	3,1
PP/INF	1157	29,8	49%	3,5
PP INV	1825	34,8	22%	3,9
PP/V.CONJ	1155	26,3	43%	2,3
QUE/DONT	732	26,5	71%	2,5

Nous savons de nos discussions informelles avec Druide que la haute proportion de surdétections (voir tableau 3.1) n'est pas représentative de la distribution des détections durant la marche habituelle d'Analytix, pour un texte donné. Il est même difficile d'imaginer pareil scénario : le correcteur en serait lourdement handicapé. Au contraire, les surdétections sont surreprésentées dans le corpus d'entraînement à dessein. En effet, cette situation artificielle nous permet d'étudier aussi bien les caractéristiques et les contextes d'occurrence des détections légitimes que ceux des surdétections.

Le nombre moyen de détections par phrase semble corroborer nos observations sur la mauvaise qualité de certains des textes dans le corpus d'entraînement.

Chapitre 4
Approche par apprentissage machine

La tâche du projet Scorali en est une de classification binaire : il s'agit en effet de déterminer si une détection donnée est légitime ou, au contraire, si c'est une surdétection. Nous détenons par ailleurs des exemples annotés de détections justes et de surdétections, décrits au chapitre précédent.

Il est donc tout naturel de nous tourner vers une technique d'apprentissage machine supervisé. Celle-ci vise la création de *classificateurs*[1], soit des fonctions qui permettent de sélectionner une classe (une étiquette, par exemple) à partir d'entrées. Ces fonctions sont le fruit d'un *entraînement*, c'est-à-dire l'apprentissage à partir d'un ensemble de données correctement étiquetées par un *expert*. Idéalement, le classificateur ainsi créé est en mesure d'imiter les décisions de l'expert.

L'approche est particulièrement indiquée dans notre effort d'amélioration du correcteur, car nous nous situons en aval de la chaîne de détection, sans accès facile au fonctionnement interne d'Analytix. Il nous est donc possible de créer des filtres de post-traitement.

[1] Il nous a été difficile de trancher entre « classifieur » et « classificateur », car les deux existent dans la littérature du domaine. Nous optons pour le deuxième terme, ne serait-ce que pour éviter que notre correcteur grammatical nous signale que « classifieur » est un mot erroné…

D'un point de vue commercial, cela simplifie le développement, car il ne nous est pas nécessaire de nous familiariser avec le code complet du moteur de correction grammaticale, et encore moins d'y opérer de délicates modifications, soumises aux validations du cycle de développement adopté par Druide.

4.1 Les sorties du classificateur : terminologie

Dans ce travail, les classificateurs tranchent entre deux classes : surdétection ou détection légitime (abusivement « mauvaise » et « bonne » détection, respectivement). La classe que nous cherchons à identifier est la surdétection. Or, un classificateur peut se tromper dans son identification de cette classe. Le tableau 4.1 (une *matrice de confusion*) récapitule les sorties possibles d'un classificateur pour le cas qui nous intéresse et clarifie les notions de *vrai et faux positifs*, dans le cadre de ce travail. Nous jugeons que cette présentation est utile pour clarifier notre propos.

Tableau 4.1 — Matrice de confusion appliquée à notre étude des surdétections

		Sortie du classificateur	
		Surdétection	**Détection légitime**
Condition réelle	**Surdétection**	Vrai positif	Faux négatif (erreur de type II)
	Détection légitime	Faux positif (erreur de type I)	Vrai négatif

Le *taux de faux positifs* (FP) est le pourcentage de détections légitimes erronément classées comme surdétections. Un haut taux de FP signifie un moins bon rappel des fautes commises par le scripteur. On a vu à la section 3.2 que le cahier des charges bornait supérieurement cette statistique à 10 % dans la sélection d'un classificateur.

Le *taux de faux négatifs* (FN) est le pourcentage de surdétections erronément classées comme détections légitimes. Un haut taux de FN expose l'utilisateur à beaucoup de surdétections. Le *taux de vrais positifs* (VP) est relié au précédent; il vaut $\mathrm{VP} = 1 - \mathrm{FN}$. Le cahier des charges de la section 3.2 stipulait que $\mathrm{VP} \geq 66\ \%$ pour qu'un classificateur soit acceptable.

Dans ce travail, on utilisera également comme variable synthétique de la performance d'un classificateur la *F-mesure*, soit la moyenne harmonique de sa précision P et de son rappel R, selon la formule $F = 2 \cdot (P \cdot R)/(P + R)$, où $P = VP/(VP + FP)$ et $R = VP/(VP + FN)$.

4.2 Méthodologie : deux protocoles expérimentaux distincts

La chaîne de traitement classique pour le développement de classificateurs est expliquée ci-dessous.

1. Création des données d'entraînement. Colligées par Druide, ces données sont décrites à la section 3.3.
2. Ingénierie et extraction de *traits*[1]. L'approche d'apprentissage supervisé requiert l'identification de traits (*features*) pour chaque détection. Ces traits sont des attributs attachés aux instances, et peuvent être très variés. Ce sont eux qui servent à discriminer une instance donnée, et leur choix est donc à la fois critique et délicat.
3. Sélection et entraînement d'un classificateur. Bien souvent, une foule de classificateurs sont envisageables pour un problème donné, et il importe de choisir ceux qui correspondent aux objectifs de complexité et de performance visés. Une fois ce choix fait, il faut entraîner le classificateur sur les instances d'entraînement, de façon à en fixer les paramètres.
4. Évaluation du ou des classificateurs sur les données de test. Il s'agit de soumettre le classificateur élaboré précédemment sur des données de test et de l'évaluer. Dans notre cas, il faut que les taux de faux et de vrais positifs correspondent aux exigences du cahier des charges.

Les étapes 2 et 3 décrites plus haut sont tributaires du cahier des charges, décrit à la section 3.2. L'extraction des traits ne peut donc pas faire usage de processus ou de données informatiques qui alourdiraient considérablement le correcteur. De plus, il nous apparaît préférable, d'un point de vue commercial, de sélectionner des classificateurs dont la mise en œuvre (*implementation*) et l'interprétation sont faciles, afin de faciliter le portage du laboratoire vers l'entreprise.

---

[1] Là encore, la traduction du mot anglais *feature* est ambiguë en français. Termium® (http://www.termiumplus.gc.ca/) propose `attribut`, `caractéristique` et `propriété`. Nous leur préférons `trait`, qui est équivalent et moins ambigu dans ce travail, à notre avis.

Cependant, par souci d'exhaustivité et par curiosité scientifique, nous souhaitons également que cette étude couvre des traits dérivés de modèles plus exigeants et utilise des familles de classificateurs plus complexes que ceux évoqués plus haut.

C'est pour ces raisons que notre méthodologie est double : nous proposons un sous-projet ScoraliA pour la méthodologie visant l'entreprise et un projet ScoraliB, prolongeant le premier. Nous présenterons donc deux protocoles expérimentaux (apparentés) et deux ensembles de résultats.

4.3 Utilisation de Weka pour le développement des classificateurs

Nous avons fait appel au logiciel libre Weka[1], version 3.6.0 (Hall *et al.*, 2009), écrit en Java, pour mener à bien l'ingénierie et la sélection des traits, de même que l'étude et la création des classificateurs. Le logiciel, muni d'une interface graphique de grande qualité, permet d'explorer facilement de nombreuses familles de classificateurs et possède des fonctions intéressantes, tels la visualisation et le prétraitement des données d'entraînement. Il est enfin possible de contourner l'interface graphique et de lancer un classificateur à partir de la ligne de commande pour effectuer du traitement par lots (*batch processing*), ce qui s'est avéré indispensable dans notre cas, puisque nous avons exploré des milliers de configurations de classificateurs.

4.4 Ingénierie et extraction des traits (*features*)

Nous avons effectué l'extraction des traits en écrivant un module C++ s'appuyant sur l'API de programmation fournie par Druide pour désérialiser les données annotées (voir section 3.3.2). Ceci a été fait sur Mac OS X 10.5.

Il nous apparaît utile de diviser les traits extraits en deux catégories : ceux qui touchent la détection elle-même et ceux décrivant son contexte. Il est clair, en effet, que ce dernier a un rôle crucial dans la classification d'une détection donnée. Le voisinage d'une détection peut être simplement les mots précédant ou suivant le site de la détection ou, puisque le matériel d'entraînement inclut l'arbre d'analyse syntaxique, les mots régissant ou subordonnés dans cet arbre d'analyse. L'exploration du contexte est conforme à ce que bien d'autres chercheurs concluent, parmi lesquels Golding & Roth (1998). Cette dernière étude porte sur la correction de substitutions orthographiques fautives telles que ***causal** *[casual]* et s'intéresse à la présence de mots

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

spécifiques dans une fenêtre de 10 mots autour de la détection, et à la présence de collocations modélisant le contexte plus immédiat. Des correcteurs pour des langues plus éloignées font usage de techniques similaires. Par exemple, Young-Soog (1998) corrige le coréen en étudiant les mots dans le voisinage de la détection ainsi que les mots régissant et subordonnés aux mots de la détection, à l'aide d'un analyseur de dépendances syntaxiques. Il nous est utile d'imiter nos prédécesseurs en cela.

Par exemple, dans la confusion OU/OÙ, il est manifeste que si le mot `là` précède la confusion, c'est presque toujours le mot `où` qui est attendu ensuite. De même, pour la confusion QUE/DONT, un parent qui est un verbe transitif direct appellera le mot `que`, alors qu'un verbe transitif indirect appellera le `dont`. Pour ce dernier exemple, on pourrait s'interroger : à quoi bon refaire ici le travail du correcteur ? En effet, s'il a identifié le parent de QUE/DONT, n'est-il pas en mesure de faire une détection toujours correcte alors ? C'est sans compter les erreurs d'analyse qui perturbent l'identification des dépendances syntaxiques. Or, une partie de nos traits tentent de pallier ceci en s'intéressant au contexte positionnel du site de la détection, sans tenir compte de l'analyse syntaxique. On pourra ainsi découvrir dans ce voisinage un mot qui renseignera sur la validité de la détection effectuée, afin de la classer correctement.

Le voisinage n'est pas nécessairement lexicalisé. L'exemple MAJ_2 montre qu'il est peu indiqué de corriger un mot ayant une capitale lorsqu'il suit un autre mot avec une capitale : on est vraisemblablement en présence d'une entité nommée.

En tout, nous extrayons environ 1200 traits décrivant la détection et les mots de son voisinage (section 4.4.1). À ceux-là, nous ajoutons environ 50 traits issus de la désambiguïsation sémantique (voir section 4.4.2). Pour ScoraliB, nous explorons également environ 50 traits dérivés de modèles de langue (section 4.4.3). Dans la mesure où nous ne nous intéressons qu'à un seul site dans la phrase, ces traits sont comparativement nombreux, en grande partie grâce à la richesse des informations produites par Analytix.

Il n'en demeure pas moins que le nombre de traits utilisés dans cette étude peut sembler modeste lorsque comparé à ceux dérivés d'une phrase complète. Par exemple,

dans le projet Alpino, van Noord (2007) décrit une approche de reclassement (*reranking*) exploitant 42 000 traits.

4.4.1 Traits décrivant la détection et les mots de son voisinage

Fort des observations faites ci-dessus, nous avons extrait les traits suivants, qui sont partagés par ScoraliA et ScoraliB.

- Les traits de la phrase dans laquelle on trouve la détection: sa longueur en mots, le nombre de relations de dépendance syntaxique, le nombre de mots impliqués dans des détections, le nombre de mots inconnus, de relations « bi-don » (voir section 3.3.2)[1].
- Les traits du mot au site de la détection, soit sa casse, sa longueur en lettres, son genre, son nombre (dans le cas d'un nom), ses mode, temps, nombre et personne (dans le cas d'un verbe), sa catégorie morphosyntaxique, sa position dans la phrase et les caractéristiques qui renseignent son entrée dans le lexique du correcteur (voir à nouveau la section 3.3.2).
- Les mêmes traits que ceux du point précédent, mais cette fois pour les mots précédant et suivant le mot de la détection, ainsi que pour son mot parent dans l'arbre d'analyse syntaxique. Si le mot n'a pas de régissant, ces derniers traits sont considérés manquants.
- Les traits de la détection elle-même : la certitude avec laquelle le correcteur la propose, le nombre d'autres détections au même site, ainsi que les traits de la correction proposée pour remplacer le mot fautif.
- La nature des relations syntaxiques auxquelles le mot de la détection appartient, c'est-à-dire les relations qu'il entretient avec son parent et ses enfants éventuels dans l'arbre d'analyse.

Tous les traits mentionnés plus tôt sont communs à toutes les détections. Nous en avons conçu quelques-uns qui reflètent nos intuitions quant aux particularités de certaines d'entre elles. Par exemple, pour la détection PP/INF, la présence de crochets autour du mot semblait confondre le correcteur, par conséquent, un trait capture cette situation. Pour PP/INF, nous tenons également compte de la présence des verbes `pouvoir` et `faire` avant le site de la détection.

[1] Notons que la plupart des traits qui sont des dénombrements sont doubles : pour un trait qui est le compte, il en existe un autre qui est le compte normalisé par la longueur de la phrase.

4.4.2 Traits de désambiguïsation sémantique

Pour la confusion QUE/DONT, il nous a également paru intéressant de reformuler une partie de notre problème de classification comme une application de la désambiguïsation sémantique (*word sense disambiguation* ou WSD), et ce, à la fois dans le cadre de ScoraliA et de ScoraliB. En effet, on peut voir le site de la détection comme un mot fictif inconnu et $S \equiv \{\texttt{que}, \texttt{dont}\}$ comme l'ensemble des étiquettes « sémantiques » potentielles. La désambiguïsation permet alors de savoir laquelle des étiquettes $s \in S$ doit être insérée à l'endroit de l'ambiguïté.

Il existe plusieurs techniques de désambiguïsation sémantique. Manning & Schütze (1999) en expliquent les plus répandues : approches par apprentissage supervisé et non supervisé (*clustering*), ainsi que celles basées sur des dictionnaires. Dans notre cas, l'apprentissage supervisé semble une solution naturelle, car il est facile de trouver des corpus d'entraînement, où la « désambiguïsation » vers `que` ou `dont` a déjà été faite correctement, par des humains (voir plus bas). Les approches par dictionnaires, quant à elles, s'appuient notamment sur les sens des étiquettes sémantiques, tels que répertoriés dans des dictionnaires ou des thésaurus. Ces approches s'appliquent difficilement à notre problème, car `que` et `dont` sont des mots-outils, et leurs définition ou synonymes sont peu discriminants.

Nous appliquons cette approche en ajoutant des traits qui explorent le contexte de la confusion et le comparent aux contextes dans lesquels on retrouve habituellement `que` et `dont`, dans des textes réputés sans fautes. Pour l'exemple QUE/DONT$_1$ (p. 30), la présence du verbe `parler` aux abords de la confusion pourrait ainsi contribuer à indiquer qu'un `dont` est plus probable qu'un `que`.

Nous basons nos traits sur une adaptation de la technique de Yarowsky (1995), décrite par Manning & Schütze (1999). Conformément à cette dernière, on s'intéresse aux probabilités de collocation $P(s|C)$ pour le ou les mots du contexte C, au site de la détection. Le ratio

$$r = \frac{P(\texttt{que}|C)}{P(\texttt{dont}|C)}$$

est ensuite calculé.

On peut dès lors produire des traits binaires : on privilégie le mot `que` ssi $r > 1$, `dont` sinon. On inclut également r dans les traits, pour alimenter les classificateurs utilisant seulement des traits aux valeurs continues.

Il est également possible de créer des traits plus discriminants, en utilisant les valeurs booléennes indicatrices $v_t \equiv r \geq t$ pour une dizaine de seuils t fixés empiriquement entre 2 et 50. Si v_t es vraie pour une grande valeur de t, alors le mot `que` est favorisé.

L'approche de Yarowsky (1995) permet des rectifications facultatives de ces calculs pour forcer l'homogénéité du sens d'un mot ambigu au sein du même document. Notre cas n'autorise pas ces perfectionnements, puisque le document où l'on fait la désambiguïsation se résume à une seule phrase.

Expliquons maintenant ce que nous entendons par le contexte C et comment nous calculons les probabilités de la collocation $P(s|C)$.

Pour une partie des traits, nous définissons C simplement comme le mot qui précède la détection. Un site de détection en position i concernant le mot w_i dans une phrase de n mots $w_0 \dots w_i \dots w_n$ explorera donc les probabilités $P(s|w_{i-1})$ pour les étiquettes s. Pour d'autres traits encore, nous considérons C comme les mots de fenêtres qui peuvent être $w_{i-5} \dots w_{i-1}$ ou encore $w_{i+1} \dots w_{i+5}$. Nous avons utilisé des fenêtres plus grandes encore : $w_0 \dots w_{i-1}$ et $w_{i+1} \dots w_n$. Nous utilisons de plus une variante de tous ces contextes à plusieurs mots, en ignorant les mots grammaticaux (*stopwords*) qui pourraient s'y trouver et qui sont des indicateurs pauvres du contexte, car ubiquistes.

Dans les cas où C contient plusieurs mots w_i, on cherche l'étiquette $\hat{s}$ la plus probable

$$\hat{s}, \hat{\imath} = \underset{s \in S, w_i \in C}{\operatorname{argmax}} P(s|w_i)$$

conformément à l'approche de Yarowsky, qui se fie à une et une seule collocation pour trancher entre plusieurs étiquettes sémantiques, soit la collocation la plus probable qui maximise la probabilité $P(s|w_i)$, à l'exclusion de toute autre.

Nous avons précalculé $P(s|w_i)$ en nous basant sur la version française du Hansard canadien[1]. Pour chacune des étiquettes `que` et `dont`, nous avons extrait un corpus de 40 000 phrases les contenant, choisies aléatoirement. La probabilité $P(s|w_i)$ y correspond à la fréquence relative de la collocation de s avec w_i par rapport à la collocation de s avec tous les mots du contexte. Deux mots spéciaux BOS et EOS ont été ajoutés respectivement au début et à la fin de la phrase pour dénoter ces deux événements statistiques linguistiquement importants. Ainsi, lorsque $C = w_{i+1} \dots w_{i+5}$, les 3 premiers mots w_i qui maximisent $P(\text{dont}|w_i)$ sont EOS ($P = 0{,}0173$), `besoin` ($P = 0{,}0160$) et `saisie` ($P = 0{,}00877$), ce qui est raisonnable.

Ces modèles statistiques calculés à l'avance ont le mérite d'être relativement compacts, une qualité essentielle selon le cahier des charges. En effet, chacun consiste en une collection de paires $(w, |w|)$ associant un mot w à sa fréquence $|w|$ de collocation avec `que` ou `dont`. Si l'on élimine les mots peu fréquents pour leur donner une fréquence basse arbitraire ε, la plupart des modèles ne contiennent que 5 000 paires $(w, |w|)$. En discrétisant la fréquence sur 1 octet et en encodant chaque mot sur 2 octets[2], chaque modèle pourrait ne prendre que 30 ko. Puisque la stratégie décrite plus haut requiert une dizaine de ces modèles, le tout occuperait environ 300 ko. Naturellement, si nous avions eu plus d'étiquettes « sémantiques », l'approche aurait été d'autant plus lourde. Cette stratégie n'est donc pas applicable à des confusions impliquant un grand nombre de formes, par exemple toutes les flexions d'un verbe donné.

Nous n'avons pas étendu l'approche aux confusions LA/LÀ et OU/OÙ à cause de résultats préliminaires mitigés.

4.4.3 ScoraliB : traits additionnels issus des modèles de langue

On l'a vu à la section 2.2, certains correcteurs statistiques s'appuient exclusivement sur des phénomènes de surface : ils vérifient dans quelle mesure une séquence de mots est plausible dans une langue donnée. Si elle est improbable, alors elle pourrait être le site d'une faute. D'une certaine manière, cela rejoint le travail que nous avons

[1] Ces débats, disponibles sur le Web (http://www.parl.gc.ca/common/chamber.asp), sont couramment utilisés dans le cadre des activités de notre laboratoire. L'auteur a utilisé une des copies qu'il a préparées dans le cadre d'un autre projet.
[2] Un code à longueur variable, comme le codage de Huffman, pourrait réduire cette taille.

décrit à la section précédente : l'intuition veut que la séquence `ce que tu parles` est moins fréquente que `ce dont tu parles` dans un corpus de qualité[1].

Un *modèle de langue* statistique est l'outil que les praticiens du traitement des langues utilisent pour résoudre ce type de problème. Succinctement, c'est un modèle statistique prédictif qui permet, étant donné un début de phrase (une ébauche), de connaître la probabilité qu'un mot donné vienne poursuivre l'ébauche de phrase. Un modèle de langue est donc capable, après avoir été entraîné sur un corpus volumineux contenant des exemples de phrases correctes, de déterminer quel mot est le plus probable dans un contexte donné.

Dans notre cas, nous avons entraîné un modèle de langue trigramme, c'est-à-dire permettant d'estimer la probabilité $P(w_i|w_{i-2}, w_{i-1})$ pour un mot donné w_i. Nous l'avons entraîné sur 3,3 M de phrases françaises d'une tranche du Hansard canadien, qui concerne les débats effectués entre 1986 et 1994. L'entraînement et l'évaluation ont été effectués à l'aide de la boîte à outils SRILM[2] (Stolcke, 2002). Le modèle de langue utilise un lissage par interpolation de type Kneser-Ney, basé sur des modèles d'ordre inférieur. Nous avons également écrit un programme utilisant l'API de SRILM pour les besoins de ce travail.

Soit w_i le mot au site de la détection et w_i' la détection proposée par le correcteur. Il nous est possible de créer les traits correspondant aux probabilités $P(w_i|w_{i-2}, w_{i-1})$ et $P(w_i'|w_{i-2}, w_{i-1})$. Cela revient à calculer les plausibilités respectives de la séquence qui fait l'objet de la détection et de celle qui est proposée par le correcteur, selon un modèle de langue donné.

Dans la même veine, nous avons également renversé le modèle de langue afin de pouvoir estimer les probabilités $P(w_i|w_{i+2}, w_{i+1})$ et $P(w_i'|w_{i+2}, w_{i+1})$. Pour ce faire, il suffit d'entraîner un modèle de langue sur le même corpus, mais après en avoir renversé l'ordre des mots. Le calcul de ces probabilités permet de tenir compte du contexte qui suit le site de la détection.

---

[1] De fait, notre version du corpus Hansard, recensant 25 ans de débats parlementaires canadiens, contient 21 fois la séquence `ce qu'il parle` (comme dans `je dois m'opposer à ce qu'il parle`) et 72 fois `ce dont il parle`.

[2] http://www-speech.sri.com/projects/srilm/

Naturellement, nous aurions pu également utiliser notre modèle de langue pour tenir compte tout à la fois du contexte gauche et droit[1] du mot w_i en calculant $P(w_i|w_{i+2}, w_{i+1,}w_{i-2}, w_{i-1})$ par normalisation de

$$P\left(w_{i+2}, w_{i+1,}w_i\middle|w_{i-2}, w_{i-1}\right) \propto P(w_{i+2}|w_i, w_{i+1}) \cdot P(w_{i+1}|w_{i-1}, w_i) \cdot P(w_i|w_{i-2}, w_{i-1}).$$

Les traits utilisés ici ne sont envisageables que dans le cadre de ScoraliB. En effet, ils exigent chacun quelque 500 Mo sur disque, soit 1 Go au total. Ces modèles ont une taille comparable en mémoire vive, et prennent chacun 20 s à charger sur un ordinateur cadencé à 2 GHz.

4.5 Sélection des classificateurs

Pour chacun des 14 types de fautes, nous avons mis au point un classificateur différent. Ceci nous a permis d'isoler chaque faute comme un problème distinct, et de diagnostiquer séparément les problèmes rencontrés. De plus, comme on l'a vu à la section précédente, ce cloisonnement autorise la conception de traits spécifiques à une détection particulière.

Weka permet de prototyper une cinquantaine de classificateurs, regroupés en 8 familles, incluant les classificateurs bayésiens, les arbres de décision, les perceptrons, les séparateurs à vaste marge (SVM, *support vector machines*) et les méta-classificateurs. Ces derniers combinent d'autres classificateurs, par exemple en les faisant voter. Ces classificateurs peuvent être contrôlés par 1 à 20 hyperparamètres discrets ou continus, engendrant chacun des variantes distinctes du même algorithme. Ceci engendre une explosion combinatoire du nombre de classificateurs possibles. Ici encore, pour les raisons avancées à la section 4.2, les projets ScoraliA et ScoraliB diffèrent.

4.5.1 ScoraliA : classificateurs symboliques

Pour ScoraliA, nous avons concentré nos efforts sur des classificateurs symboliques, produisant des règles ou des arbres de décision distinguant les bonnes des mauvaises détections. Le tableau 4.2 qui suit donne un aperçu de la gamme de classificateurs étudiés. Le lecteur pourra consulter la documentation de Weka pour plus de détails à leur sujet.

[1] Cette idée nous a été aimablement suggérée par Yoshua Bengio.

Tableau 4.2 — Classificateurs explorés pour le projet ScoraliA

Nom du classificateur Weka	Description sommaire
`rules.ConjunctiveRule`	Apprentissage de règles conjonctives
`rules.DecisionTable`	Table de décision
`rules.JRip`	Apprentissage de règles prédictives de type RIPPER (Cohen, 1995)
`trees.ADTree`	Arbre de décision alternatif
`trees.DecisionStump`	Arbre de décision de profondeur 1
`trees.J48`	Équivalent de l'algorithme C4.5

Ces classificateurs s'appuient sur des réponses à une suite de questions fermées et sont bien adaptés à notre tâche. En effet, comme l'expliquent Duda *et al.* (2001) pour les arbres de décision, ces classificateurs sont très facilement interprétables. Une fois construit, un arbre de décision contient pour chaque classe sa description logique. Par exemple, un tel classificateur pourrait montrer qu'une surdétection de type x survient systématiquement lorsqu'un mot w_x en précède le site et qu'une étiquette morpho-syntaxique m_x le suit. Ceci simplifie non seulement la compréhension de l'algorithme par un humain, mais permet de plus le raffinement ou l'ajout de tests logiques au cœur de l'algorithme, si l'on désire y ajouter des connaissances supplémentaires. Il est enfin utile de noter que ces classificateurs sont extrêmement rapides.

4.5.2 ScoraliB : séparateurs à vaste marge (SVM)

Les SVM constituent une famille de classificateurs évolués qui fonctionnent en représentant les données dans un espace vectoriel en haute dimension, puis en tentant de trouver un hyperplan qui sépare le mieux possible les deux classes dans cet espace.

Nous les avons privilégiés pour ScoraliB, car ces classificateurs s'illustrent dans la littérature du traitement des langues naturelles, notamment par leur stabilité face aux disparités entre corpus d'entraînement et de test. Malheureusement, ces avantages se gagnent au prix de difficultés dans l'exportation du classificateur vers du code industriel, étant donné leur complexité relativement plus grande que les classificateurs vus à la section précédente.

Le classificateur SVM utilisé est entraîné en utilisant l'algorithme d'optimisation séquentielle minimale de Platt (1998). L'implémentation de Weka gère les traits manquants et transforme également les attributs discrets en attributs aux valeurs continues. Un attribut discret prenant k valeurs est transformé en $k - 1$ nouveaux

attributs binaires. Nous avons testé plusieurs noyaux de classificateurs, dont les noyaux polynomiaux (*polynomial kernel*) de degrés 1, 2 et 3 et le noyau de fonction à base radiale (*RBF*).

4.6 Protocole expérimental

En fin de compte, le cœur de notre méthodologie a consisté à explorer les performances des classificateurs mentionnés plus haut, sur les instances fournies par Druide, après extraction des traits décrits. Nous avons complété l'exploration de ces réglages en ajoutant deux variables supplémentaires, soit le type de prétraitement des instances et l'utilisation d'une matrice de coût pour guider l'entraînement du classificateur.

4.6.1 Prétraitement des traits des données d'entraînement

Le prétraitement des traits extraits des données d'entraînement peut avoir plusieurs avantages. Dans notre cas, il aura permis avant toute chose d'adapter les traits aux classificateurs. Certains d'entre eux, on l'a vu pour les SVM, exigent des traits aux valeurs continues, alors que d'autres fonctionnent mieux avec des traits aux valeurs discrètes. Nous avons utilisé les suggestions de Weka, dont les choix sont très raisonnables.

Nous avons également prétraité les traits pour en réduire le nombre, en filtrant tous ceux qui étaient constants pour toutes les instances. Dans le cas des traits à valeur discrètes, nous avons commencé par éliminer tous les traits pour lesquels le ratio entre le nombre de valeurs distinctes et le nombre total d'instances excédait 99 %. Ce filtre nous a permis de réduire le nombre de traits d'environ 1200 à 400, avec quelques variations entre types de fautes.

Ce filtrage permet presque toujours d'accélérer l'entraînement des classificateurs et (souvent) de produire des classificateurs dont les règles impliquent moins de traits, donc simplifiés, ce qui fait partie de nos objectifs.

Nous avons donc poussé plus loin ce filtrage, en créant une configuration de prétraitement de plus, utilisant un filtre `weka.attributeSelection.CfsSubsetEval`, éliminant les traits qui semblent hautement corrélés entre eux. Parmi les dizaines de filtres proposés par Weka, celui-là nous semblait le plus agressif, tout en conservant les traits qui, empiriquement seulement, nous semblaient saillants pour chaque type

de faute. Pour les SVM, on court le risque de supprimer des traits qui semblent corrélés dans leur espace vectoriel d'origine, alors qu'ils ne le sont pas dans l'espace dans lequel on les projette. C'est une raison de plus pour laquelle nous conservons les traits avant et après ce type de filtrage.

Nous possédons en fin de compte deux ensembles de traits : l'un peu filtré, l'autre agressivement filtré. Nous n'avons pas filtré les instances, jugeant leur nombre déjà trop réduit.

4.6.2 Entraînement des classificateurs et matrice de coût

Étant donné le nombre restreint d'instances dans le corpus d'entraînement pour une faute donnée, de même que l'absence de corpus de test aisément accessible, nous avons recouru à la validation croisée en k blocs[1] (avec $k = 10$ blocs de tailles égales), avec répartition aléatoire des instances dans chaque bloc.

Lors de l'entraînement, la fonction de coût utilisée a été l'erreur de classification, pondérée par une matrice de coût. L'erreur de classification est le pourcentage d'instances mal classifiées. Quant à elles, les *matrices de coûts* ont été essentielles au développement de classificateurs dans Weka. Elles permettent de spécifier des coûts pour chaque type d'erreur (I et II, voir le tableau 4.1). Ainsi doté d'une matrice de coût, un classificateur sélectionnera la classe dont le coût d'erreur de classification est le moins élevé. En jouant sur les coûts stockés dans la matrice, il nous est alors devenu possible de créer des classificateurs dont les décisions sont guidées afin de satisfaire les exigences du cahier des charges en termes de vrais positifs et faux négatifs (voir section 3.2). Nous avons expérimenté avec des coûts pour les deux types d'erreurs (I et II) variant de 1 (le coût par défaut) à 30.

Les matrices de coût ne peuvent être couplées à tous les classificateurs, nous avons donc pris soin de choisir des classificateurs compatibles pour ScoraliA et ScoraliB.

4.6.3 Résumé des configurations différentes

Somme toute, en tenant compte des prétraitements différents des traits, des classificateurs variés (et de leurs réglages respectifs), ainsi que des matrices de coûts, chaque type de faute a demandé l'entraînement de nombreux classificateurs, afin d'explorer une partie appréciable de l'espace de recherche ainsi créé.

---

[1] k-*fold cross-validation*

Pour ScoraliA, nous avons entraîné et évalué 2300 classificateurs.

Pour ScoraliB, nous avons exploré les mêmes 2300 classificateurs, cette fois-ci avec les traits additionnels issus des modèles de langue décrits à la section 4.4.3. Les classificateurs de type SVM, quant à eux, sont au nombre de 600, et nous les avons testés dans un premier temps sur les mêmes traits que ScoraliA, et dans un deuxième temps sur les traits de ScoraliA additionnés des traits des modèles de langue. Cela fait donc 3 configurations distinctes pour ScoraliB. Cela nous a permis de comparer l'apport des SVM et des traits associés aux modèles de langue indépendamment, *ceteris paribus*.

Pour ScoraliA, il a fallu dépouiller les résultats des expériences (voir section 5.1) pour suggérer un classificateur satisfaisant les besoins de Druide Informatique.

4.6.4 Environnement technique d'expérimentation

Nous avons lancé les entraînements sur une grappe (*cluster*) de 32 ordinateurs faisant tourner GNU/Linux. Les deux processeurs de chacune des machines étaient cadencés à 3 GHz et nécessitaient 2 Go de mémoire vive pour accommoder les deux instances de Weka que nous avons fait tourner sur chaque machine. Nous avons lancé les tâches d'entraînement en utilisant un logiciel qui répartissait automatiquement les processus sur la grappe de calcul, selon la charge de chaque machine.

L'entraînement et l'évaluation de chaque classificateur a pris entre quelques secondes (pour les arbres de décision) jusqu'à quelques minutes (pour les SVM). Pour chaque type de faute, l'évaluation de ScoraliA prenait 2 h (soit environ une journée pour les 14 types de fautes) et 3 h pour ScoraliB (pour un total d'un jour et demi de calcul environ).

Les milliers de fichiers de sortie de Weka sont ensuite analysés pour en extraire les statistiques de performance, les plus importantes étant les taux de vrais et faux positifs et les vrais et faux négatifs. Ces fichiers contiennent également la description de chacun des classificateurs entraînés.

Chapitre 5
Résultats

Ce chapitre décrit les résultats que nous avons obtenus pour chacune des expériences décrites au chapitre précédent, soit ScoraliA (section 5.1), puis ScoraliB (section 5.2). Étant donné la grande quantité de résultats, on montrera, pour chaque expérience, un exemple représentatif[1] de classificateur satisfaisant et moins satisfaisant. Pour le premier, nous présenterons les résultats pour PP/V.CONJ (confusion entre le participe passé et le verbe autrement conjugué) et pour le second, les résultats de MODE (mode verbal fautif).

Pour compléter cet exposé, nous présentons également des tableaux synthétiques des résultats obtenus et, en annexe A, nous montrons les résultats obtenus sur toutes les détections étudiées.

5.1 ScoraliA

5.1.1 La détection PP/V.CONJ

La figure 5.1 montre les résultats obtenus par les 2352 classificateurs créés pour la détection PP/V.CONJ. Chacun des points du nuage représente les performances d'un

[1] La représentativité est un terme subjectif, mais le choix d'un classificateur ici sert à illustrer des constatations et conclusions qui s'appliquent à la majorité de nos expériences. Lorsque c'est nécessaire cependant, nous expliquons les particularités de certains classificateurs que notre exemple pourrait ne pas avoir illustrées.

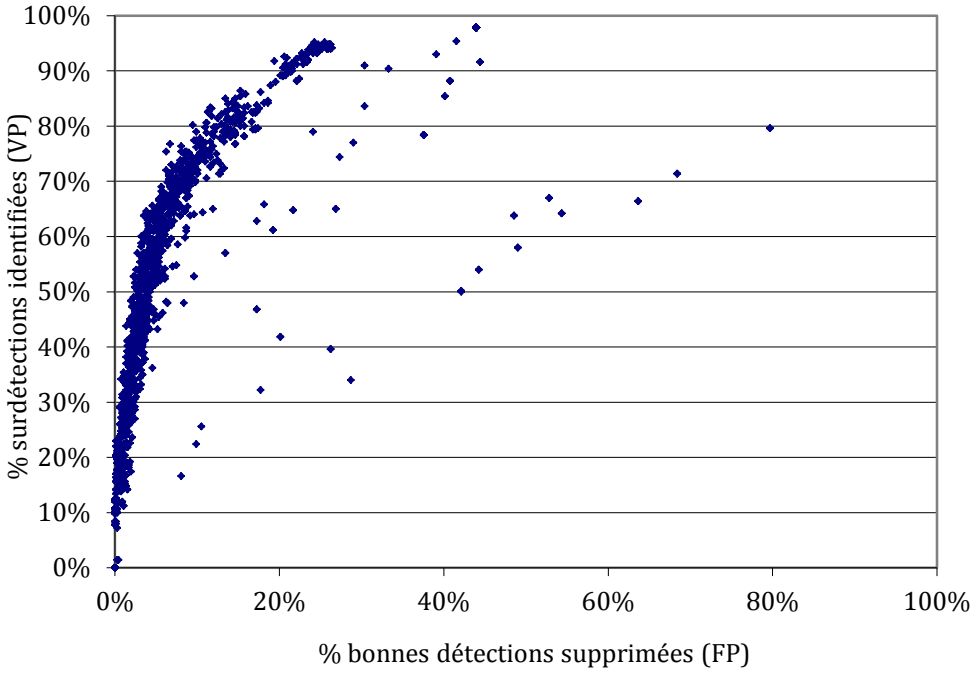


Figure 5.1 — Performances des classificateurs testés pour la détection PP/V.CONJ

classificateur. L'axe des abscisses représente le pourcentage de faux positifs (FP) et l'axe des ordonnées, le pourcentage de surdétections correctement identifiées (vrais positifs). Comme expliqué à la section 4.6.2, ce sont les résultats de validation croisée sur le corpus fourni par Druide.

Les points étant densément groupés dans la région FP < 20 %, on concentre l'attention du lecteur sur ce sous-ensemble des résultats à la figure 5.2, n'écartant ainsi que 286 classificateurs manifestement inutiles, puisque leurs performances sont en deçà de celles souhaitées par Druide.

Il est remarquable de constater que, nonobstant quelques points singuliers, le nuage de points est relativement compact, traçant une bande qui marque les compromis possibles qu'offrent les classificateurs entraînés.

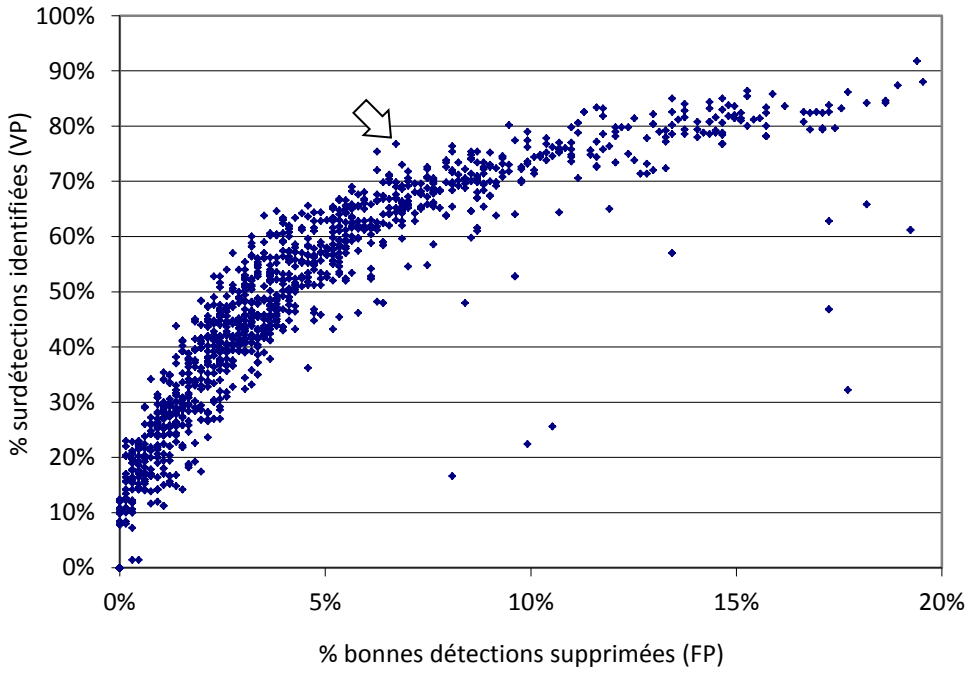


Figure 5.2 — Performances des classificateurs testés pour la détection PP/V.CONJ pour FP < 20 %

En fait, on se retrouve avec l'équivalent d'une *caractéristique de fonctionnement du récepteur* (*Receiver Operating Characteristic*, ROC). Cette courbe apparaît dans l'analyse des classificateurs binaires lorsque l'on met en relation le taux de FP avec celui de VP, ce qui est exactement notre cas. Sur une courbe ROC, un classificateur aléatoire sans pouvoir de discrimination donnerait des points sur la droite allant de (0,0) à (1,1) dans le graphique de la figure 5.2. On voit dans notre cas que les familles de classificateurs qui se dessinent en faisant varier les hyperparamètres des classificateurs permettent de s'éloigner considérablement de cette droite de non-discrimination, ce qui est bon signe.

Le choix du classificateur est fait à partir de la figure 5.2, en tenant compte des contraintes énumérées dans le cahier des charges (section 3.2), qui stipule notamment que VP > 66 % et FP < 10 %. Dans le cas qui nous préoccupe, le classificateur recom-

mandé à Druide est celui dont le point est indiqué par une flèche à la figure 5.2. C'est un arbre de décision C4.5 binaire, élagué, permettant l'identification de 77 % des surdétections, au prix de la perte de 8 % des bonnes corrections. Par ailleurs, l'arbre de décision classifie 86 % des instances correctement (voir tableau 5.1, page 58). D'autres classificateurs aux performances similaires ont été écartés, car ils étaient trop complexes à implémenter, trop lents, ou utilisaient un trop grand nombre de traits. Ce choix a été fait manuellement pour chaque type de faute. La section 6.6.2 propose quelques méthodes d'automatisation de cette sélection.

La figure 5.3 est un extrait des règles au cœur du classificateur sélectionné. Puisque l'arbre est binaire, on peut le décrire en une suite de tests `si` … `sinon` … dans du pseudocode. La figure a été allégée de certains branchements pour ne pas occuper plusieurs pages dans ce document. L'arbre complet comprend 25 feuilles, qui correspondent chacune à une classe « bonne détection » ou « surdétection ».

Un des branchements fait référence au trait que nous symbolisons φ. Ce trait s'applique aux verbes dont la forme au passé simple, à la première personne du singulier, ne peut être confondue avec le participe passé.

Les paires de nombres (x/y) à la suite de chaque étiquette de classe associée à une feuille donnée de l'arbre sont des mesures de pureté. Une paire (x/y) signifie que x exemples correspondent à ce cas de figure et sont correctement classés, et que y exemples correspondent à ce cas de figure mais ne sont pas correctement classés. Ce sont des nombres réels, car la matrice de coût vient multiplier ces nombres, comme on l'a mentionné à la section 4.6.2.

Chaque arbre produit est un cas particulier. On peut par exemple décoder celui de la confusion PP/V.CONJ, à la figure 5.3. On y lit que, s'il n'y a qu'une seule détection au site de la détection, et que le mot n'a pas le trait φ et que le mot est un participe passé, alors la détection est légitime, c'est-à-dire que le correcteur est presque toujours (472,85/7,02) en droit de signaler à l'utilisateur qu'il aurait dû utiliser un autre temps de verbe.

```
si le nombre de détections au site étudié ≤ 1 alors
   si trait du mot du site est φ alors
      surdétection (55,52/8,93)
   sinon
      si catégorie du mot du site est participe passé alors
         bonne détection (472,85/7,02)
      sinon
         …
      finsi
   finsi
sinon
   si catégorie du mot du site est part. passé alors
      bonne détection (2,55/0)
   sinon
      si catégorie du mot précédent est pronom pers. sujet alors
         bonne détection (2,55)
      sinon
         surdétection (82,96/2,55)
      finsi
   finsi
finsi
```

Figure 5.3 — Extrait de l'arbre de décision produit pour la détection PP/V.CONJ

5.1.2 La détection MODE

La figure 5.4 montre les performances des 2352 classificateurs entraînés pour la détection MODE, concernant les modes verbaux fautifs. La courbe dessinée par les points est fort différente de celle de la figure 5.1, pour la détection PP/V.CONJ. La tendance de la figure 5.4 est bien plus proche des compromis offerts par un classificateur qui trancherait au hasard entre les deux classes, ce qui rend futile la tâche de sélection d'un classificateur digne d'une intégration dans Analytix : il n'y en a pas[1]. À la rigueur, pour illustration, on pourrait tenter d'en choisir un aux extrêmes limites des bornes de taux de FP et VP fixés par Druide.

Par exemple, le classificateur pour lequel VP > 66 % et qui minimise FP a un taux de faux positifs de 33 %, ce qui est trois fois la limite prescrite dans le cahier des charges. À l'opposé, le classificateur pour lequel FP < 10 % et qui maximise VP a un taux de vrais positifs de seulement 33 %, ce qui est la moitié seulement des exigences énoncées dans le cahier des charges.

[1] Au chapitre 6, nous proposons des explications de ces difficultés lors de la classification.

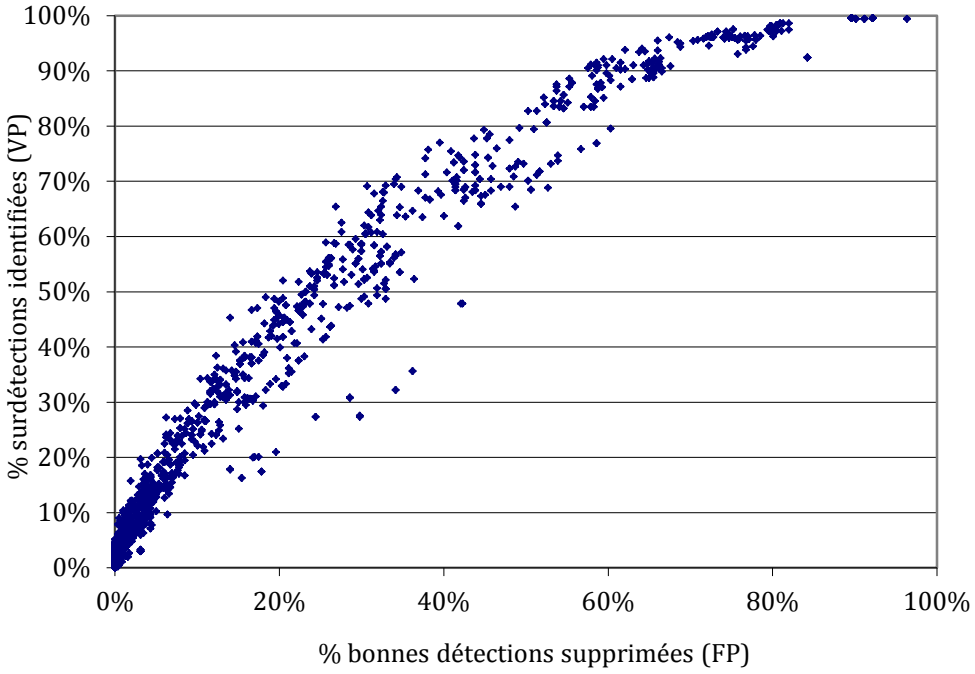


Figure 5.4 — Performances des classificateurs testés pour la détection MODE

5.1.3 La détection QUE/DONT et les traits de désambiguïsation sémantique

La détection QUE/DONT mérite également que l'on s'y attarde, puisqu'elle a donné de mauvais résultats malgré les efforts investis pour ramener le problème de classification à un problème de désambiguïsation sémantique (WSD)[1]. La série de points « ScoraliA », à la figure 5.5, montre que le projet ScoraliA n'a pu trouver qu'un classificateur pour lequel FP = 9 % et VP = 57 %, ce qui est insuffisant.

Face à cet échec, nous avons relancé les expériences pour QUE/DONT, mais cette fois-ci sans les traits de désambiguïsation, afin de vérifier que (1) ces traits ne détériorent pas les performances par quelque interférence avec les traits plus « classiques » et (2) pour mesurer leur apport, lorsqu'on les ajoute aux traits plus classiques. Cette nouvelle expérience prend le nom de « sans WSD » à la figure 5.5.

[1] Voir les traits supplémentaires de WSD à la section 4.4.2.

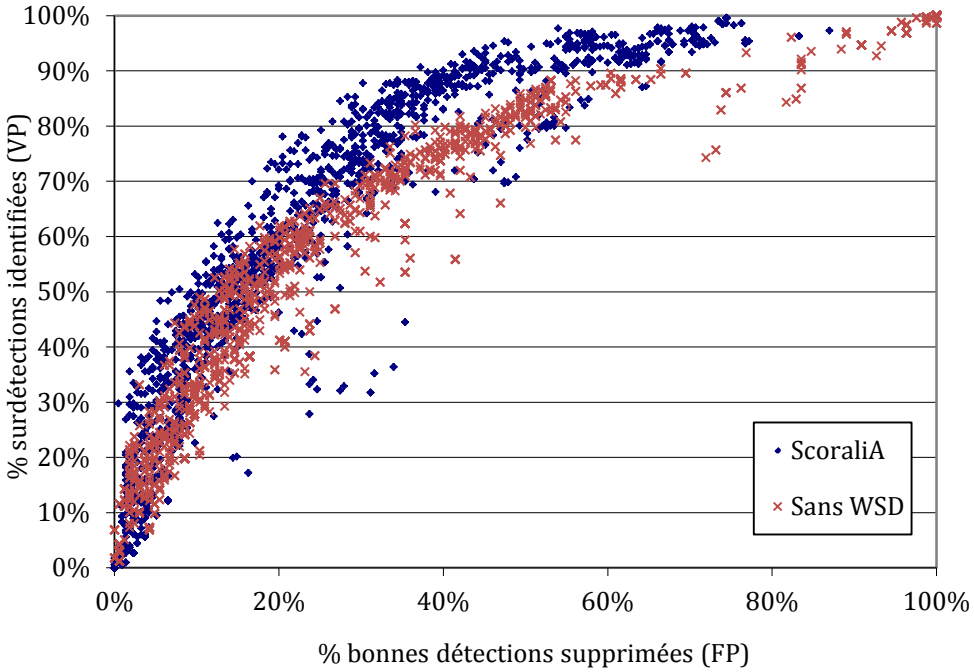


Figure 5.5 — Performances des classificateurs testés pour la détection QUE/DONT avec et sans WSD

On le constate graphiquement, les traits de désambiguïsation sémantique semblent bien améliorer les résultats. Le nuage de points de ScoraliA se situe jusqu'à 10 % au-dessus de celui de l'expérience sans les traits de WSD, pour l'axe des ordonnées (VP). S'il avait fallu choisir un classificateur sans WSD, alors on aurait eu seulement FP = 9 % et VP = 47 %. Notons enfin que l'exploration de méta-classificateurs de type forêt d'arbres décisionnels (*random forest*) a été infructueuse également.

5.1.4 Sélection des classificateurs pour Scorali

Nous avons effectué le travail d'analyse des résultats décrit aux sections précédentes pour les 14 types de fautes fournis par Druide. Les résultats de ce travail sont présentés au tableau 5.1. Les classificateurs qui y figurent sont ceux qui ont été livrés à Druide dans le cadre du projet Scorali, c'est-à-dire le projet dénommé ScoraliA dans cette étude. Les lignes grisées du tableau correspondent aux détections pour lesquelles un classificateur répondant aux exigences du cahier des charges n'a pu être identifié. Ces classificateurs insatisfaisants ont néanmoins été livrés à Druide pour leur gouverne.

Tableau 5.1 — Performances des meilleurs classificateurs (expérience ScoraliA)

Type de faute	% FP	% VP	% classifications correctes
ACCORD	9 %	40 %	68 %
APOS	6 %	73 %	84 %
CONJUG★	8 %	66 %	83 %
ÉLISION★	7 %	87 %	89 %
ER/EZ★	9 %	65 %	78 %
INVAR★	8 %	71 %	83 %
LA/LÀ★	9 %	84 %	88 %
MAJ	7 %	71 %	82 %
MODE	9 %	27 %	51 %
OU/OÙ★	9 %	49 %	74 %
PP/INF★	7 %	68 %	81 %
PP INV★	6 %	80 %	91 %
PP/V.CONJ	8 %	77 %	86 %
QUE/DONT	9 %	57 %	67 %

★ Classificateur intégré dans Analytix par les informaticiens de Druide

On constate ainsi que nous avons réussi à créer des classificateurs satisfaisants pour 9 des 14 détections. Ce sont tous des arbres de décision, et pour cause : à performances égales ou presque, nous sélectionnions un arbre de décision pour uniformiser la tâche d'implémentation lorsque nos observations préliminaires ont montré que ces arbres de décision étaient toujours présents parmi les meilleurs classificateurs pour chaque détection.

Le taux de classifications correctes est systématiquement inférieur à 80 % lorsqu'un classificateur doit être rejeté. Les taux pour les classificateurs sélectionnés vont jusqu'à 91 %. En examinant les arbres (dont la figure 5.3 est un exemple), il n'y a pas vraiment de traits ou de famille de traits qui se détachent des autres quant à leur utilité lors de la classification.

5.1.5 Tests chez Druide et intégration des classificateurs dans Analytix

Le RALI a livré les classificateurs à Druide et celle-ci s'est chargée de leur intégration dans la chaîne de traitement d'Analytix (voir section 3.2). Afin d'assurer un portage

fidèle depuis le laboratoire vers le code industriel, nous avons proposé de valider ce processus d'intégration à l'aide du protocole suivant :

1. Sélection d'un classificateur simple pour faciliter la mise en route du processus de portage (nous proposions la détection ÉLISION);
2. Rédaction du code d'extraction des traits pour cette détection;
3. Rédaction du code du classificateur (arbre de décision);
4. Vérification que les phrases du corpus d'entraînement produisent les mêmes valeurs de traits au laboratoire et chez Druide;
5. Comparaison des sorties (classes) du nouveau classificateur avec celles produites par notre classificateur;
6. Analyse des erreurs, le cas échéant.

Pour que les étapes 4 et 5 plus haut soient possibles, nous avons livré à Druide les valeurs de traits pour chacune des phrases du corpus d'entraînement ainsi que les sorties du classificateur Weka sur celles-ci.

Les tests préliminaires ayant été couronnés de succès, les informaticiens de Druide ont pu concevoir un programme de conversion des arbres sérialisés par Weka en code utilisable au cœur d'Analytix. Des arbres de décision trop profonds leur ont causé quelques problèmes de compilation, mais nous avons remplacé ces classificateurs capricieux par des candidats élagués aux performances presque identiques.

Druide a choisi de convertir et intégrer les 11 meilleurs classificateurs du projet ScoraliA, puis de les tester sur un corpus distinct du corpus d'entraînement fourni, une pratique recommandée dans l'industrie (voir par exemple Helfrich & Music [2000]). Ces tests ont révélé que 3 classificateurs[1] dégradaient les performances du correcteur grammatical au point où il a fallu les exclure. Ces classificateurs n'étaient pas nécessairement les moins bons durant l'entraînement en laboratoire. Les 8 classificateurs survivants[2] font désormais partie d'Analytix et sont donc intégrés notamment dans Antidote HD (Brunelle & Charest, 2010). L'interface graphique du logiciel inclut une case à cocher qui permet d'activer ou de désactiver ces classificateurs. Ceux-ci sont activés par défaut.

[1] Ce sont les classificateurs APOS, MAJ et PP/V.CONJ.
[2] Ils sont indiqués par une étoile (⋆) au tableau 5.1.

Druide a manifestement trouvé féconde l'approche proposée dans ce travail, car, en plus de l'intégrer dans Analytix, quelques chercheurs de l'entreprise montréalaise ont repris notre approche pour la bonifier, notamment en enrichissant les corpus d'entraînement (S. Charest, communication personnelle, 10 juin 2011).

5.2 ScoraliB

Le projet ScoraliB prolonge ScoraliA, que l'on considérera ici comme ligne de base (*baseline*), et offre 3 nouveaux ensembles de classificateurs. Récapitulons les configurations pour clarifier.

- **base** est pour nous ScoraliA, soit la ligne de base.
- **+LM** correspond à ScoraliA enrichi de traits issus des modèles de langue.
- **+SVM** correspond aux mêmes traits que ScoraliA, mais en utilisant des classificateurs de type séparateurs à vaste marge (SVM).
- **+LM+SVM** indique des SVM entraînés sur les traits de +LM.

5.2.1 La détection PP/V.CONJ

La figure 5.6 montre les résultats obtenus par la ligne de base (ScoraliA) ainsi que ceux de +SVM et de +LM+SVM. On tait les résultats de +LM pour alléger la figure; ce dernier nuage de résultats est intercalé entre les classificateurs de la ligne de base et ceux de +SVM, ce qui rendrait la figure illisible. À nouveau, on concentre notre attention sur la région FP < 20 %. On y constate clairement que les tendances observées pour ScoraliA ne se démentent pas pour cette détection : les classificateurs sont groupés en 3 nuages relativement distincts, en forme de courbe ROC, à quelques points singuliers près.

Une observation centrale est que les classificateurs SVM sont en mesure d'améliorer les résultats de ScoraliA. À la figure 5.6, les classificateurs du nuage +SVM flottent à environ 5 % au-dessus du nuage de ScoraliA (base). L'ajout de traits dérivés des modèles de langue (+LM+SVM) augmente encore le nombre de vrais positifs, pour un gain additionnel de quelque 5 %.

Il est quelque peu hasardeux cependant de nous fier à l'observation d'un graphique pour déterminer l'influence qu'a l'ajout de certains traits ou encore le recours à des classificateurs plus sophistiqués, *a fortiori* aveuglé par le grand nombre de résultats

obtenus. Nous avons donc voulu appuyer ces constatations visuelles sur une analyse statistique plus rigoureuse.

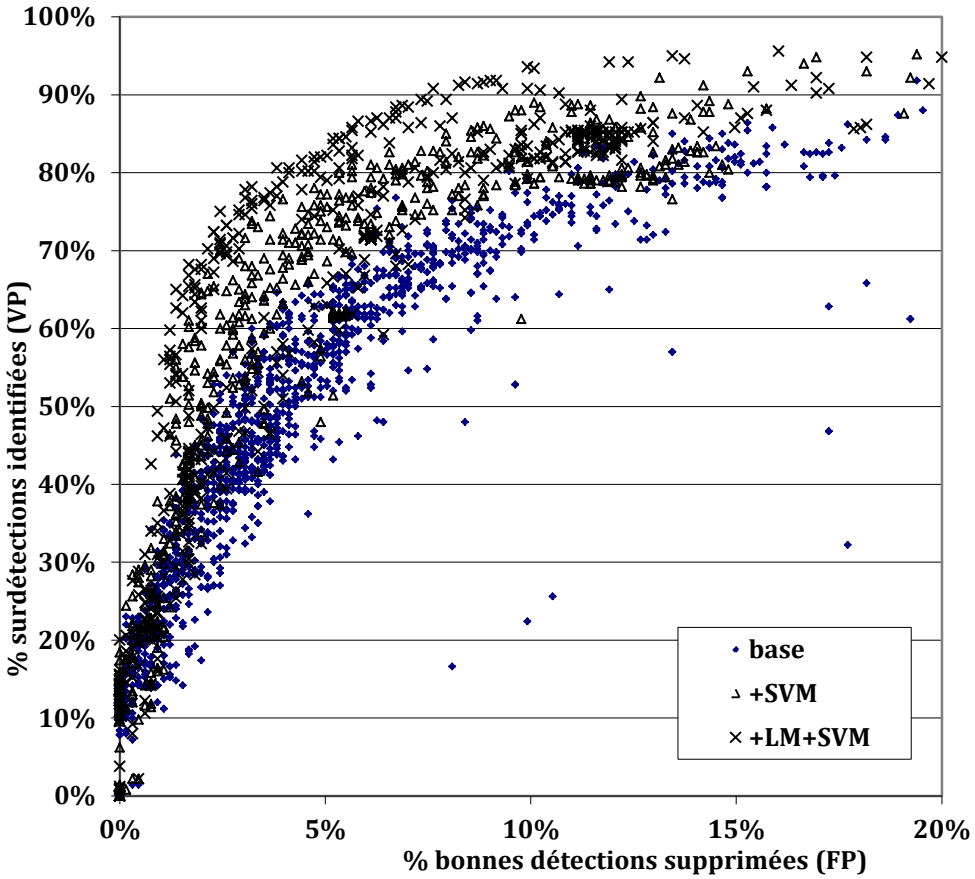


Figure 5.6 — Performances des classificateurs ScoraliA (base), +SVM et +LM+SVM pour la détection PP/V.CONJ

Pour ce faire, nous avons procédé à un test statistique afin de déterminer si les différences observées entre les populations correspondant à chacune des 4 configurations récapitulées plus haut sont statistiquement significatives. La F-mesure a été choisie pour caractériser les performances d'un classificateur donné. Son calcul est expliqué en section 4.1.

Un test *t* de Student n'est pas applicable, car l'hypothèse de normalité des populations étudiées n'est pas confirmée. En effet, comme le montre la figure 5.7, pour la F-mesure de la configuration +LM de ScoraliB, la distribution est très fortement négati-

vement dissymétrique pour cette statistique. C'est un problème répandu dans les populations de classificateurs de notre étude. Même si le test *t* est réputé robuste face aux cas de divergence de la normalité, à cause de la grande déformation de la distribution, nous préférons utiliser les équivalents non paramétriques de ce test, soit les tests de Wilcoxon et de Mann-Whitney.

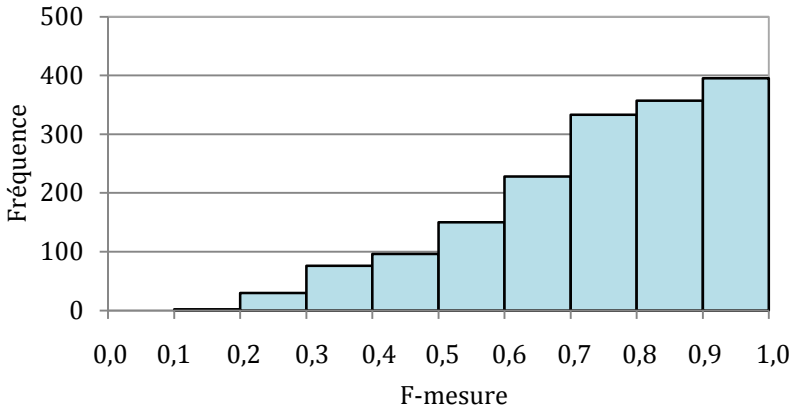


Figure 5.7 — Distribution des F-mesures des classificateurs pour la configuration +LM de la détection PP/V.CONJ

Lorsque l'on a comparé des populations qui ne différaient que par les traits utilisés (par exemple entre +SVM et +LM+SVM), nous avons effectué un test jumelé de Wilcoxon (*Wilcoxon's signed-rank for matched pairs test*). Lorsque les échantillons des populations n'étaient pas jumelés (par exemple entre la ligne de base et +SVM, qui comporte de nouveaux classificateurs), on a plutôt eu recours à un test de Mann-Whitney. Nos tests étaient unilatéraux. Nous avons écrit pour ce faire un programme Java s'appuyant sur l'API `jsc` écrite par Andrew Bertie[1].

On a comparé chacune des configurations l'une à l'autre, pour une détection donnée, ce qui correspond à 6 comparaisons différentes. Les résultats sont consignés au tableau 5.2. On y ordonne les 4 configurations en ordre croissant de F-mesure. Dans ce tableau, la notation $A < B$ indique que la configuration A produit une F-mesure inférieure à la configuration B, et ce, avec un taux de signification statistique $p \leq 0{,}01$. Le

[1] http://www.jsc.nildram.co.uk/

signe égal (=) indique qu'il n'y a pas de différence statistiquement significative entre deux configurations.

Tableau 5.2 — Comparaison des F-mesures produites par les 4 configurations de ScoraliB

Type de faute	Comparaison des F-mesures des classificateurs						
ACCORD	Base	<	+LM	=	+SVM	<	+LM+SVM
APOS	Base	<	+LM	<	+LM+SVM	<	+SVM
CONJUG	Base	<	+SVM	<	+LM	<	+LM+SVM
ÉLISION	Base	<	+LM	<	+SVM	=	+LM+SVM
ER/EZ	Base	<	+LM	<	+SVM	<	+LM+SVM
INVAR	Base	<	+LM	<	+SVM	<	+LM+SVM
LA/LÀ	Base	<	+SVM	<	+LM	<	+LM+SVM
MAJ	Base	<	+LM	<	+SVM	<	+LM+SVM
MODE	Base	<	+LM	<	+SVM	<	+LM+SVM
OU/OÙ	Base	<	+SVM	<	+LM	<	+LM+SVM
PP/INF	Base	<	+LM	<	+SVM	<	+LM+SVM
PP INV	Base	<	+SVM	<	+LM	<	+LM+SVM
PP/V.CONJ	Base	<	+LM	<	+SVM	<	+LM+SVM
QUE/DONT	Base	<	+LM	<	+LM+SVM	<	+SVM

Le tableau 5.2 montre clairement que l'ajout des modèles de langue, le passage aux SVM et ces deux facteurs pris ensemble améliorent de façon statistiquement significative ($p \leq 0{,}01$) les résultats de base, soit ceux de ScoraliA. Nos intuitions se confirment donc à la lecture de cette analyse. Il est bon de noter que ces améliorations sont observées pour toutes les détections, peu importe les résultats qu'elles ont donnés dans ScoraliA.

Il semble également que +SVM l'emporte plus souvent sur +LM, confirmant l'utilité cruciale des SVM dans la classification. La configuration +LM+SVM, quant à elle, est 11 fois sur 14 (79 % des cas) la meilleure des 4 configurations en lice.

Au-delà de cette comparaison entre des populations de classificateurs, ce qui nous intéresse en fin de compte est la sélection du *meilleur* classificateur au sein de chaque population. Pour PP/V.CONJ, on a donc repris l'exercice de sélection (section 5.1.1)

pour chacune des configurations de ScoraliB, et on a comparé nos choix avec ceux de ScoraliA. Ceci est illustré au tableau 5.3.

Tableau 5.3 — Description des meilleurs classificateurs de ScoraliA et ScoraliB pour la détection PP/V.CONJ

Configuration	FP	VP
base	8 %	77 %
+LM	7 %	77 %
+SVM	6 %	81 %
+LM+SVM	6 %	87 %

Même si le processus manuel de sélection est assez subjectif, puisqu'il faut faire son choix parmi la famille de compromis FP/VP qu'offrent les classificateurs, il apparaît au tableau 5.3 que le passage de la configuration de la ligne de base vers un classificateur +LM+SVM pourrait améliorer considérablement le meilleur classificateur livré à Druide, au prix de complexités et lourdeurs additionnelles dans le transfert du code et de sa nouvelle implémentation dans le produit fini. Le taux de faux positifs diminuerait de 2 % pour un gain de 10 % dans les vrais positifs, ce qui est appréciable.

5.2.2 La détection MODE

On poursuit notre description des résultats obtenus avec la détection MODE, qui n'a pas donné de bons résultats pour ScoraliA. La figure 5.8 montre les résultats obtenus par la ligne de base (ScoraliA) ainsi que ceux de +SVM et de +LM+SVM, sur la région FP < 20 %.

Les résultats de la comparaison des 4 configurations pour cette détection (ligne de base, +SVM, +LM, +SVM+LM) ont déjà été présentés au tableau 5.2. Les nuages de points se comportent comme pour la plupart des autres détections, avec la ligne de base surpassée par l'addition des SVM et des modèles de langue. Cette dernière observation constitue une vérification partielle de la logique de nos expériences (*sanity check*) sur cette détection, dont les mauvais résultats auraient pu indiquer un problème dans notre chaîne de traitement. C'est une mince consolation, cependant, car il

nous est toujours impossible de suggérer un classificateur qui satisfasse le cahier des charges pour MODE.

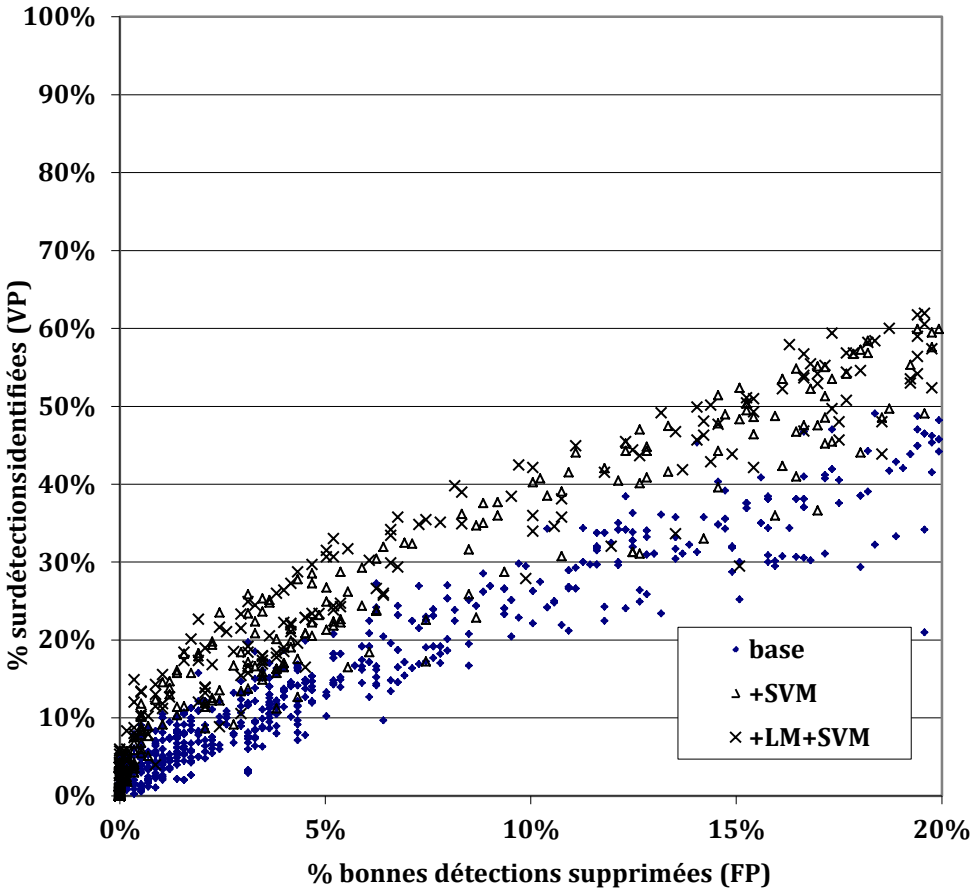


Figure 5.8 — Performances des classificateurs ScoraliA (base), +SVM et +LM+SVM pour la détection MODE

Chapitre 6
Discussion et perspectives

6.1 Succès et difficultés pour Scorali

Somme toute, le bilan du projet ScoraliA est positif : nous avons été en mesure de livrer (à temps) des classificateurs satisfaisants pour 9 des 14 détections sur lesquelles portait le projet, ce qui s'est traduit par le portage de 8 d'entre eux par Druide au sein d'Analytix. Ces classificateurs sont donc à la disposition de la nouvelle mouture d'Antidote, soit Antidote HD. En moyenne, on épargne aux utilisateurs 75 % des surdétections (tableau 5.1), au prix de la perte d'environ 8 % des bonnes détections, un compromis coûts-bénéfices qui a manifestement plu à notre partenaire commercial.

Naturellement, notre attention se porte d'abord sur les détections pour lesquelles nous avons échoué. En fait, il est difficile d'expliquer pourquoi certaines détections se sont bien prêtées à la classification et d'autres pas. Malgré nos efforts, QUE/DONT n'a pas conduit à un bon classificateur, alors que la confusion ÉLISION, pour laquelle il semblait à première vue bien difficile d'induire des règles (voir les exemples disparates de la section 3.3.1), s'est révélée très facile à traiter.

Le premier réflexe dans une tâche de classification qui veut imiter l'humain est de vérifier dans quelle mesure ce dernier est bien capable de classer les instances du corpus d'entraînement correctement, afin de borner supérieurement la qualité des

algorithmes possibles. Nous avons donc vérifié manuellement la classification de 100 instances du corpus QUE/DONT ainsi que 100 instances de MODE et n'y avons trouvé qu'une seule erreur de classification, commise sans doute par inadvertance. C'est donc dire que 2 êtres humains (l'annotatrice originale et nous) sont d'accord sur la classification, et que la tâche est théoriquement possible, du moins pour les détections vérifiées. La distribution des bonnes détections et des surdétections, ainsi que le nombre moyen de détections par phrase (tableau 3.1) n'indiquent rien de particulier pour les détections difficiles à classer : ces statistiques sont comparables à celles des détections plus faciles.

Il n'en demeure pas moins que certains exemples sont particulièrement difficiles à classer sans exploiter une connaissance du monde qui transcende les simples phénomènes linguistiques accessibles à un ordinateur. Les exemples des détections MODE et ACCORD sont éloquents. Pour l'exemple ACCORD$_1$, `accepte le [*la] marche`, il faut vraiment comprendre le contexte pour deviner que le scripteur voulait écrire `marché` et non pas `marche`, ce qui explique la surdétection. Il y a fort à parier que les classificateurs ont de la difficulté avec ces cas de figure, *a fortiori* lorsque ces détections peuvent se faire dans des contextes très différents, et pour des raisons différentes. C'est le cas d'ACCORD, qui peut être une faute d'accord en genre ou en nombre. Il est utile de rappeler que nous avons mis en œuvre des traits qui font la différence entre ces deux variantes de la détection ACCORD. Peut-être serait-il préférable de combiner plusieurs traits en un seul, pour obtenir par exemple un trait de type de détection qui serait « genre seulement », « genre et nombre », « nombre seulement », pour guider le classificateur de façon plus décisive. Il serait également envisageable de produire des corpus d'entraînement différents pour chaque cas.

Naturellement, l'induction de règles de classification est facilitée si le correcteur commet des surdétections sur une certaine construction de façon systématique. C'est le cas par exemple avec la détection LA/LÀ, où l'examen de l'arbre de décision recommandé montre que 20 % des surdétections surviennent lorsque le mot déterminé par `la` est un complément du nom. À nouveau, pour les détections où la classification est difficile à automatiser, il est certain qu'un contexte extrêmement changeant dans le corpus d'entraînement défie l'induction des règles. La lexicalisation des contextes (et, partant, leur explosion combinatoire) est une limite bien connue du domaine du trai-

tement des langues naturelles. Comme le soulignent Golding & Roth (1998), « Deux propriétés caractéristiques de ce genre de problèmes est leur très grande dimensionnalité et le fait que les classes cibles se réfèrent à un sous-ensemble restreint des traits dans cet espace »[1]. Nous avons constaté la même chose lorsque nous avons filtré nos traits : la plupart d'entre eux étaient inutiles, pour une détection donnée (voir section 4.6.1), ce qui est prévisible. Les auteurs de la précédente étude proposent de recourir aux classificateurs de type *winnow*, qui sont des perceptrons rapides à entraîner, qui savent manipuler un grand nombre de traits inutiles. Malheureusement, c'est une approche que nous avons tentée au tout début du projet Scorali, au moment de notre exploration de Weka, sans obtenir de résultats encourageants.

Il y a lieu également de s'intéresser à la richesse, le nombre et la qualité des traits que nous avons extraits des exemples d'apprentissage. *A priori*, toutefois, il est fort difficile de comprendre pourquoi les mêmes traits qui ont permis d'obtenir un pourcentage de VP de 84 % pour la détection LA/LÀ atteignent le score décevant de 49 % pour OU/OÙ, une détection très semblable. Il est certain qu'il est toujours possible d'ajouter des traits : là où l'on explorait le mot qui précède et qui suit, on peut examiner les mots dans une fenêtre de 2 mots avant et après le site de la détection, par exemple. On peut tenter de réduire la dimensionnalité des modèles de langue en construisant des modèles non pas entraînés sur les mots, mais sur les catégories morphosyntaxiques. En fait, les possibilités sont innombrables. Nous avons jugé que cet effort d'ingénierie des traits était complété lorsque nous en sommes arrivés aux quelque 1300 traits que nous proposons ici, mais c'était là un choix manifestement perfectible.

Les exemples d'entraînement peuvent également avoir concouru aux difficultés rencontrées pour quelques classificateurs. On l'a entrevu lorsque l'on a présenté les instances d'entraînement, les styles de rédaction sont extrêmement hétérogènes. Comme on l'a souligné à la section 3.4.1, la qualité du texte varie d'irréprochable à catastrophique, en fonction de la source de ce texte.

[1] Notre traduction de « Two characteristic properties of such problems are that their feature space is of very high dimensionality, and their target concepts refer to only a small subset of the features in the space. » dans Golding & Roth (1998).

Une façon de pallier ces difficultés serait de prévoir une stratégie de classification adaptée au style de rédaction. Les modèles de langue s'avèrent des outils idéaux pour ce faire. Après tout, Analytix est déjà en mesure de reconnaître un passage rédigé dans une langue autre que le français, afin d'y éviter les détections. On entrevoit ici une stratégie possible : lorsque le style d'un texte est reconnu comme susceptible d'engendrer des surdétections (comme les messages SMS), il vaudrait peut-être mieux taire toute détection. Une mesure intermédiaire serait de prévoir des jeux de classificateurs différents, activés selon l'habileté de celui qui rédige, ou bien encore d'ajouter un trait indiquant la qualité du texte à corriger ou l'habileté générale du scripteur. D'ailleurs, Analytix utilise déjà des réglages de modulation de ce genre.

On a dit que les modèles de langue se prêtent à cette identification, mais on pourrait également, pour détecter un texte où la détection est périlleuse, s'appuyer sur diverses caractéristiques que nous jugeons très discriminantes, par exemple :

- l'absence systématique de majuscules;
- l'absence systématique de diacritiques (nommément, les accents et cédilles) sur des mots qui devraient en contenir;
- l'utilisation de certains mots typiques du style télégraphique (`tjs` pour `toujours`, `pb` pour `problème`, etc.);
- l'utilisation de certains mots typiques du style de rédaction de certains internautes ou des utilisateurs de téléphones portables (abréviations de type `lol` ou `dem1` pour « demain », frimousses [*smileys*], etc.).

Dans une voie connexe, si certains classificateurs ont connu des ratés pour 5 des 14 détections étudiées ici, c'est peut-être en partie dû aux interactions entre différentes fautes au sein d'une même phrase. Dans un texte de piètre qualité, celles-ci confondent l'analyseur et peuvent donner, en fin de compte, des traits trompeurs. Dans les modèles de langue, par exemple, si les mots du contexte conditionnant contiennent déjà des fautes ou des corrections issues elles-mêmes de surdétections, alors les probabilités obtenues sont erronées.

6.2 La piste des apprenants de l'anglais

Pour bonifier les stratégies vues ici, nous pourrions nous inspirer des travaux effectués sur la détection de fautes dans les corpus d'apprenants de l'anglais. Ce problème

est apparenté au nôtre, puisqu'il s'agit de déterminer si un site donné contient une faute ou pas. Dans notre cas, cependant, c'est afin de confirmer ou d'infirmer le diagnostic du correcteur en amont.

Chodorow *et al.* (2007), dans une étude tentant de détecter les fautes sur les prépositions anglaises, utilisent notamment un classificateur à entropie maximale exploitant des traits tels que les bigrammes et trigrammes de part et d'autre du site de la détection, le nom et le verbe qui précèdent, et le mot de tête qui suit. Les auteurs observent une précision de 0,8 et un rappel de 0,3 sur des essais écrits par des apprenants chinois et russes de l'anglais. Même si leur approche est relativement lourde, les traits qu'ils proposent sont intéressants. Par ailleurs, leur modèle a de la difficulté avec les phrases qui contiennent plusieurs fautes, ce qui s'avérerait une fâcheuse vulnérabilité dans notre cas. Quoi qu'il en soit, il est intéressant de noter que, comme pour nous, certaines prépositions se montrent bien plus difficiles à traiter que d'autres. Les auteurs concluent leur article en proposant la combinaison de classificateurs, une suggestion que nous jugeons judicieuse.

Pour leur part, Lee & Seneff (2008) ont eu l'idée d'inventorier les perturbations engendrées par les fautes verbales sur les arbres d'analyse syntaxique. Pour ce faire, ils ont commencé par étudier les patrons d'utilisation licite des verbes anglais, puis ont introduit des erreurs contrevenant à ces patrons dans leur corpus d'entraînement. Enfin, ils ont procédé à l'analyse syntaxique du texte corrompu et ont recensé les anomalies dans les arbres d'analyse causées par l'introduction des erreurs. Armés de ces patrons d'arbres, ils ont pu ainsi détecter les erreurs sur les verbes commises par des étudiants chinois et japonais, avec une précision de 99 % et un rappel de 43 %. Il faut noter que les auteurs ont dû recourir à un très volumineux modèle de langue pour augmenter la précision. Néanmoins, Scorali pourrait sans doute profiter d'un pareil catalogue de patrons d'arbres suspects afin de déterminer si une détection est valide ou pas. La constitution de pareille ressource dans le cadre de cette étude reste une question ouverte.

6.3 Désambiguïsation sémantique et détection QUE/DONT

La section 4.4.2 décrit des traits supplémentaires de désambiguïsation sémantique (WSD, *word sense disambiguation*) pour la détection QUE/DONT. Les résultats (section 5.1.3) sont décevants. Toutefois, ceux-ci ne condamnent pas notre approche de

WSD. En effet, comme la figure 5.5 l'illustre, les résultats auraient été encore pires si nous n'avions pas fait usage de ces traits moins orthodoxes : le meilleur classificateur aurait ainsi eu un taux de vrais positifs qui aurait chuté de 57 % à 47 %. Certes, aucun des deux n'est acceptable selon le cahier des charges, mais un gain de 10 % est appréciable. Les raisons des faibles performances pour cette détection recoupent sans doute celles que nous évoquions à la section 6.1.

Dans tous les cas, la désambiguïsation sémantique apporte beaucoup à la tâche de classification pour QUE/DONT. Notre hypothèse initiale se vérifie, car on peut bel et bien considérer `que` et `dont` comme des étiquettes possibles d'un mot au « sens inconnu » au site de la détection. En fait, la technique de WSD permet essentiellement d'aller fouiller plus loin dans le contexte de la détection pour trouver un indice de la validité de cette détection. Les traits autres que ceux de la WSD explorent déjà le contexte, puisque des traits sont consacrés aux mots qui précèdent et qui suivent le site de la détection de même qu'au régissant et subordonné de la détection, mais cette fenêtre reste étroite. Cette limite est particulièrement débilitante lorsque la collocation la plus discriminante contient un terme qui est séparé du site de la détection par de nombreux mots-écrans. C'est le cas de l'exemple QUE/DONT$_3$ (page 30), où la collocation importante est celle entre `dont` et `envie`, qui confirme la validité de la détection faite par le correcteur, mais qui s'étale sur 4 mots. Il est presque certain que les modèles de langue utilisés dans ScoraliB agissent par le même principe. Nos trigrammes se limitent cependant à un contexte de deux mots.

À la réflexion, il n'est pas très étonnant que cette détection profite tant de la désambiguïsation sémantique. En effet, `que` et `dont` sont des mots très similaires, lorsque décrits par les traits sans WSD extraits dans ce travail. Tous deux sont des pronoms relatifs, et sont donc susceptibles d'être utilisés dans le même contexte, soit pour l'introduction d'une proposition subordonnée relative. On les verra par exemple précédés de `ce` ou encore suivi d'un pronom personnel débutant la proposition relative. Ce qui les différencie peut être mieux capturé par les collocations à longue distance de modèles statistiques qui s'intéressent au *sens* du mot ambigu. Le classificateur a dès lors accès à ces subtiles différences sémantiques, autrement insaisissables.

Une étude complète sur les traits de désambiguïsation pourrait à elle seule faire l'objet d'un mémoire supplémentaire, tant les questions soulevées sont importantes

et complexes. On pourrait certainement commencer par tenter l'application de la stratégie expliquée ici à d'autres détections candidates, par exemple LA/LÀ et OU/OÙ, qui offrent une confusion entre deux étiquettes. Il serait également utile d'étudier d'autres méthodes de désambiguïsation sémantique lorsqu'il y a une possible confusion entre deux mots, ou de voir dans quelle mesure on pourrait étendre cette approche à d'autres détections. Les pistes de recherche abondent, et leur dévouer toute l'attention qu'elles méritent dépasserait le cadre de ce travail. Une chose est sûre, à ce point, c'est que la désambiguïsation sémantique aide nos classificateurs, au moins pour QUE/DONT.

6.4 ScoraliB : Modèles de langue et SVM

Comme l'ont montré les résultats de ScoraliB, les traits dérivés des modèles de langue et les SVM se sont révélés deux stratégies fructueuses pour améliorer les résultats livrés à Druide. Lorsque combinées, ces dernières permettent une augmentation du taux de vrais positifs qui peut atteindre 15 % sans augmenter le taux de faux positifs.

Les modèles de langue permettent à eux seuls des améliorations moyennes de 5 à 10 % dans le taux de vrais positifs, sans pour autant augmenter le taux de faux positifs. On le rappelle, on n'a pu livrer ces modèles à Druide, car ceux-ci étaient relativement volumineux (voir section 4.4.3). Toutefois, il faut noter que des techniques relativement bien connues de simplification de ces modèles sont envisageables pour diminuer leur taille. Ainsi, on pourrait simplifier le vocabulaire sur lequel travaillent ces modèles de langue ou encore discrétiser les probabilités qu'ils consignent. Par exemple, Whittaker & Raj (2001) montrent que la quantification (*quantization*) et l'élagage de modèles statistiques de langue pour l'anglais autorisent une diminution de 60 % de la taille en mémoire du modèle, sans aucune perte notable dans la distribution. Si l'on consentait ici à quelques pertes dans le modèle de langue, il y a fort à parier que nous égalerions cette marque, ou la dépasserions. Il serait également possible de supprimer l'un des deux modèles, s'il s'avère qu'il contribue peu aux traits les plus discriminants, divisant ainsi les tailles et temps de chargement par deux. Il n'en demeure pas moins que ces modèles resteraient volumineux et ralentiraient sensiblement le correcteur.

Les SVM ont également montré leur utilité, améliorant le taux de vrais positifs jusqu'à 10 % pour le meilleur classificateur sélectionné. Ici aussi, nous avons été réticents à

les intégrer d'emblée dans le livrable final du projet, car ils posent des contraintes techniques importantes. Nous expliquons ici les principales difficultés anticipées et esquissons des solutions.

L'exportation des SVM est difficile à partir de Weka, car ce cadre applicatif ne supporte pas « naturellement » les SVM. Par conséquent, alors que les arbres de décision produisent des sorties détaillées rendant relativement simple le travail d'implantation de ceux-ci dans le correcteur, les SVM n'offrent que peu d'indications sur leur fonctionnement. Reproduire un SVM créé par Weka dans le moteur de correction s'annonce donc particulièrement délicat, voire impossible. L'option la plus populaire est l'exportation du modèle sous forme sérialisée, lisible ensuite par Weka, ce qui est incompatible avec notre idée du portage du classificateur vers Analytix, qui doit s'affranchir complètement de Weka (et de Java). Dans notre cas, on souhaite en effet réécrire le classificateur dans le code d'Analytix. Il n'est pas dit non plus que d'autres librairies logicielles feraient mieux à ce chapitre, et il est difficilement envisageable de s'appuyer sur des librairies externes qu'il faudrait distribuer avec les avatars commerciaux d'Analytix, tout en respectant leur licence d'exploitation.

Il y a également un risque que le calcul de classification soit plus long avec un SVM, car il requiert le calcul de tous les traits avant de débuter la classification, alors qu'un arbre de décision n'a besoin des valeurs des traits qu'à la demande, c'est-à-dire lorsqu'il est confronté à un branchement.

Par ailleurs, les modèles des SVM sont plus difficiles à interpréter pour un humain que les arbres de décision, dont on a vu les qualités à la section 4.5.1. Les SVM ne contiennent en effet aucune règle explicite, seulement des paramètres continus, ce qui les rend plus opaques et difficiles à modifier, sans avoir à recourir à un nouvel entraînement. Par ailleurs, ils se prêtent moins bien à une rétroaction sur le fonctionnement du correcteur, comme expliqué plus loin, à la section 6.6.

Plusieurs solutions sont possibles pour régler ou réduire les difficultés expliquées ci-dessus. Il faudrait peut-être troquer Weka contre un autre logiciel spécialisé dans les SVM. Si Weka nous a bien servis dans un premier temps, il rend difficiles l'exportation et l'entraînement des SVM, comme on l'a vu. Si l'on désire s'épargner la rédaction d'un classificateur SVM maison, plusieurs librairies spécialisées dans les SVM sont envisa-

geables. Il faudrait les explorer en tenant compte de leur facilité d'utilisation, d'entraînement, d'exportation, et des réserves que nous avons formulées plus haut sur leurs licences d'exploitation.

Les principaux logiciels offrant des classificateurs SVM sont les suivants :

- Weka (puisque l'on a accès au code source de Weka et de son moteur SVM, on pourrait tenter de s'inspirer du code pour réécrire un classificateur);
- libsvm[1] (la version C++ qui a engendré le moteur Java utilisé par Weka);
- SVM*light*[2] (un moteur C++ réputé rapide).

De nombreuses autres librairies sont disponibles[3]. Le paquetage libsvm a le mérite d'être écrit en C++ et la licence semble assez permissive.

6.5 Comportement des classificateurs en « milieu sauvage »

Un des succès de ce projet est le portage réussi d'une technologie mise au point en laboratoire vers un produit commercial[4]. Il reste qu'il y aura sûrement des comportements inattendus des classificateurs lorsqu'ils seront confrontés aux détections faites en « milieu sauvage », à l'extérieur du cadre chaud et douillet du laboratoire.

On l'a écrit plus haut, une des difficultés inhérentes à l'entraînement des classificateurs est la différence entre les corpus d'entraînement et les textes sur lesquels le correcteur sera appliqué. À notre avis, ces différences entre les corpus d'entraînement et de test seront une des principales causes de la détérioration des performances des classificateurs livrés. Une façon de lutter contre ce problème est, répétons-le, de tenter de détecter le style de rédaction des phrases sur lesquelles le correcteur est appliqué.

Par ailleurs, le milieu « réel » dans lequel évoluent ces classificateurs inclut des changements dans le temps. Primo, comme le souligne notre partenaire commercial, les données d'entraînement de Scorali ont été générées à un moment particulier de l'histoire d'Analytix. Son évolution au cours des versions ultérieures altérera son fonc-

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvm/
[2] http://svmlight.joachims.org
[3] Voir une liste non exhaustive ici : http://www.support-vector-machines.org/SVM_soft.html
[4] Ce succès n'est pas seulement celui de notre approche, mais il est aussi dû au travail méticuleux et inspiré de nos confrères de Druide informatique.

tionnement et son comportement de détection (et donc de surdétections). Secundo, avec l'avènement de formes neuves de communication électroniques, comme le microblogage (*microblogging*) et l'étendue de celles comme les messages SMS[1], il est fort probable que de nouvelles fautes (pour ne pas dire de nouvelles langues) verront le jour. Ces changements pourront désarmer certains des classificateurs livrés dans ce projet. Idéalement, il faudrait alors répondre en procédant à une nouvelle collecte de données d'entraînement, suivie d'un réentraînement de classificateurs.

6.6 Fouille d'erreurs sur les sorties du correcteur grammatical

6.6.1 Rétroaction double de Scorali sur le correcteur grammatical

Le but de cette étude est l'amélioration d'un correcteur grammatical. On propose d'atténuer les surdétections en utilisant l'apprentissage machine, mais cet exercice a un effet secondaire bien désirable : il permet une analyse des erreurs du moteur d'analyse du correcteur. Cette analyse des erreurs se fait sur deux fronts.

Le premier, qui ne concerne pas cette étude directement, est une rétroaction de l'annotatrice après son travail sur les exemples d'entraînement. En effet, pour chacun des types de fautes étudiés, elle a produit, au terme de son annotation, un document qui expliquait les principales surdétections qu'elle constatait (voir section 3.3). Par exemple, pour la détection MODE, elle constate que la surdétection de type **`Traduit`** *`[*Traduis]`* `de l'espagnol` (donc le préfixe `Traduit de` suivi d'un nom de langue) est souvent présente. Une rectification peut alors être apportée au moteur de correction pour tenir compte de ces précieuses observations.

Le second front est celui des règles induites par les classificateurs durant l'entraînement, et c'est ici que nous pouvons intervenir. De fait, il nous semble qu'il y a une compatibilité fort heureuse entre, d'une part, le fonctionnement du correcteur grammatical et, d'autre part, les règles au cœur des classificateurs livrés : ces dernières pourraient s'ajouter à celles qui ont été entrées manuellement[2] dans Analytix ou encore contribuer à rectifier certaines règles existantes.

---

[1] Pour une étude d'envergure sur le sujet, voir le projet sms4science (http://www.sms4science.org).

[2] Naturellement, nous ne prétendons pas connaître les détails d'implémentation d'Analytix, mais l'utilisation de règles confectionnées manuellement est conforme à l'état de l'art (voir chapitre 3).

Si l'on examine l'arbre de décision produit pour la détection LA/LÀ, par exemple, on constate que le premier branchement se fait en considérant la relation de dépendance syntaxique avec le parent du mot au site de la détection. Si cette relation en est une de complément du nom, alors c'est *toujours* une surdétection dans le corpus d'entraînement. Ce cas de figure survient pour 118 des 621 cas de surdétections des exemples, soit dans 19 % des cas, ce qui est remarquable. Un exemple parmi d'autres est `Il est considéré comme un expert de` **`la`** *`[*là]`* `« grimpe » des falaises.` Au vu de ces étrangetés, il est envisageable de retracer la source de cette erreur dans le moteur de correction. Peut-être est-ce une ressource embarquée qui est la coupable, ou peut-être a-t-on affaire à un caprice de l'analyseur syntaxique, seule Druide peut le déterminer avec certitude.

Il est bon de noter que ce ne sont pas tous les arbres de décision qui se prêtent à ce genre d'analyse : certaines règles induites sont difficilement interprétables par un être humain. En outre, certains arbres possèdent un grand nombre de feuilles qui se partagent chacune une petite fraction des instances d'entraînement. C'est donc dire que, pour ces derniers, il est difficile d'isoler un cas de figure particulièrement suspect, c'est-à-dire des conditions qui engendrent un grand nombre de surdétections. Pour les classificateurs qui n'ont pas donné de bons résultats, on pourrait être tenté de les rejeter d'emblée, mais une analyse est nécessaire. En effet, ce n'est pas parce que le résultat final est inacceptable qu'il n'existe pas quelques règles susceptibles d'engendrer d'utiles rétroactions. Après examen de quelques candidats intéressants, cependant, aucune information d'intérêt ne nous est apparue. Notons que cette tâche de fouille d'erreurs aurait été beaucoup plus complexe si nous avions opté pour des SVM ici. Il faut cependant nuancer et mentionner qu'il existe bien des méthodes d'interprétation visuelle des SVM (voir par exemple Cook *et al.* [2004] pour des outils en basse dimension), et que les vecteurs supports (*support vectors*) encodés par chaque modèle correspondent chacun à un exemple d'entraînement délimitant l'hyperplan séparateur entre les deux classes.

6.6.2 Une fouille d'erreurs automatisée

Cette fouille d'erreurs n'est pas sans rappeler les travaux de van Noord (2004, 2007) et de Sagot & Villemonte de la Clergerie (2009), expliqués à la section 2.4. Les détails diffèrent, mais le principe général est le même : raffiner un outil en lui soumettant une

grande quantité d'entrées, chercher un critère d'échec dans les sorties de cet outil, puis tenter de corréler ces échecs avec des « suspects » dans les entrées. Ces méthodes permettent des rétroactions fécondes. La méthode présentée ici est plus lourde que celle des études précitées. En effet, nous requérons une (coûteuse) intervention humaine à trois reprises, soit (1) lors de l'annotation des sorties, (2) au moment de la sélection d'un classificateur, et (3) lors de l'examen des règles produites. Il serait intéressant de déterminer comment s'affranchir des ces interventions, si nous voulions créer un outil automatique de fouille d'erreurs. Nous expliquons ci-après quelques propositions pour alléger chacune de ces étapes.

Annotation Pour l'annotation des détections, il serait peut-être faisable de soumettre chaque phrase à plusieurs (autres) moteurs de correction grammaticale, puis de comparer automatiquement les résultats[1]. Un vote entre correcteurs pour une détection serait alors une méthodologie intéressante pour repérer les fausses alertes. Il est possible également de soumettre au correcteur un texte réputé irréprochable, comme les débats du Hansard canadien ou certains corpus de qualité disponibles sous forme électronique. Toute détection y serait alors illégitime. Le danger évident de cette dernière approche est un apprentissage biaisé sur des textes de rédacteurs très habiles, qui constituent (hélas) une minorité. Les exemples éluderaient notamment toute la question des interactions entre plusieurs fautes au sein de la même phrase, ce qui est fort regrettable.

Une méthode possible pour mitiger cette dernière limite serait de combiner des corpus de qualités différentes. Par exemple, en agençant, au sein d'une même phrase, des tronçons issus d'un corpus *A* sans faute avec des tronçons sélectionnés dans un corpus *B* de « mauvaise qualité », on obtient un corpus hybride fort utile. En effet, le correcteur y est mis à l'épreuve, car il doit éviter toute détection sur les tronçons venant de *A*, tout en étant exposé à des erreurs sur les tronçons du corpus *B*.

Un protocole plus complexe consisterait à transformer les phrases d'un corpus sans faute en phrases imitant le niveau de langue des utilisateurs moyens du correcteur, avec fautes. Les Suédois Bigert *et al.* (2007) décrivent par exemple le logiciel Missplel,

[1] Si l'automatisation est vraiment impossible, on peut toujours diminuer les coûts d'annotation en utilisant un outil comme Mechanical Turk (https://www.mturk.com/) pour répartir la fastidieuse tâche d'annotation entre plusieurs vérificateurs.

en mesure de générer des fautes artificielles de syntaxe, d'homophonie, d'orthographe et même de frappe. En utilisant une méthode de génération de phrases artificielles fautives, on pourrait donc altérer une phrase du Hansard en une phrase contenant une faute à un endroit précis. Toute autre détection faite par le correcteur serait une surdétection[1].

Toutefois, cette approche manque de finesse. On peut, à la rigueur, transformer un `Je peux` en un `Je peu` en utilisant les techniques expliquées par Bigert *et al.*, mais nous voyons mal comment celles-ci pourraient générer l'exemple MODE$_2$ (voir page 28) à partir du Hansard. Il faudrait plutôt élaborer une typologie des fautes fréquentes pour chaque catégorie d'utilisateurs, et tenter de recréer celles-ci dans le corpus artificiel, un travail de longue haleine.

Les travaux récents de Wisniewski *et al.* (2010) offrent une solution de rechange potentielle à la création manuelle d'un corpus fautif. Ils ont en effet créé un corpus contenant notamment 74 100 corrections grammaticales à partir de l'historique des révisions des articles français de Wikipédia[2]. Il serait alors réalisable de renverser ces modifications colligées *in vivo* afin d'insérer des fautes là où le texte original est autrement irréprochable.

Sélection d'un classificateur de fouille d'erreurs Ensuite, on pourrait sélectionner automatiquement un classificateur en se fiant à une statistique particulière, comme le pourcentage d'instances correctement classées, l'aire sous la courbe ROC, ou encore la F-mesure. On pourrait même utiliser des heuristiques plus simples, en sélectionnant le classificateur qui maximise VP tout en satisfaisant 7 % ≤ FP ≤ 9 %, ce qui est le cas pour nos sélections. On peut pousser cette exploration plus loin en parcourant les compromis FP/VP possibles à l'aide de réglages d'un classificateur donné.

Qui plus est, on pourrait envisager de court-circuiter complètement cette deuxième étape de sélection d'un classificateur. En effet, si la conjecture faite plus haut se vérifie, un classificateur ne doit pas nécessairement s'illustrer par ses performances pour constituer un outil de fouille d'erreurs utile. On passerait directement à la dernière

[1] Notons que, dans ces conditions, si le correcteur ne détecte pas la faute introduite artificiellement, on est en présence d'une sous-détection, un problème certes connexe, mais qui dépasse le cadre de ce travail.

[2] Les auteurs de cette étude utilisent l'exemple `dans le but de *`**`sensibilisé`** *`[sensibiliser]`* `les changements`.

étape, en considérant un ensemble relativement grand de classificateurs. Naturellement, il faudrait d'abord vérifier notre hypothèse de travail, quelque peu hardie.

Examen des classificateurs Enfin, l'inspection des règles produites par les classificateurs pour y déceler des cas de figures générateurs de surdétections pourrait être menée à bien en utilisant les taux de classification correcte émis par Weka pour les feuilles des arbres de décision[1]. On pourrait alors décider d'extraire, depuis les classificateurs sélectionnés, les règles qui classifient une grande proportion des surdétections du corpus d'entraînement.

À la lumière des propositions que nous venons de faire, il n'est donc pas interdit de concevoir une chaîne de fouille d'erreurs complètement automatisée, qui annoterait des exemples d'entraînement, sélectionnerait des classificateurs entraînés sur ces exemples, puis en explorerait les règles afin d'identifier des cas de figure où le correcteur se comporte de façon suspecte. L'humain utilisera ces rétroactions où il le jugera nécessaire pour bonifier la chaîne de traitement en amont.

[1] Voir la section 5.1.1, qui explique la signification des couples (x/y) annotant chaque feuille des arbres de décision.

Chapitre 7
Conclusion

Le but initial du projet Scorali a été d'offrir à Druide un moyen d'améliorer leur moteur de correction grammaticale en utilisant l'apprentissage machine pour inhiber une partie de ses fausses alertes, que nous avons appelées surdétections dans cette étude. Pour ce faire, des exemples annotés de bonnes et mauvaises détections pour 14 types de fautes ont permis l'entraînement de classificateurs binaires dont 8 ont été intégrés avec succès en aval d'Analytix, le moteur d'analyse et de correction grammaticale de Druide. Le projet a nécessité l'exploration de milliers de classificateurs, ainsi qu'un délicat équilibre entre performances prescrites et contraintes techniques, toutes détaillées dans le cahier des charges de Scorali.

La confusion entre les mots `que` et `dont` a permis de démontrer qu'il était possible de repenser la tâche de classification des détections et de la voir comme une désambiguïsation sémantique, avec gains significatifs à l'appui. Ce résultat laisse présager que cette stratégie pourrait s'étendre à d'autres détections, ou pourrait être substituée par des algorithmes de désambiguïsation plus sophistiqués.

Pour le moment toutefois, certaines détections semblent ne pas se prêter à l'approche proposée, peut-être parce que la détection se fait dans des contextes trop variés ou apparaît pour des raisons trop diverses, défiant l'induction de règles. Il se peut égale-

ment que nous soyons en butte à l'importante variabilité des niveaux de langue des exemples d'entraînement, mais, après tout, c'est là une contrainte nécessaire qui reflète la diversité des textes réels. Nous proposons comme avenue de recherche future de mitiger cette difficulté en détectant le registre de langue utilisé dans une phrase ou un texte à corriger à l'aide de règles ou de modèles de langue.

Le fait que nous avons traité chaque faute de manière indépendante est aussi un point perfectible de notre étude : on observe effectivement un phénomène de cascade, où par exemple un diacritique manquant sur un mot confond le correcteur suffisamment pour le faire commettre une surdétection. Là aussi, une évaluation de l'habileté du scripteur pourrait ne pas être inutile.

Néanmoins, notre travail tel que livré à Druide offre une approche complémentaire aux travaux menés sur la détection de problèmes dans les grammaires à large couverture (van Noord, 2004; van Noord, 2007; Sagot & Villemonte de la Clergerie, 2009), et présente l'avantage de ne pas nécessiter la modification de la grammaire, une entreprise délicate qui n'est pas toujours souhaitable dans une application grand public. Ceci est possible puisque nos classificateurs se situent en aval du moteur de correction, et sont donc découplés de la chaîne de traitement qui les précède.

Notons en outre que Scorali est un exemple particulièrement réussi du déploiement d'une approche d'ingénierie statistique au service d'une application langagière grand public robuste. Nous avons eu en effet le privilège de voir les résultats de notre travail rapidement et expertement intégrés au logiciel Antidote HD, un transfert que nous espérons heureux pour les utilisateurs du correcteur. Leur rétroaction pourrait en fin de compte nous aider dans la conception de filtres de surdétections, cela nous semble évident. En outre, même si la majorité de nos classificateurs ont réussi les tests de régression et d'intégration chez Druide, il sera instructif d'observer comment ils se comporteront au cœur d'un produit en évolution. L'apparition éventuelle d'un problème de qualité nécessiterait sans doute une nouvelle collecte de données et un réentraînement de classificateurs.

Dans un second volet qui visait l'étude de stratégies d'atténuation des surdétections difficilement réconciliables avec une implémentation industrielle, et que nous jugions essentielle à l'exploration d'un problème aussi vaste qu'intéressant, nous avons ana-

lysé les performances de classificateurs de type SVM et nous avons ajouté des traits calculés à partir de modèles de langue. Les gains obtenus sont significatifs. Les SVM seuls montrent un gain en pourcentage de surdétections identifiées, par rapport aux classificateurs livrés, au prix d'une complexité de calcul accrue et d'une expressivité pour le moins réduite. L'ajout de traits en provenance des modèles de langue augmente encore le nombre de vrais positifs. Ceci est observé pour les 14 types de détections.

L'intérêt évident de ces stratégies plus sophistiquées pose la question de leur compatibilité éventuelle avec Analytix. Nous esquissons quelques pistes de solutions. Elles visent d'une part l'allégement des modèles de langue pour limiter l'encombrement (*footprint*) mémoire et disque de ces données statistiques, une exigence industrielle incontournable. D'autre part, elles ciblent l'utilisation des SVM, soit en disséquant les librairies logicielles utilisées ici afin de les rendre portables vers l'industrie, soit en sélectionnant d'autres outils s'y prêtant mieux. Ici aussi, explorer ces voies requerrait une étude en bonne et due forme.

Pour certains types de détections, nos classificateurs s'avèrent de potentiels outils de fouille d'erreurs, dans la mesure où les règles induites par les arbres de décision livrés à Druide décrivent en fin de compte des cas de figure où ceux-ci rencontrent des surdétections, donc des anomalies dans le fonctionnement du correcteur. Ces outils de diagnostic pourraient conduire à des rectifications du moteur de correction ou à des bonifications des ressources embarquées, à la manière de Sagot & Villemonte de la Clergerie (2009), qui repéraient des formes suspectes dans les phrases sur lesquelles l'analyse syntaxique échouait.

Ce travail tente d'ailleurs de pousser plus loin cette fouille d'erreurs en proposant un ensemble d'idées qui concourraient à automatiser les procédés d'annotation des exemples, d'entraînement et sélection de classificateurs de surdétections, et de repérage des cas de figure linguistiques les plus susceptibles de faire apparaître des surdétections. Une telle chaîne de traitement serait éminemment utile, à notre avis, pour améliorer n'importe quel correcteur grammatical doté d'une API autorisant l'extraction de traits riches.

En dernière analyse, nous pensons avoir présenté de manière claire que les approches statistiques, symboliques et d'apprentissage machine peuvent faire bon ménage. Nous espérons de plus avoir montré que l'atténuation de surdétections dans un correcteur grammatical constitue une tâche « réelle » du traitement des langues, présentant des défis scientifiques intéressants et qui offre un retour bénéfique à une large communauté d'utilisateurs.

Bibliographie

Aït-Mokhtar, S., Chanod, J.-P., & Roux, C. (2002). Robustness beyond shallowness: incremental deep parsing. *Natural Language Engineering, 8*(3), pp. 121-144.

Atwell, E. S. (1987). How to detect grammatical errors in a text without parsing it. *Proceedings of the third conference on European chapter of the Association* (pp. 38-45). Morristown, NJ, USA: Association for Computational Linguistics.

Bernth, A. (1997). EasyEnglish: a tool for improving document quality. *Proceedings of the fifth conference on Applied natural language* (pp. 159-165). Morristown, NJ, USA: Association for Computational Linguistics.

Bigert, J., Ericson, L., & Solis, A. (2007). AutoEval and Missplel: Two Generic Tools for Automatic Evaluation. Reykjavik, Islande.

Brunelle, É. (1987). La détection automatique des fautes de syntaxe en français. Montréal: Université de Montréal.

Brunelle, É., & Charest, S. (2010). Présentation du logiciel Antidote HD. *TALN 2010.* Montréal.

Bustamante, F. R., & León, F. S. (1996). GramCheck: A Grammar and Style Checker. *16th COLING*, (pp. 175-181). Danemark.

Callison-Burch, C., Koehn, P., Fordyce, C. S., & Monz, C. (2007). *Proceedings of the Second Workshop on Statistical Machine Translation.* Prague: Association for Computational Linguistics.

Charniak, E. (2000). A maximum-entropy-inspired parser. *Proceedings of the 1st North American chapter of the Association for Computational Linguistics* (pp. 132-139). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Chodorow, M., Tetreault, J. R., & Han, N.-R. (2007). Detection of Grammatical Errors Involving Prepositions. *4th ACL-SIGSEM Workshop on Prepositions* (pp. 25-30). Prague: Association for Computational Linguistics.

Clément, L., Gerdes, K., & Marlet, R. (2009). Grammaires d'erreur – correction grammaticale avec analyse. *16è TALN.* Senlis, France.

Collins, M. (2003). Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics, 29*(4), pp. 589-637.

Cook, D., Caragea, D., & Honavar, V. (2004). Visualization for Classification Problems, with Examples Using Support Vector Machines. *Proceedings of the COMPSTAT 2004, 16th Symposium of IASC.* Prague.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification.* New York: John Wiley and Sons.

Dugast, L., Senellart, J., & Koehn, P. (2007). Statistical Post-Editing on SYSTRAN's Rule-Based Translation System. *Second Workshop on Statistical Machine Translation* (pp. 220-223). Prague: Association for Computational Linguistics.

Fontenelle, T. (2005). Dictionnaires et outils de correction linguistiques. *X-2*, pp. 119-128.

Fontenelle, T. (2006). Les nouveaux outils de correction linguistique de Microsoft. *TALN*, (pp. 3-19). Louvain.

Foster, J. (2004). Parsing ungrammatical input: An evaluation procedure. *Proceedings of the 4th International Conference on Language Resources and Evaluation*, (pp. 2039-2042).

Foster, J. (2005). Good Reasons for Noting Bad Grammar: Empirical Investigations into the Parsing of Ungrammatical Written English. Dublin, Irlande: University of Dublin.

Foster, J., & Vogel, C. (2004). Parsing Ill-Formed Text Using an Error Grammar. *Artificial Intelligence Review, 21*(3-4), pp. 269-291.

Golding, A. R., Roth, D., Mooney, J., & Cardie, C. (1999). A Winnow-Based Approach to Context-Sensitive Spelling Correction. *Machine Learning*, (pp. 107-130).

Hall, M., Eibe, F., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations, 11*(1).

Helfrich, A., & Music, B. (2000). Design and evaluation of grammar checkers in multiple languages. *Project notes and demonstration at the 18th COLING*, (pp. 1036-1040). Saarbrücken, Allemagne.

Kanal, L. N., & Chandrasekaran, B. (1972). On Linguistic, Statistical, and Mixed Models for Pattern Recognition. *Frontiers of Pattern Recognition* (pp. 161-185). Academic Press.

Klavans, J. L., & Resnik, P. (1996). The balancing act : combining symbolic and statistical approaches to language. Cambridge, Mass.: MIT Press.

Knutsson, O., Pargman, T. C., & Eklundh, K. S. (2003). Transforming grammar checking technology into a learning environment for second language writing. *Proceedings of the HLT-NAACL 03 workshop on Building educational* (pp. 38-45). Morristown, NJ, USA: Association for Computational Linguistics.

Lee, J., & Seneff, S. (2008). Correcting Misuse of Verb Forms. *ACL-08: HLT*, (pp. 174-182). Columbus, Ohio.

Les Logiciels Machina Sapiens inc. (1999). *Manuel d'utilisation du Correcteur 101 pour Linux*. Récupéré sur http://robot.gmc.ulaval.ca/interne/documentation/man101linux.html

Manning, C. D., & Schütze, H. (1999, Juin 18). *Foundations of Statistical Natural Language Processing* (éd. 1). The MIT Press.

Morin, R. (1995). Sur l'intégration du correcticiel à la didactique du thème français.

Mudge, R. (2010, June). The Design of a Proofreading Software Service. *Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics* (pp. 24-32). Los Angeles, CA, USA: Association for Computational Linguistics.

Naber, D. (2003). *A rule-based and grammar checker.* Technische Fakultät, Universität Beilefeld.

Napolitano, D., & Stent, A. (2009). TechWriter: An Evolving System for Writing Assistance for Advanced Learners of English. *CALICO Journal, 26*(3), pp. 611-625.

Platt, J. (1998). Machines using Sequential Minimal Optimization. Dans B. Schoelkopf, C. Burges, & A. Smola (Éds.). MIT Press.

Sag, I. A., Kaplan, R., Karttunen, L., Kay, M., Pollard, C., Shieber, S., et al. (1986). Unification and Grammatical Theory. *Proceedings of the 1986 West Coast Conference on Formal*, (pp. 238-254).

Sagot, B., & de la Clergerie, É. (2009). Fouille d'erreurs sur des sorties d'analyseurs. *Traitement Automatique des Langues, 49*(1), pp. 41-60.

Sågvall Hein, A. (1998). A chart-based framework for grammar checking: Initial studies. *The 11th Nordic Conference of Computational Linguistics.*

Scannel, K. P. (2011). *An Gramadóir*. Récupéré sur http://borel.slu.edu/gramadoir/index.html

Schwischay, B. (2003). *Introduction à la syntaxe structurale de L. Tesnière.* Osnabrück: Universität Osnabrück.

Simard, M., Goutte, C., & Isabelle, P. (2007). Statistical Phrase-based Post-editing. *Actes de NAACL HLT 2007* (pp. 508-515). Rochester, NY: Association for Computational Linguistics.

Sofkova Hashemi, S. (2001, May). Detecting Grammar Errors in Children's Writing: A Finite State. *13th Nordic Conference on Computational Linguistics.* Uppsala, Suède.

Souque, A. (2008). Vers une nouvelle approche de la correction grammaticale automatique. *Récital.* Avignon, France.

Stolcke, A. (2002). SRILM - an extensible language modeling toolkit. *Proceedings of the international conference on spoken language processing*, 901-904.

van Noord, G. (2004). Error mining for wide-coverage grammar engineering. *ACL '04: Proceedings of the 42nd Annual Meeting on Association* (p. 446). Morristown, NJ, USA: Association for Computational Linguistics.

van Noord, G. (2007). Using self-trained bilexical preferences to improve disambiguation. *IWPT '07: Proceedings of the 10th International Conference on* (pp. 1-10). Morristown, NJ, USA: Association for Computational Linguistics.

Wagner, J., Foster, J., & van Genabith, J. (2007, June). A Comparative Evaluation of Deep and Shallow Approaches to the Automatic Detection. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural* (pp. 112-121). Prague, Czech Republic: Association for Computational Linguistics.

Wedbjer Rambell, O. (2000). *Error Typology for Automatic Proof-reading Purposes.* Uppsala: Uppsala University, Department of Linguistics.

Whittaker, E., & Raj, B. (2001). Quantization-based language model compression. *EuroSpeech*, (pp. 33-36). Aalborg.

Wisniewski, G., Max, A., & Yvon, F. (2010). Recueil et analyse d'un corpus écologique de corrections orthographiques extrait des révisions de Wikipédia. *Traitement automatiques des langues naturelles.* Montréal.

Yarowsky, D. (1995). Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. *33rd meeting of the ACL*, (pp. 189-196). Cambridge, MA.

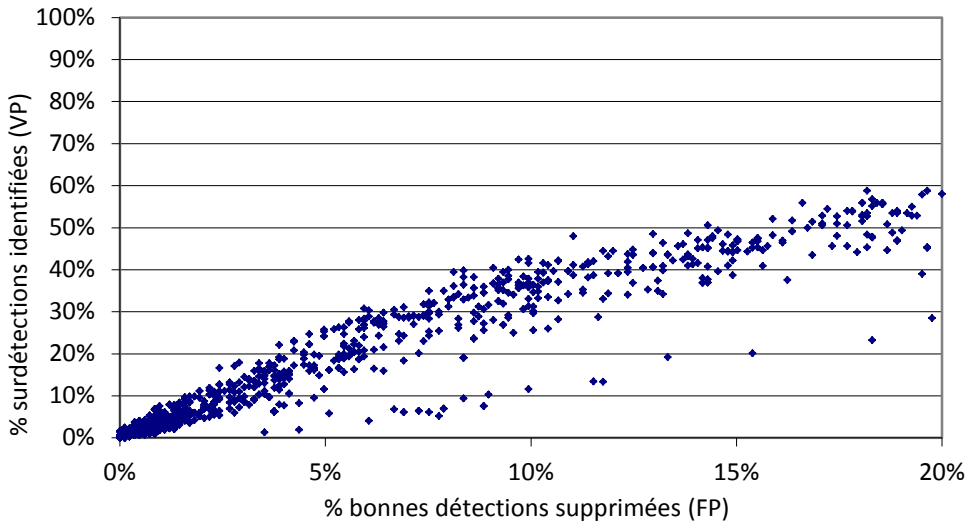
Young-Soog, C. (1998). Improvement of Korean Proofreading System Using Corpus and Collocation Rules. *Language, Information and Computation (PACLIC12)*, (pp. 328-333).

Annexe A
Classificateurs obtenus pour les 14 détections de Scorali

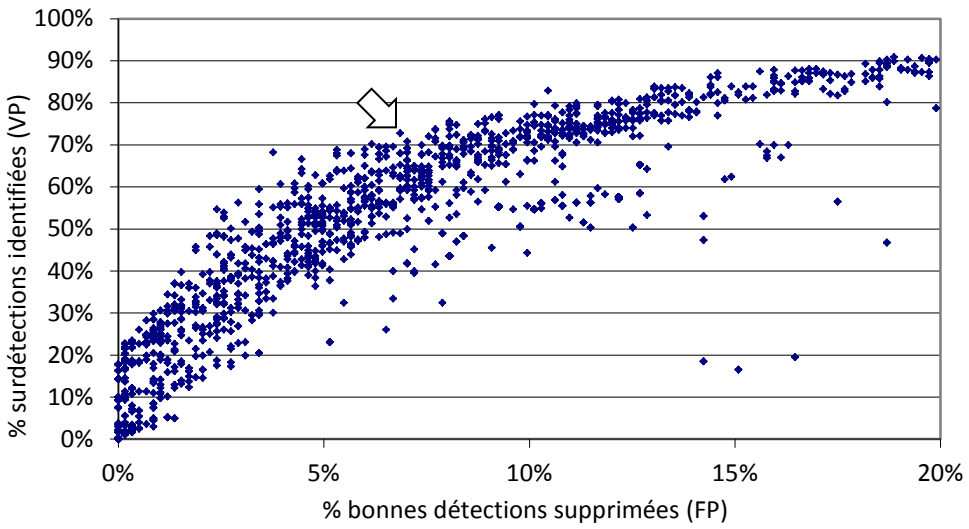
ScoraliA : Sélection de classificateurs

Cette section montre les classificateurs entraînés pour ScoraliA, pour chacune des 14 détections étudiées. Chaque graphique indique à l'aide d'une flèche le classificateur retenu et livré à Druide. Si aucune flèche n'apparaît sur le graphique d'une détection donnée, c'est qu'aucun classificateur ne répondait au cahier des charges pour cette détection.

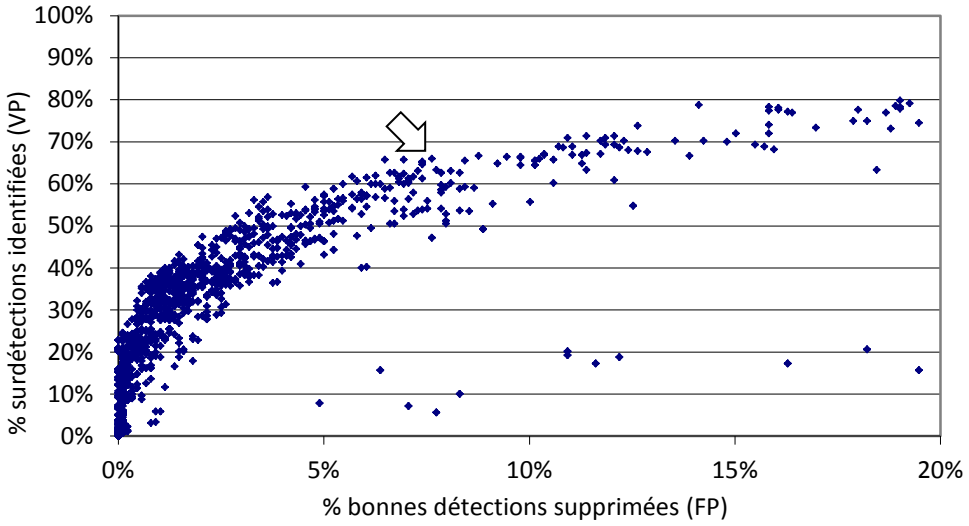
ACCORD



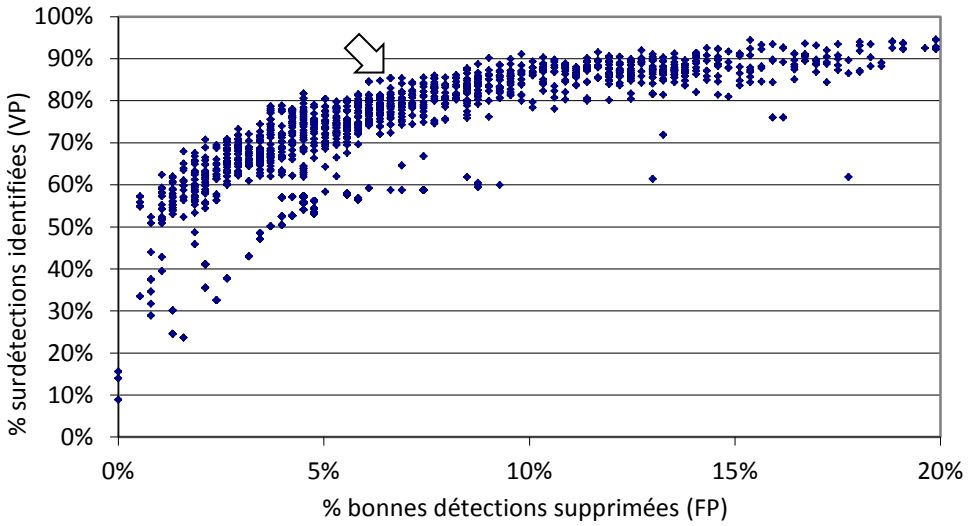
APOS



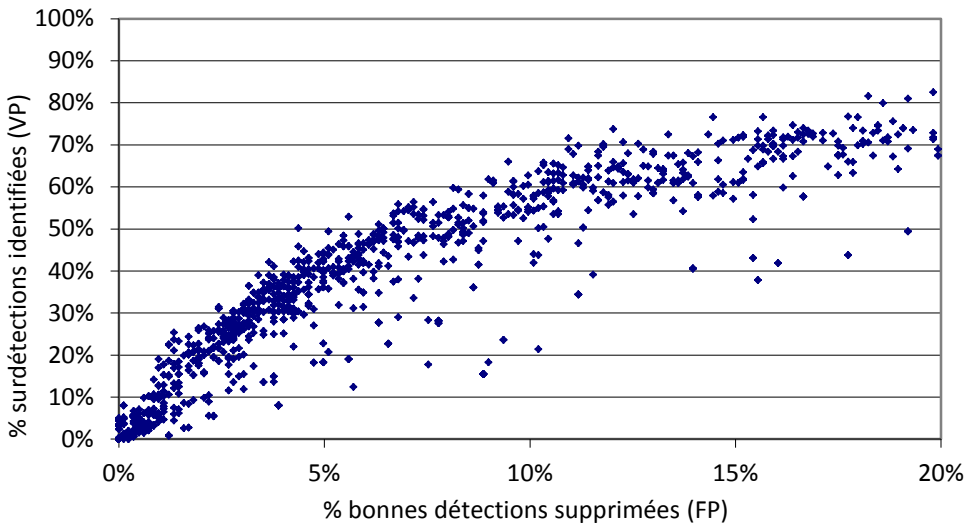
CONJUG



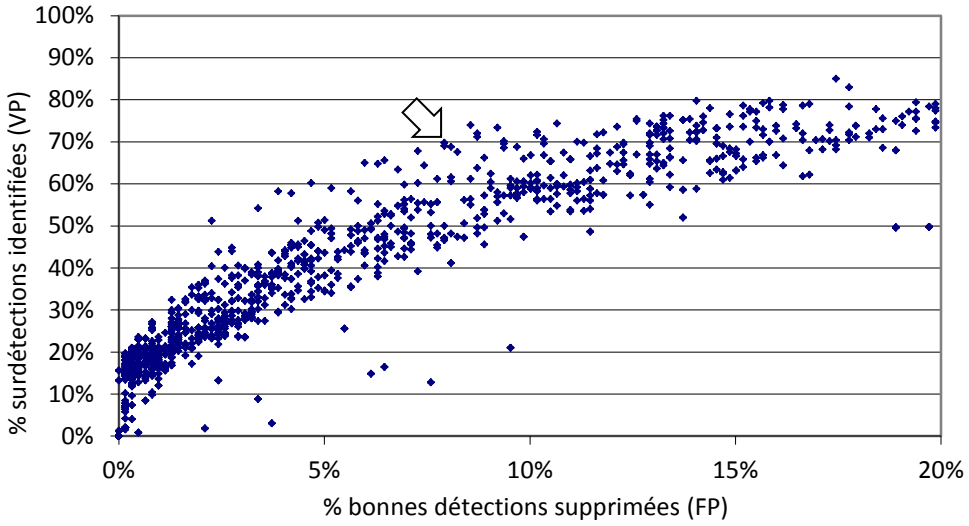
ÉLISION



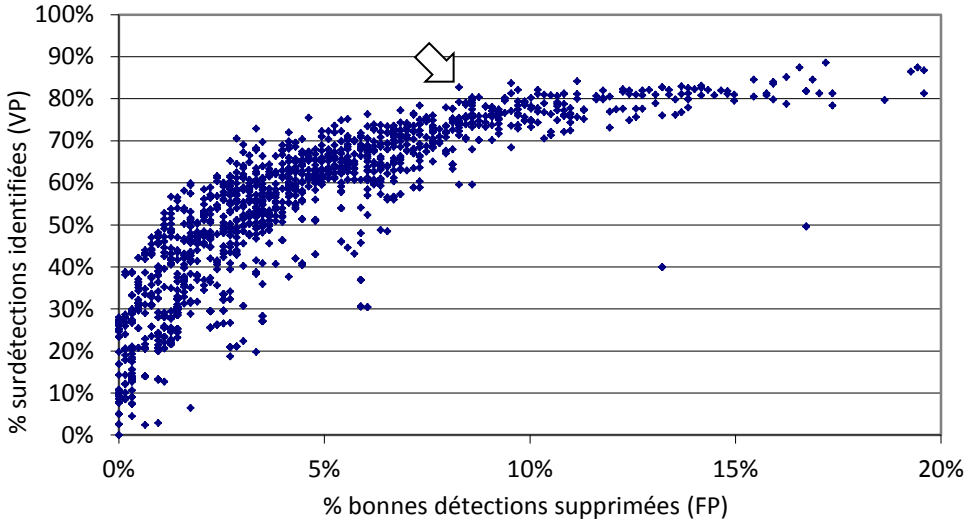
ER/EZ



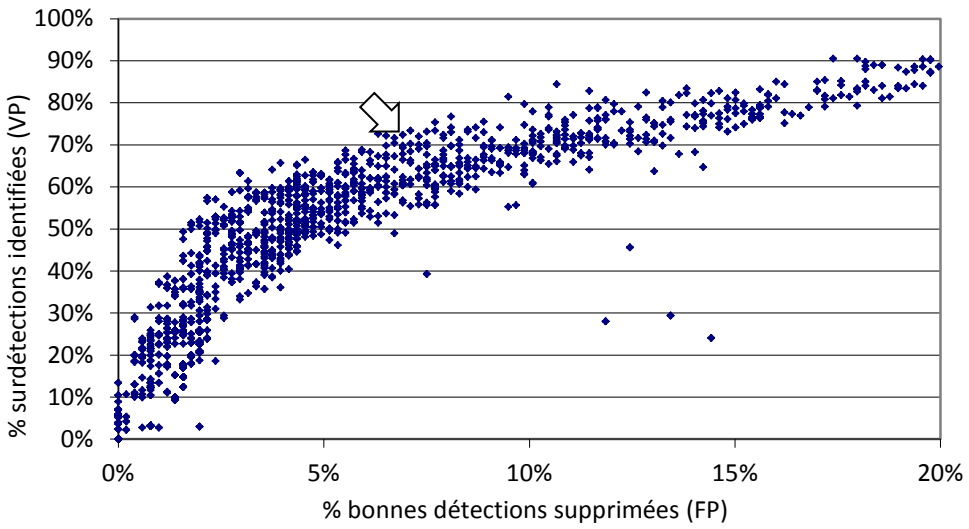
INVAR



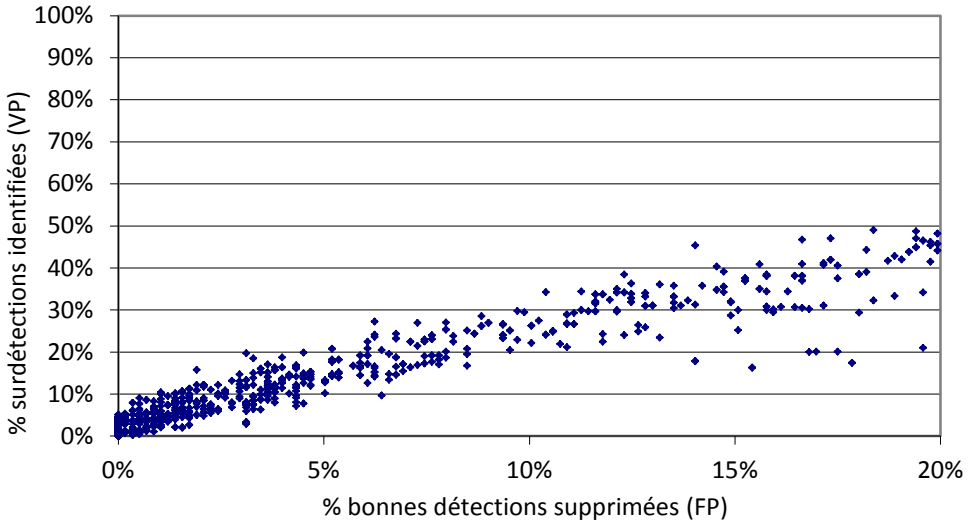
LA/LÀ



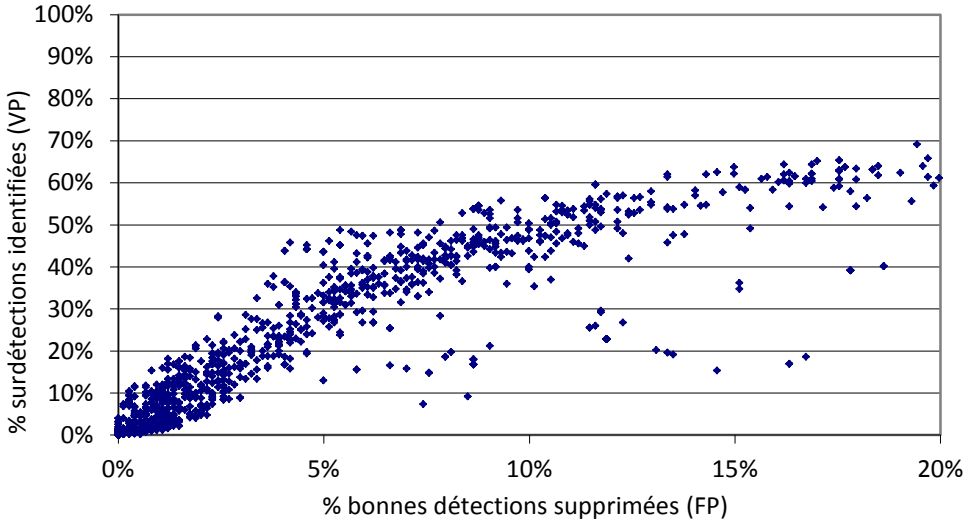
MAJ



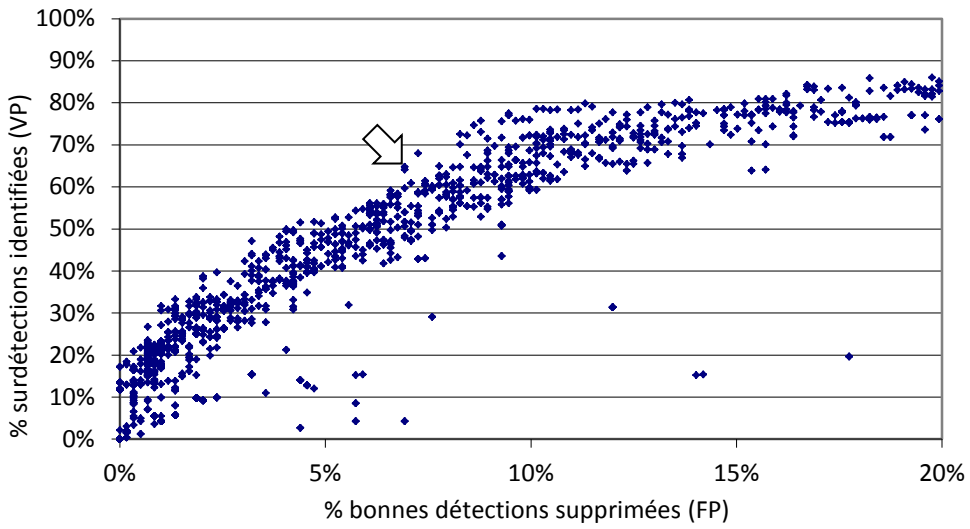
MODE



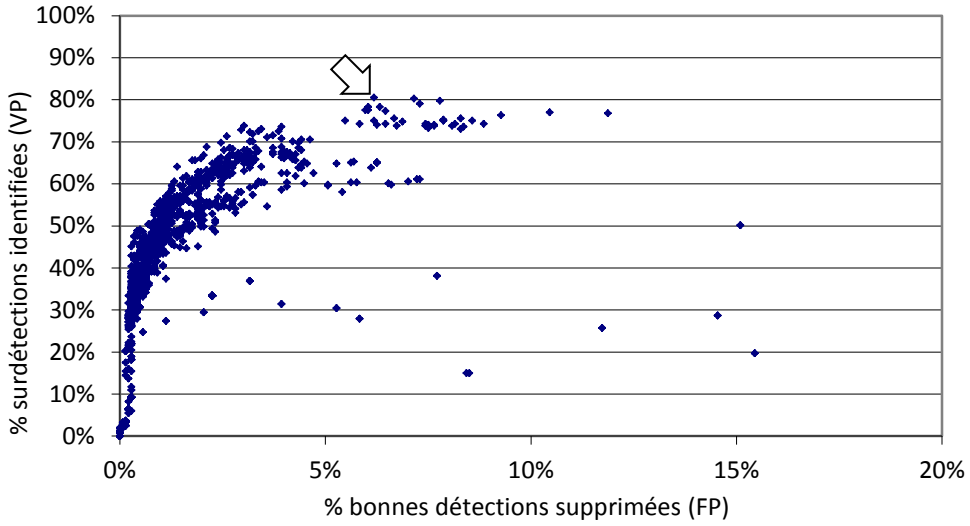
OU/OÙ



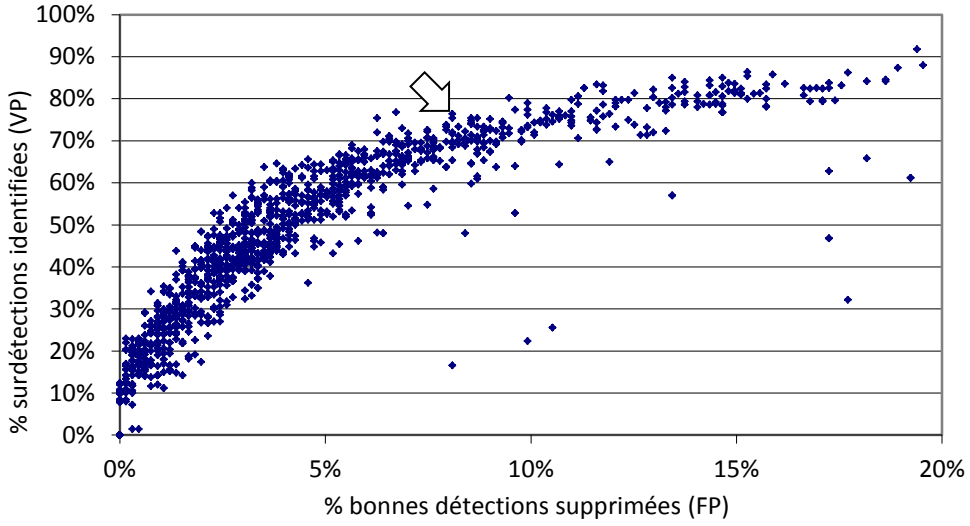
PP/INF



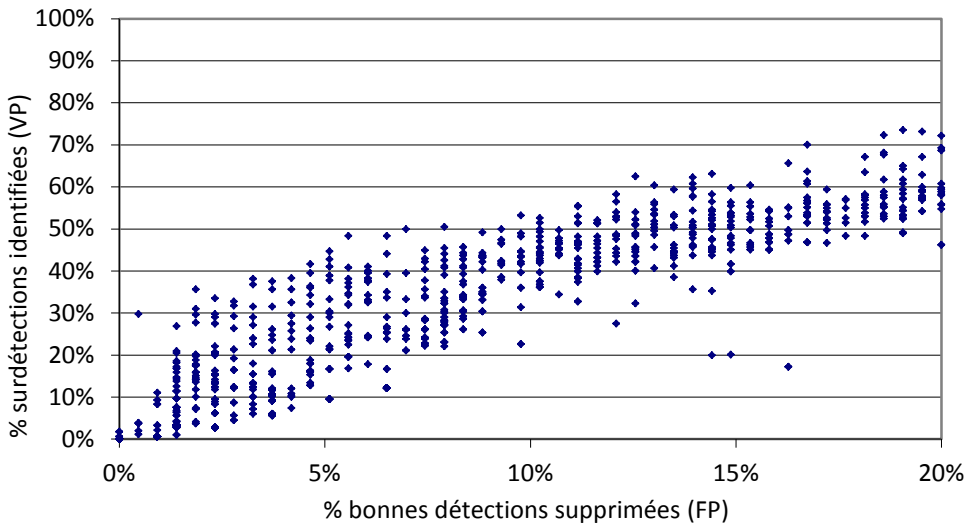
PP INV



PP/
V.CONJ



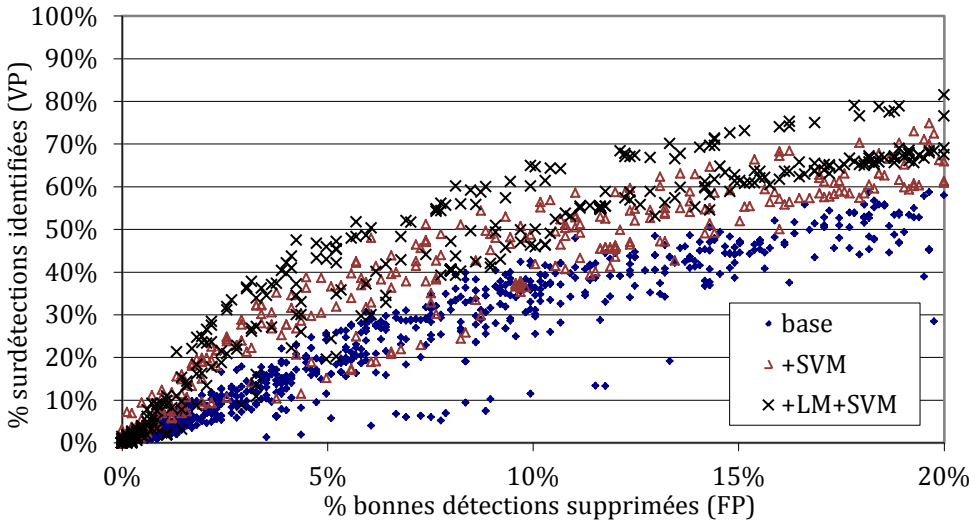
QUE/
DONT



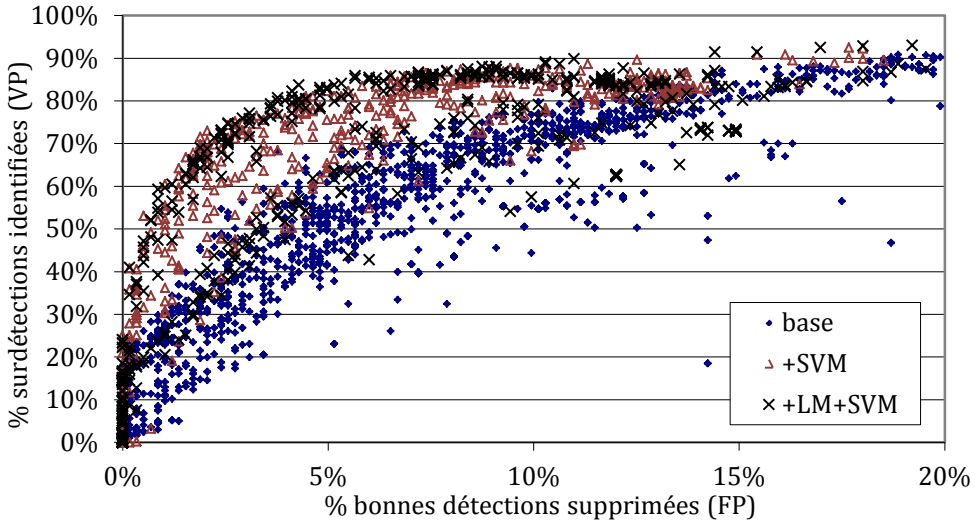
ScoraliB : Influence des SVM et des modèles de langue

Cette section présente l'influence des SVM et des modèles de langue sur les performances de classificateurs. Ceci constitue l'expérience ScoraliB. Pour que la comparaison soit plus aisée, chaque graphique contient les résultats de ScoraliA (série de points étiquetée « base »). La série de classificateurs +LM n'a pas été indiquée, afin d'alléger les figures.

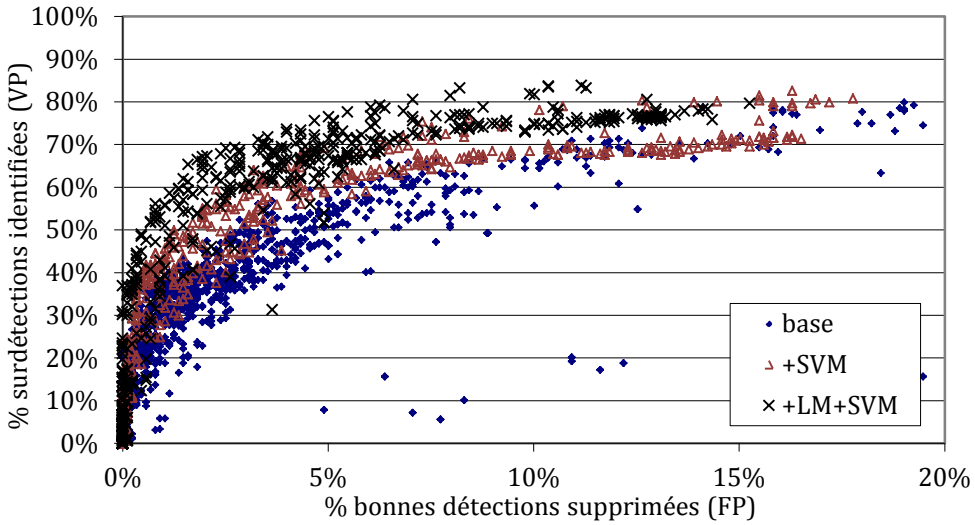
ACCORD



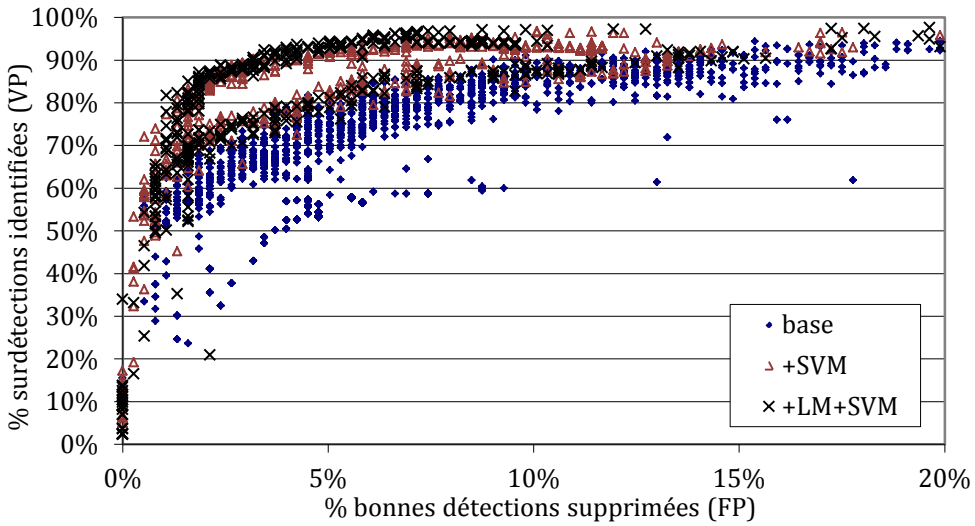
APOS



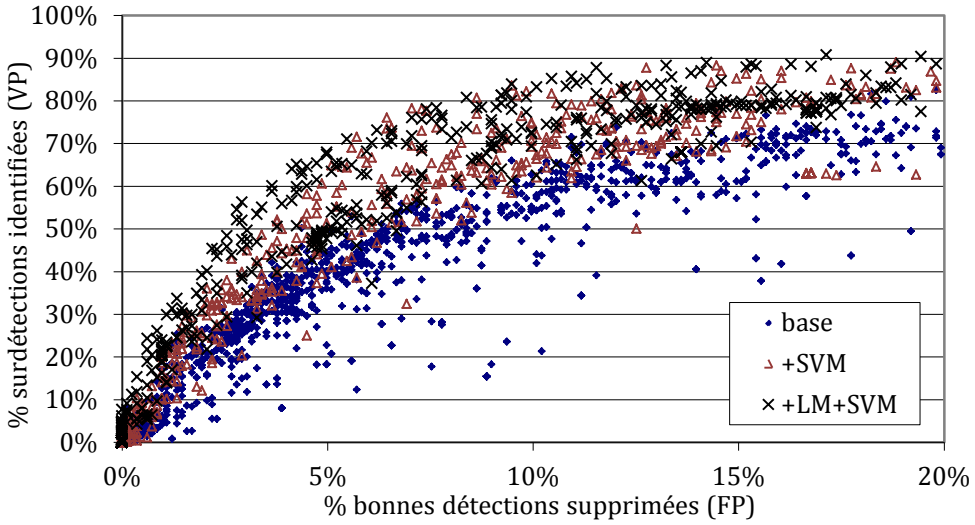
CONJUG



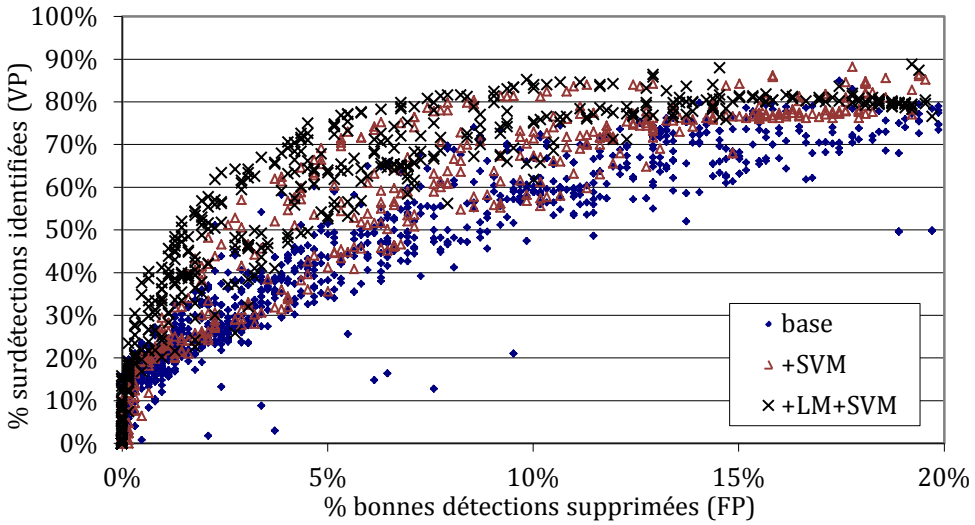
ÉLISION



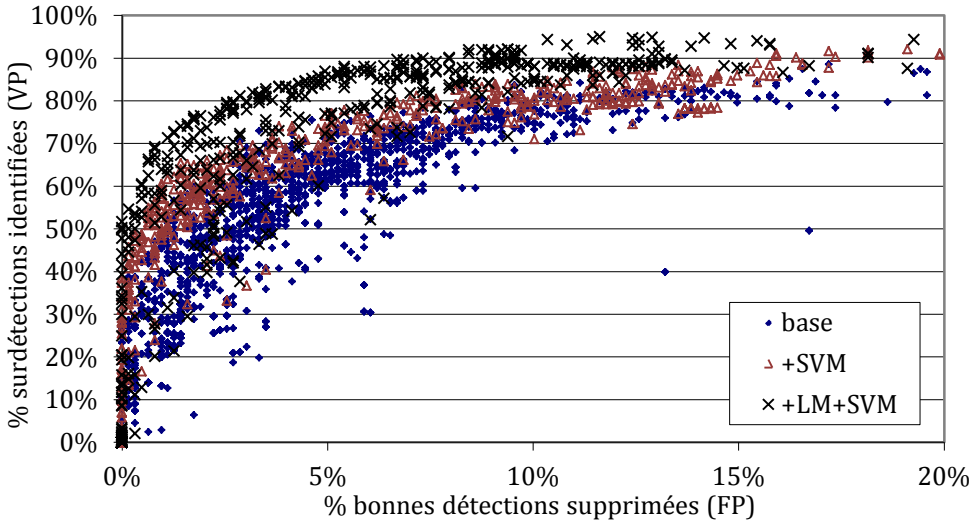
ER/EZ



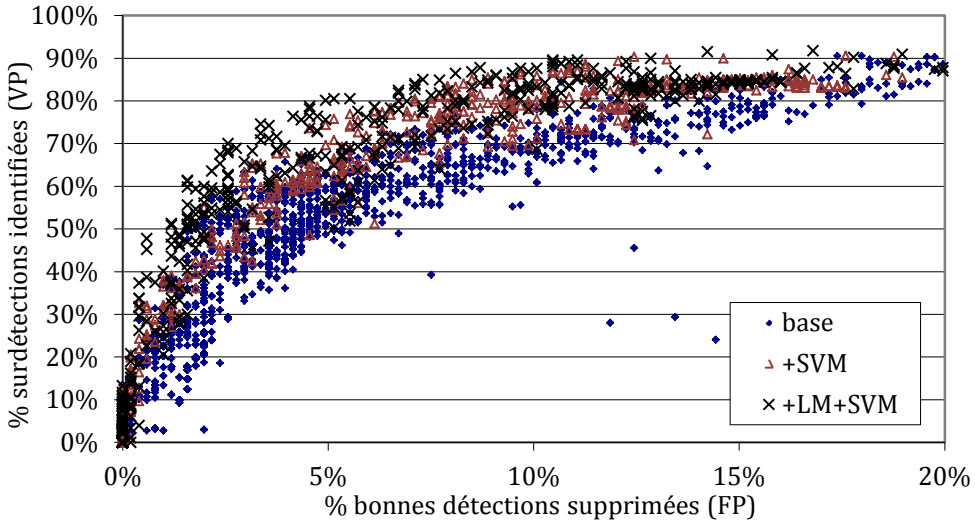
INVAR



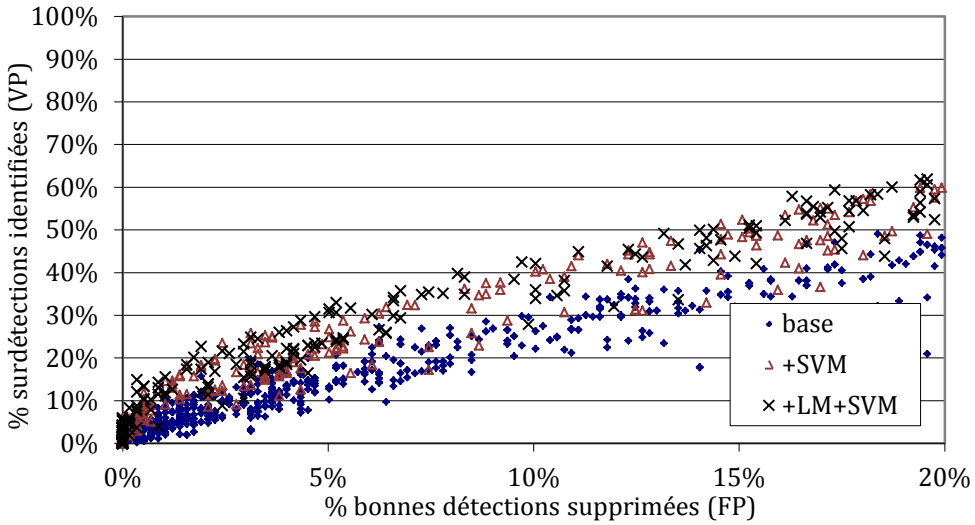
LA/LÀ



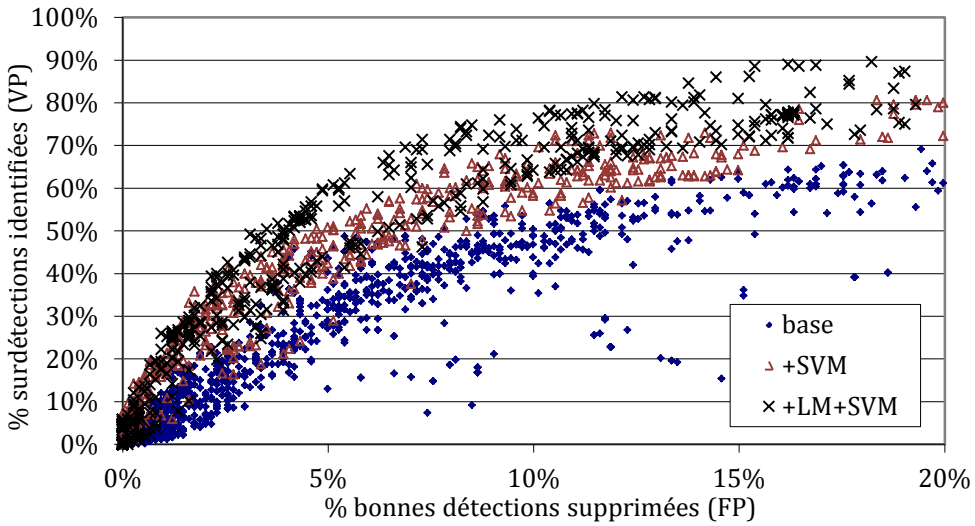
MAJ



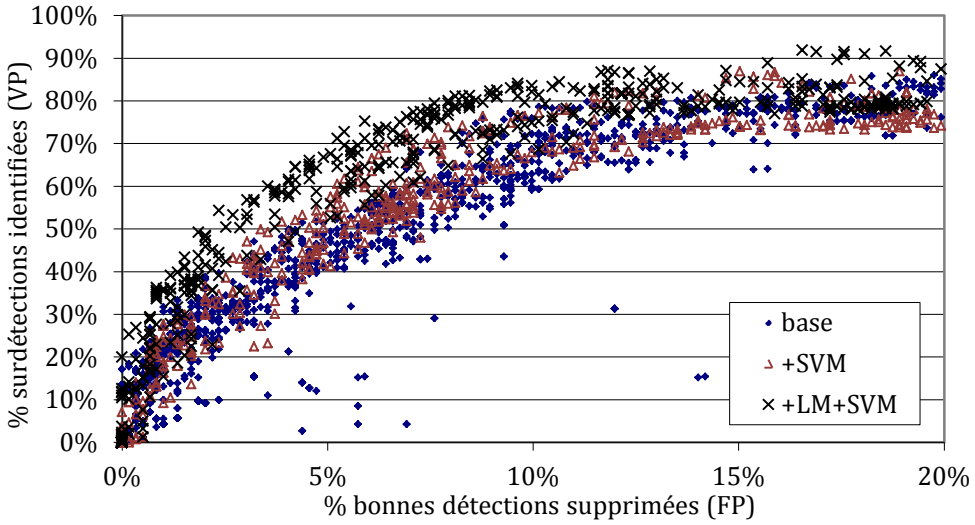
MODE



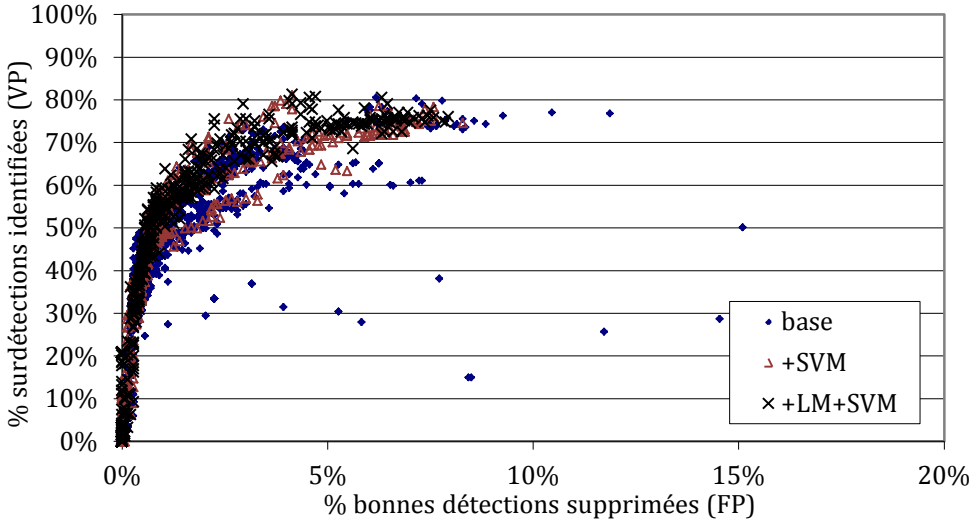
OU/OÙ



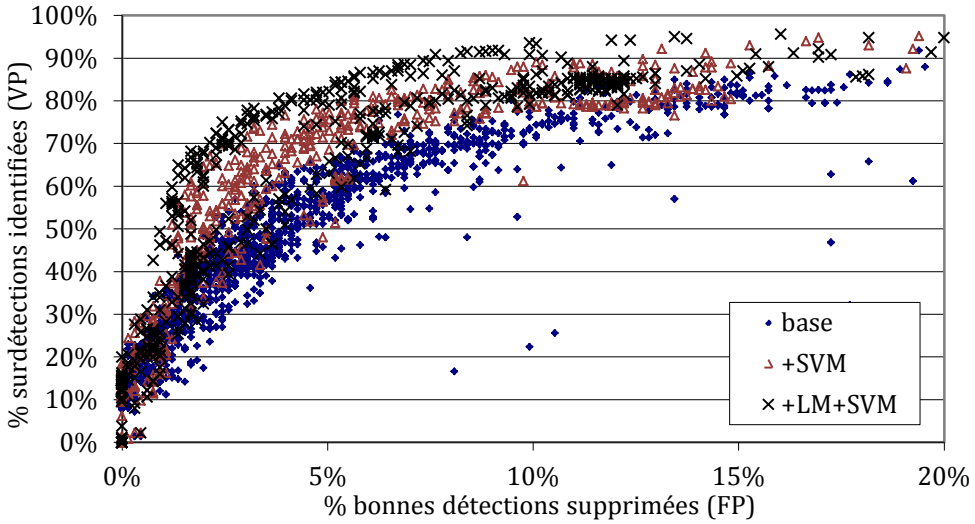
PP/INF



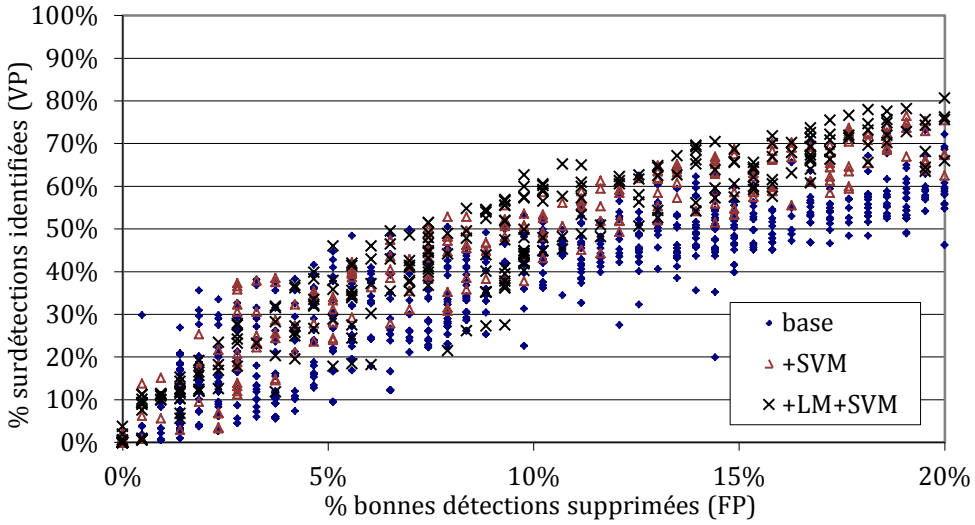
PP INV



PP/
V.CONJ



QUE/
DONT


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
% surdétections identifiées (VP)
0%
5%
10%
15%
20%
% bonnes détections supprimées (FP)
base
+SVM
+LM+SVM

Annexe B
Article accepté à la conférence CICLing 2011

Le lecteur trouvera dans cette annexe une copie de l'article *Reducing Overdetections in a French Symbolic Grammar Checker by Classification*, soumis et accepté à la conférence CICLing 2011[1]. Cet article, rédigé par nos collaborateurs et nous, résume une partie des résultats obtenus dans ce travail.

La référence complète de l'article est donnée ci-dessous.

Gotti, F., Langlais, P., Lapalme, G., Charest, S. & Brunelle, É. (2011). Reducing Overdetections in a French Symbolic Grammar Checker by Classification, *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing – Volume Part II* (pp. 390–401), Tokyo, Japon.

[1] http://www.cicling.org/2011/

Reducing Overdetections in a French Symbolic Grammar Checker by Classification

Fabrizio Gotti†, Philippe Langlais†, Guy Lapalme†,
Simon Charest‡, and Éric Brunelle‡

†DIRO/Univ. de Montréal
C.P. 6128, Succ Centre-Ville
H3C 3J7 Montréal (Québec) Canada
http://rali.iro.umontreal.ca

‡Druide Informatique
1435 rue Saint-Alexandre, bureau 1040
H3A 2G4 Montréal (Québec) Canada
http://www.druide.com

Résumé We describe the development of an "overdetection" identifier, a system for filtering detections erroneously flagged by a grammar checker. Various families of classifiers have been trained in a supervised way for 14 types of detections made by a commercial French grammar checker. Eight of these were integrated in the most recent commercial version of the system. This is a striking illustration of how a machine learning component can be successfully embedded in `Antidote`, a robust, commercial, as well as popular natural language application.

1 Introduction

Even though most modern writers use, often unknowingly, the grammar checker embedded in Microsoft Word, few NLP researchers have addressed the problem of improving the quality of the grammatical error detection algorithms [1,2]. Clément et al. [3] suggest that this could be explained by the lack of an annotated error corpus and by the close link that exists between a grammar checker and the proprietary word processor that embeds it.

Bustamante and Léon [4] present a typology of errors often encountered in Spanish and describe how the GramCheck project dealt with them. They distinguish structural errors (e.g. bad prepositional attachments) from non structural ones (e.g. subject verb agreement). The former is dealt with by crafting rules encoding typical errors that are added to the language parsing rules or by using auxiliary grammars on an ad hoc basis. The latter is dealt by loosening the unification process within the parser. These developments are quite complex and require a fine tuning of linguistic heuristics used within the parsing process.

Two main approaches to grammar checking have been taken by researchers. The first approach consists in comparing the sentence to proofread against a model of proper language use (a positive grammar). For instance, [5] propose using n-grams to create a language model of lemmas and part-of-speech tags (POS) occurring in proper English text. The second strategy seeks to create negative grammars in order to represent erroneous language constructs, like in [6] or [7]. Both approaches will often use a corpus of correct and faulty sentences to learn sequences of words that are then compared with the text to check. [8]

propose the use of grammar error rules derived from a normal grammar's rule so that the relationship between the correct rule and its derived error rules reflects a possible error as well as the correction to apply. Finding a representative training corpus is a challenge for these approaches, although one could use the one described in [9] or derived from it [10], but more important is the fact that regular expressions cannot reliably detect errors between distant words.

Sofkova Hashemi [11] also uses (positive) regular grammars on POS tags to detect grammatical errors. Using a notion of automata subtraction, she builds on the idea that if a coarse grammar (e.g. not taking into account number and gender agreement) can parse a sentence but not a more precise one, then there is probably an error in this sequence and a detection is made.

It is natural to think that a good grammar checker should strive to reach two conflicting goals : detect all errors present, offering a good *recall*, while avoiding false flags (henceforth *overdetections*), offering a good *precision*. But these goals are not equal in the eyes of the end-user. Indeed, a low precision is a major source of dissatisfaction for them : The overdetections give an (often false) impression that the grammar checker is incorrect in all of its suggestions. Indeed, when Microsoft researched their customers in order to properly design the grammar checker incorporated into their Office suite, they concluded [12] that "increasing precision and decreasing the false flag per page rate have had a higher priority than recall for these grammar checkers." This is corroborated by [13].

In this paper, we focus on a new NLP task : the identification of *overdetections*, i.e. grammatical error detections erroneously made by the system on flawless excerpts of text. We hope to demonstrate that this task presents interesting scientific challenges while offering some feedback to a large community of end-users.

We propose to tackle this task by training classifiers in a supervised way in order to recognize these overdetections. Our work is very different from the ones alluded to above because we are positioned *downstream from* the grammar checker. Resorting to a post-processing strategy has several potential benefits. First, the approach we propose can in principle be adapted to another grammar checker or to other types of errors than those we studied here (see Section 2.3). Second, we already mentioned that modifying an existing parser to account for ill-formed input is a difficult enterprise, one that we avoid here. In fact, the task we address is relatively simpler : we do not locate the errors or suggest a correction because this has already been done by the grammar checker.

We present our project in section 2. In section 3, we describe our approach to the overdetection problem. Results are presented in section 4. Contributions and new perspectives are presented in section 6.

2 ScoRali

The development of a grammar checker or its improvement is a complex endeavor involving many strategic choices as described by [12]. Here, we describe ScoRali, resulting from the close collaboration between `Druide Informatique`

(Druide) and researchers from RALI. Druide has been, for many years, actively developing a symbolic French parsing technology called Analytix which is based on rich symbolic description dictionaries and on a dependency parser both built and maintained manually by a team of linguists. The parser can deal with complex syntactic phenomena such as coordination (complete and elliptic), extraposition, correlation, punctuation use and some categorial inference. Particular care has been devoted to the parser robustness in the case of lexical and syntactical errors. A correction module uses the syntactic trees to pinpoint errors and suggest appropriate corrections. This technology is embedded in many commercial products, including Antidote, a writing assistant developed for the French language.

2.1 Requirements

The goal of the project was to train classifiers in a supervised way to detect overdetections for 14 common error types processed by Analytix. The error types have been selected by Druide according to their frequency and their overdetection risk (see section 2.3). Used after Analytix processing pipeline, these classifiers would judge the quality of the detections in order to filter out those that would most probably not be appropriate in this context. To be considered worthwhile, a classifier should identify at least 66% of overdetections and not remove more than 10% of correct detections. The classifiers should fit Analytix's processing pipeline.

2.2 Methodology

Data preparation [1] was crucial in this project. For each type of error, Druide prepared a sample of about 1000 detections, separated on average into an equal number of overdetections and legitimate detections, produced by Analytix on "real" texts, representing different types of use of the application. Each occurrence was then annotated by a linguist as being an overdetection or not and was associated with the syntactical parse produced by Analytix. This parse gives the position of each word, its grammatical category and about fifty morphosyntactic attributes such as gender, number (before and after correction) and, for verbs, their mode, tense and person. Moreover, all syntactic relations between words in the sentence were given, allowing to rebuild the syntactic parse tree of the sentence including, for each node, its grammatical category and a confidence weight. Each word was associated with a number of semantic-syntactic tags chosen from more than a thousand available.

This data was used as a basis for the features that we extracted for training our classifiers (see section 3). To help us determine the best ones, Druide also provided, for each type of detection, a summary characterization of the most frequent overdetection contexts and a linguistically motivated estimate of what they felt were the most suitable identification features.

1. This data is unfortunately not available to the community.

2.3 Types of errors

After many annotation and development cycles, 14 detection types were studied. Some of them are quite specific with respect to the linguistic phenomena they detect, e.g. the confusions between two words, like QUE/DONT — confusion between *que* ("that") and *dont* ("of that", used with a verb requiring a preposition) — or OU/OÙ — confusion between the conjunction *ou* ("or") and the adverb and pronoun *où* ("where"), which share the same pronunciation and differ only by the grave accent, a common source of error in French texts. An example of a good detection and a incorrect one is shown in Figure 1 for QUE/DONT. In these examples, the underlined word is the site of the detection and $*$ indicates an overdetection. An English translation of the original text and its correction is also provided.

`Je comprends ce que tu dis mais pas ce que` *`[dont]`* `tu parles.`
I understand what you say but not **what** *[of what]* you speak.

`Mais bon dieu que` *`[*dont]`* `les adultes s'amusent!`
But gosh **what** *[*of what]* fun these adults have!

Figure 1. An example of a good detection (top) and of an overdetection (bottom) for QUE/DONT. This type of detection is concerned with the confusion between 2 French words, "que" (what) and "dont" (of what).

Other detections are more general, in the sense that a given detection, like PP/VC (confusion between the past participle of a verb and its other conjugations) could encompass numerous different linguistic manifestations, given that it applies to many inflected forms of different verbs, sometimes with intervening words within the ill-formed construct. We give an example of such an overdetection for PP/VC in Figure 2 below.

`Roman ou récit, la ≪ Collection blanche ≫ de Gallimard éblouit`
`[*ébloui].`
Novel or story, the ≪ Collection blanche ≫ from Gallimard **dazzles** *[*dazzled].*

Figure 2. An example of an overdetection for PP/VC, the confusion between the past participle of a verb and its other forms.

It should be pointed out that there are many different types of texts in the training corpus : Some sentences were extracted from Wikipedia articles (including some headers), others from Internet discussion boards or scientific texts, etc. Some were even text messages without any diacritical marks, sometimes resorting to *phonetic* spelling. The quality therefore greatly varies.

One observable consequence of the poor quality of some of these sentences is that some overdetections result from other problems in the same sentence.

In the example of Figure 3, the correction of ***le***[*the (masc.)*] into ***la***[*the (fem.)*] is proposed because the word ***marché***[*deal (masc.)*] is misspelled ***marche***[*step (fem.)*]. As explained by [12] and [5], the (obviously poor) French of the writer could have been taken into account when making corrections, here.

`Ils veulent un libéralisme VRAI, accepte le [*la] marche mais...`
They want a TRUE liberalism, accepts **the (masc.)** *[*the (fem.)]* step but...

Figure 3. An example of an overdetection for ACCORD : an agreement error either in number or gender. In the translation, we purposely introduced errors in "TRUE" (capitalization), "accept" and "step" (the writer misspelled "deal") to illustrate the errors in the French original text, for the corresponding words.

3 Classification of detections

3.1 Feature extraction

The manual inspection of hundreds of instances of correct and incorrect detections (like those presented in Figures 2 and 3) shows that words in the neighborhood of the detections made by `Analytix` can guide the classification of a given detection. This context can simply be words before and after the detection or, since the training data includes the syntactical parse produced by `Analytix`, head words or dependents. For instance, for the confusion OU/OÙ, it is rather obvious for a French speaker that, whenever the word "là" precedes a potential confusion, a "où" is expected, rather than "ou". Similarly, for the confusion QUE/DONT, if the head word for the site of the confusion is a verb calling for a prepositional object, then "dont" should be used. Other features of the word flagged as a detection are important. For instance, when detecting capitalization errors, it is not advisable to correct a capitalized word when it follows another capitalized word : they probably participate in a named entity.

As a consequence of the previous observations, we selected a set of more than 1200 features per detection, among which :

The features of the word at the site of the detection. They are : its case, its length in letters, its gender, its number (for a noun), its tense, its number (for a verb), its part of speech, its position in the sentence, as well as features specific to the parser used by `Analytix`, for instance "verbs ending in -yer than can be confused with a noun".

The features of the words surrounding the site of the detection. The same features as those of the previous item, but this time for the words preceding and following the detection, as well as for the head word of the detection, in the parse.

The features of the detection itself. Precisely, the certainty with which `Analytix` proposes a correction, and the features of the correction proposed as a replacement for the word detected.

The features of the sentence in which the detection is found. That is, its length in words, the numbers of dependency relations identified, and the number of unknown words [2].

The nature of the dependency links in which the word detected participates. Namely, the links between the detection and its head word or its possible dependents. Here, we identify the usual relations, like "noun adjunct" or others, more specific to the grammar checker, e.g. the French "tel que" ("for instance").

Some ad hoc features, specific to each type of detection. For instance, we attempted to reframe the classification problem at hand as a word sense disambiguation (WSD) task, for the confusion QUE/DONT. Indeed, if we examine the example in Figure 1, we can consider the site of the detection as a placeholder, and "que" and "dont" as potential "semantic" labels. The disambiguation process allows us then to determine which one of those labels to insert at the site of the ambiguity. This is similar in spirit to the strategy adopted in [14] for fixing context-sensitive spelling corrections, that is, spelling mistakes resulting in existing words (e.g. *piece* versus *peace*). Among other strategies, we used an adaptation of the technique described by [15]. Unfortunately, our incursion into the WSD territory meant that we had to build and use external texts for modeling the context of *que* and *dont*, which precluded the integration of this classifier in `Analytix`.

3.2 Classifiers studied

To create and put to the test the required classifiers, we used the free software package Weka [16], written in Java [3]. This package allows the easy experimentation of numerous families of classifiers and possesses valuable features, like the visualization of data and classifiers as well as the preprocessing of training data. Moreover, it is possible to bypass the graphical interface and launch a classifier from the command line, which proved invaluable in our case when batch-processing thousands of classifier commands.

Weka allows the prototyping of roughly 50 classifiers, grouped into 8 families, e.g. Bayesian classifiers, decision trees, perceptrons, SVM and meta-classifiers. The latter combine other classifiers, by making them vote, for instance. Each of these classifiers is typically controlled by 1 to 20 hyperparameters, discrete or continuous. This causes a combinatorial explosion in the number of possible classifiers. Therefore, our first efforts focused on the exploration of classifiers that are rapid during training and classification, partly to satisfy the specifications for the project. Additionally, we preferred classifiers which were conceptually "simple", in order to facilitate their tuning, design, and eventual implementation within `Analytix`. These reasons led us to concentrate our efforts on symbolic classifiers (but see section 4.4), namely `rules.ConjunctiveRule`, `rules.DecisionTable`,

2. It is noteworthy that most of the features which are numerical counts are doubled : one is the count itself, the other is the count normalized by the length of the sentence.

3. `www.cs.waikato.ac.nz/ml/weka/`

`rules.JRip` (a propositional rule learner, like RIPPER [17]), `trees.ADTree` (alternating decision trees), `trees.DecisionStump`, `trees.J48` (C4.5 decision trees) and `trees.J48graft`.

Beyond the selection and parametering of the classifiers, Weka allows diverse pre-processing strategies on the training data. We first filtered features, to remove those which did not vary enough or varied too much among the instances of the training set (these features cannot be used to discriminate). Furthermore, for every type of detection, we tried different filtering strategies for their features, reducing in some case the 1200 features to a mere dozen. Naturally, this simplifies the training and test of the classifiers, but also their eventual implementation. We also varied the cost that Weka attributes to false negatives and false positives. Ultimately, for each of the 14 detection types, we tested about 4000 classifier settings in order to find one which would meet the requirements set by `Druide`. This exploration was made on a 16 dual-core computer cluster, with a computing time of 5 days for each detection.

4 Results

It is impossible to fit all the results obtained on all classifiers tested in this article. For this reason, in this section, we will detail the results for the detection PP/VC, for which we provided an example earlier in Figure 2. We chose this example because it lent itself quite successfully to the approach, and because it is representative of the results we obtained on several other detections. Also, it is an illustration of a detection arising in very varying contexts, which proves challenging. We will nonetheless present a global overview of the results for all the detection types later in this article.

4.1 The pp/vc detection

Figure 4 shows the performance of more than 3000 classifiers for the project SCORALI (it is the scatter plot cluster labeled "System". The x-axis represents the percentage of good corrections erroneously flagged as bad corrections by the classifiers (false positives), while the y-axis represents the percentage of bad corrections correctly identified as such (true positives). For each classifier, this data was obtained through 10-fold cross-validation on the training set.

Remarkably, the scatter plot cluster is relatively compact, forming a band encompassing the possible compromises each classifier offers. The choice of a classifier is made manually, based on this kind of figure, while striving to meet the requirements set by `Druide` (section 2.1). In our case, the classifier recommended for `Analytix` is the one whose data point is circled in the figure. It is a C4.5 decision tree, grafted and pruned, allowing the identification of 77 % of overdetections, at the cost of a loss of 8 % of good corrections. The decision tree classifies 88 % of instances correctly, with a substantial agreement of $\kappa = 0.76$ between all 10 folds of the cross-evaluation. Other classifiers with similar performances were discarded, either because they were too complex to implement or because they used too many features.

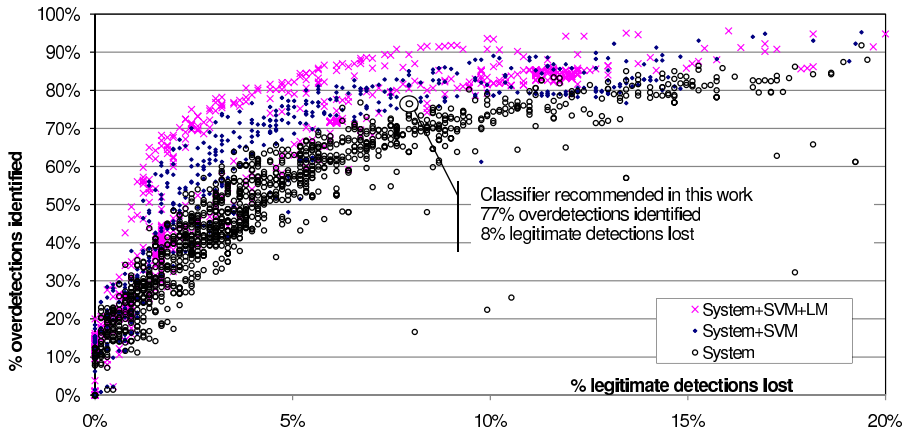


Figure 4. Classifiers tested for the detection PP/VC. Each point represents the performance of a single classifier, i.e. a compromise between the true and false positives. The graph shows 3 scatter plots, one for SCORALI per se (Section 4, labeled "System") and 2 others, resulting from further research (Section 4.4).

4.2 An overview of all detection types

We applied the strategy described in the previous section for all 14 types of detections provided by `Druide`. Although the lack of place prevents us from describing each error type, the overall results are presented in Table 1. We first observe that we succeeded in creating classifiers meeting the project's requirements for 9 out of the 14 detection types. They all are decision trees : when a group of classifiers proved equally good at classifying detections, we selected a decision tree among them. A closer inspection of the induced rules used in the decision trees did not allow the identification of features consistently present in all or most of the trees.

It is difficult to explain why certain detections lent themselves well to classification, and others not. Despite our best efforts, QUE/DONT could not find a good classifier, whereas some detections proved easy to process, although it seemed at first that rule induction would be difficult because they occurred in extremely different contexts, for different reasons and often in text of very poor quality.

Naturally, classification rules are induced more easily if `Analytix` overdetects within a certain language construct in a systematic way. This is the case for the detection LA/LÀ, where a manual inspection of the decision tree produced shows that 20 % of the overdetections occur when "la" is an article ending an abruptly truncated sentence, like in the instance "recommandations de **la** *[*là]*" (recommendations of **the** *[*there]*). The construct is always the same then : a missing noun adjunct preceded by the article.

It is also obvious that certain overdetections made by `Analytix` are very difficult to classify, even for a human being. The examples shown in Figure 2 for

PP/VC and in Figure 3 are striking. In the latter Figure, the instance "accepte **le** *[*la]* marche", one must really understand the sentence to know that the author meant "accepte le marché" ("accept the deal", where "marché" is masculine) rather than "accepte la marche" ("accept the step", where "marche" is feminine). The overdetection is then due to the missing acute accent on the final letter of "marche". It is highly likely that classifiers have a hard time with these cases, especially when these detections are flagged in varying contexts, for different reasons. It is the case for ACCORD, which can be an agreement error, either in gender or number. It could be interesting to further our research by creating two different detection types : ACCORD for gender and ACCORD in number.

Detection	% **fp**	% **tp**	**Detection**	% **fp**	% **tp**	**Detection**	% **fp**	% **tp**
ÉLISION	7%	87%	INV	8%	71%	QUE/DONT	9%	57%
LA/LÀ	9%	84%	MAJ	7%	71%	OU/OÙ	9%	49%
PP INV	6%	80%	PP/INF	7%	68%	ACCORD	9%	40%
PP/VC	8%	77%	CONJUG	8%	66%	MODE	9%	27%
APOS	6%	73%	ER/EZ	9%	65%			

Table 1. Complete results of ScoRali, as delivered to `Druide`.

4.3 Tests at `Druide` and implementation

The eleven best classifiers were therefore converted and integrated to `Analytix`, then tested on a corpus distinct from the one we used for their training, a recommended practice in the industry (see for instance [12]). These tests revealed that 3 classifiers degraded the performance of the grammar checker to a point where they had to be rejected. These classifiers were not necessarily those with the worst performances during training, but had to be removed nonetheless. The 8 remaining classifiers are part of the latest commercial version of `Analytix`. Among these, 3 are used as they are, and five are subject to an ad hoc test determining whether the engine will use them, based on the context of the detection. Finally, the user interface includes an on/off switch for these classifiers : The users are presented with a checkbox labeled "statistical filtering of detections". The user wishing not to miss any good detection can deactivate this setting, but will be presented with more false detections. The classifiers are on by default.

4.4 Better classifiers

As a requirement (see Section 2.1), the classifiers delivered to `Druide` had to be simple to interpret and to embed within `Analytix`. In order to measure the improvements that could be made to ScoRali, we studied the performances offered by SVM classifiers and we added features derived from language models

trained on the Canadian Hansard. The gains obtained are shown in Figure 4 by their respective scatter plot cluster, for detection PP/VC. SVMs (labeled "System+SVM") alone allow a gain of about 5% in identification of overdetections, compared to the classifiers delivered to `Druide`, at the cost of an increase in computational needs and a decrease in expressivity of the model created. The addition of language model features (labeled "System+SVM+LM") increases the number of true positives, for a further 10% gain.

5 Related Work

A fair number of studies have been dedicated to spelling correction (e.g. [18,14,19]). Grammar checking, which we believe is a useful component of a writing assistant tool, has received — somehow paradoxically — much less attention. However, we see many advantages to studying grammar checkers on their own. Indeed, we feel this naturally belongs to the field of *grammar engineering*. Behind this expression, we group activities as diverse as making parsing faster and more robust (e.g. [20]), adapting parsers to new domains (e.g. [21]), or simply improving existing parsers (e.g. [22]).

Actually, for certain types of detections, the classifiers we trained proved very useful as error mining tools within `Analytix`'s parsing grammar.[4] For instance, we noticed that `Analytix` has difficulty recognizing the expression "faire partie" (take part) and makes the overdetection "faire **partie** *[*partit]*" (take parted), for detection PP/VC. Although the particular classifier for PP/VC did not include such a feature, it would probably be interesting to detect the presence of the verb "faire" in the context leftward of the detection.

Thus, the work we conducted could prove, as a side effect, to be complementary to the studies made on error identification in wide-coverage grammars [22,23,24], but has the advantage of not requiring the modification of the grammar studied, a delicate task which is not always possible in a complex grammar maintained manually, especially in a commercial context.

6 Conclusion and Future Work

ScoRali allowed the creation and implementation into `Analytix` of 8 out of 14 classifiers identifying overdetections, downstream of the grammar checking engine. This successful transfer of technologies from the laboratory to a commercial product entailed the exploration of thousands of different classifiers, as well as a delicate balance between performance and technical constraints.

We think that this paper clearly shows that statistical and symbolic approaches can go hand in hand, and is indeed a very clear illustration of the kind of balancing act that such a combination requires [25].

4. This is one argument in favor of classifiers such as decision trees that can be easily interpreted against other ones such as SVMs.

Despite the fact that the corpus we used in this study can not be released to the scientific community, we hope to have shown that overdetection identification constitutes a "real" task in NLP, one that presents interesting scientific challenges while offering some feedback to a large community of end-users.

This work shows some avenues that we think are worth investigating. For the time being, certain detections seem not to lend themselves to the proposed approach, maybe because they occur in contexts which are too varied, thus defying rule induction. The work we conducted on using more features and more robust classifiers (see Section 4.4) for the PP/VC detection shows that there is room for improvement. This suggests further experiments to see if such gains carry over other detections.

One limitation of our work lies in the simplifying assumption that each detection within a sentence is independent of the other possible detections within the same sentence, although evidence shows that one actual error can trigger an overdetection.

Also, we feel the evolution of SCORALI poses a number of exciting questions. The data used to train the classifiers is not frozen in time : it was generated by `Analytix` at a given moment in its life cycle. Although our classifiers passed a number of regression tests at `Druide`, it remains to be seen whether they will withstand the likely changes that will happen over time (within `Analytix`, in detection statistics, etc.) or whether new training (or adaptation) will be required.

Références

1. Fontenelle, T. : Dictionnaires et outils de correction linguistiques. Rev. franç. de linguistique appliquée **X-2** (2005) 119–128
2. Véronis, J. : Texte : Correcteurs orthographiques en panne ? Blog du 6 juil. : `http://aixtal.blogspot.com/2005/07/texte-correcteurs-orthographiques-en.html` (2005)
3. Clément, L., Gerdes, K., Marlet, R. : Grammaires d'erreur – correction grammaticale avec analyse profonde et proposition de corrections minimales. In : 16è TALN, Senlis, France (2009)
4. Bustamante, F.R., León, F.S. : Gramcheck : A grammar and style checker. In : 16th COLING, Denmark (1996) 175–181
5. Napolitano, D., Stent, A. : TechWriter : An Evolving System for Writing TechWriter : An Evolving System for Writing Assistance for Advanced Learners of English. CALICO **26(3)** (2009) 611–625
6. Rider, Z. : Grammar checking using pos tagging and rules matching. In : Proceedings of the Class of 2005 Senior Conference. Computer Science Department, Swarthmore College (2005) 14–19
7. Souque, A. : Vers une nouvelle approche de la correction grammaticale automatique. In : Récital, Avignon, France (2008)
8. Foster, J., Vogel, C. : Parsing ill-formed text using an error grammar. Artif. Intell. Rev. **21**(3-4) (2004) 269–291

9. Foster, J. : Good Reasons for Noting Bad Grammar : Empirical Investigations into the Parsing of Ungrammatical Written English. PhD thesis, Department of Computer Science - University of Dublin (May 2005)
10. Foster, J. : Treebanks gone bad : Parser evaluation and retraining using a treebank of ungrammatical sentences. Int. J. Doc. Anal. Recognit. **10**(3) (2007) 129–145
11. Sofkova Hashemi, S. : Detecting grammar errors in children's writing : A finite state approach. In : 13th Nordic Conference on Computational Linguistics, Uppsala, Sweden (May 2001)
12. Helfrich, A., Music, B. : Design and evaluation of grammar checkers in multiple languages. In : Project notes and demonstration at the 18th COLING, Saarbrücken, Germany (2000) 1036–1040
13. Bernth, A. : Easyenglish : a tool for improving document quality. In : Proceedings of the fifth conference on Applied natural language processing, Morristown, NJ, USA, Association for Computational Linguistics (1997) 159–165
14. Golding, A.R., Roth, D. : A winnow-based approach to context-sensitive spelling correction. CoRR **cs.LG/9811003** (1998)
15. Yarowsky, D. : Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In : 33rd meeting of the ACL, Cambridge, MA (1995) 189–196
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H. : The WEKA Data Mining Software : An Update. SIGKDD Explorations **11, Issue 1**(10–18) (2009)
17. Cohen, W.W. : Fast effective rule induction. In : In Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann (1995) 115–123
18. Damerau, F. : A technique for computer detection and correction of spelling errors. Commun. ACM **7**(3) (1964) 171–176
19. Brill, E., Moore, R.C. : An improved error model for noisy channel spelling correction. In : ACL '00 : Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (2000) 286–293
20. Kiefer, B., Krieger, H.U., Carroll, J., Malouf, R. : A bag of useful techniques for efficient and robust parsing (1999)
21. Rimell, L., Clark, S. : Adapting a lexicalized-grammar parser to contrasting domains. In : EMNLP '08 : Proceedings of the Conference on Empirical Methods in Natural Language Processing, Morristown, NJ, USA, Association for Computational Linguistics (2008) 475–484
22. van Noord, G. : Using self-trained bilexical preferences to improve disambiguation accuracy. In : IWPT '07 : Proceedings of the 10th International Conference on Parsing Technologies, Morristown, NJ, USA, Association for Computational Linguistics (2007) 1–10
23. Sagot, B., de la Clergerie, E. : Fouille d'erreurs sur des sorties d'analyseurs syntaxiques. Traitement Automatique des Langues **49**(1) (2009) 41–60
24. de Kok, D., Ma, J., van Noord, G. : A generalized method for iterative error mining in parsing results. In : Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks (GEAF 2009), Suntec, Singapore, Association for Computational Linguistics (August 2009) 71–79
25. Klavans, J.L., Resnik, P., eds. : The balancing act : combining symbolic and statistical approaches to language. MIT Press (1996)