Apprentissage et Evaluation de Modèles de Langage par des Techniques de Correction d'Erreurs

Laurent Miclet, Jacques Chodorowski, Pierre Dupont

ENSSAT - IRISA : miclet@enssat.fr, chodorow@enssat.fr

EURISE - Université de Saint-Etienne : pdupont@univ-st-etienne.fr

RÉSUMÉ. *Cet article a pour but de décrire la mise au point et l'expérimentation de méthodes d'apprentissage de syntaxe à partir d'exemples positifs, en particulier pour des applications de Reconnaissance de la Parole et de Dialogue oral. Les modèles syntaxiques sont extraits des données par des méthodes d'inférence grammaticale symbolique et stochastique. Ils sont fondés sur des techniques de correction d'erreurs dans les séquences. Ils sont destinés à être intégrés dans une chaîne de traitement de la parole par Modèles de Markov cachés, dont les paramètres sont eux-mêmes obtenus par apprentissage à base de corpus. L'ensemble de ce travail a été réalisé dans le cadre du contrat 97-1B-004 avec France-Telecom (Centre National d'Etudes des Télécommunications).*

Dans la première partie de cet article, nous rappellons les distances entre séquences basées sur des opérations élémentaires de correction d'erreur, ou d'édition. Nous décrivons ensuite un algorithme classique d'inférence grammaticale fondé sur cette notion et nous montrons comment on peut l'améliorer par l'apprentissage des poids des opérations élémentaires d'édition. Nous abordons à cet endroit le problème de l'évaluation d'un concept appris seulement à partir d'exemples positifs, sans contre-exemples.

Par la suite, le modèle syntaxique est étendu en attribuant des probabilités aux règles de la grammaire. Celles-ci sont également apprises à partir des données. On dispose dans ce cadre d'un outil d'évaluation probabiliste de la qualité de l'apprentissage effectué : la perplexité d'une grammaire stochastique ; cependant pour obtenir des résultats significatifs, il faut être capable de probabiliser l'espace entier des séquences, ce qui implique de lisser la grammaire stochastique apprise. Une technique de lissage est proposée, qui permet alors d'évaluer l'apprentissage sur le corpus de données issues de l'expérimentation en dialogue oral.

MOTS CLÉS : *Inférence grammaticale régulière, analyse corrective, évaluation du modèle de language*

1 Introduction

1.1 Apprentissage Automatique et Traitement Automatique de la Langue Naturelle

Les techniques d'Apprentissage Automatique sont actuellement de plus en plus appliquées aux différents aspects du Traitement du Langage Naturel. Il y a en effet une convergence entre ces deux domaines qui s'explique en particulier par les raisons suivantes :

- Le besoin de traitement automatique du langage écrit et oral a considérablement augmenté : recherche automatique sur le web à partir du contenu des pages, reconnaissance de la parole, traduction automatique, etc. Pour ces applications, et beaucoup d'autres, les corpus disponibles sont désormais de taille suffisante pour que des algorithmes d'apprentissage puissent en extraire des modèles réalistes. Notons que ces corpus sont le plus souvent composés uniquement d'exemples positifs, ce qui pose un problème particulier aux algorithmes d'apprentissage (comment limiter la généralisation ?).
- Les algorithmes d'Apprentissage Automatique se sont développés selon d'une part un axe principalement symbolique (ILP, Inférence grammaticale algébrique, etc.), d'autre part selon un axe numérique (décision statistique bayésienne, réseaux connectionnistes, etc.). Actuellement la synthèse se fait grâce à des méthodes mixtes : symboliques-numériques par exemple, l'apprentissage de règles par arbres de décision, ou, comme on le verra plus loin, l'apprentissage de grammaires stochastiques. Ces méthodes permettent en particulier d'envisager l'extraction de concepts structurés à partir de gros ensembles de données bruitées. C'est une des tâches du traitement de langue naturelle.

Compte tenu de la variété et de la complexité des problèmes posés, un bon nombre de méthodologies différentes d'apprentissage automatique ont déjà été testées pour le traitement de la langue naturelle. On peut citer entre autres :

- Les méthodes d'apprentissage à base de cas (voir [DvdBW97] pour une revue),
- Les méthodes bayésiennes (par exemple [NMTM98]),
- L'ILP (par exemple [MC95]),
- Les arbres de décision (par exemple [OW94]),
- L'inférence grammaticale algébrique et stochastique (cf. [ICG98])
- etc.

Divers colloques et congrès ont déjà été consacrés à l'apprentissage automatique appliqué au traitement de la langue naturelle. On pourra en particulier consulter les Actes de [ICM97], [AAA98].

La section qui suit introduit les techniques de correction d'erreurs utilisées par les algorithmes d'apprentissage presentés dans les sections 3 et 4. Dans ces mêmes sections nous proposerons des techniques d'evaluation des modèles de langage appris, appuyés par les experiences effectués sur les corpus artificiels et réels.

2 Modèles de correction d'erreurs

Nous rappelons ici les principales notions concernant les opérations de *correction d'erreurs* (ou opérations d'*édition*) entre séquences. Nous rappelons également leur application au calcul de la «distance» d'une séquence à une grammaire régulière, représentée comme un automate fini.

2.1 Définitions et notations

Pour déterminer les différences entre deux séquences A_1 et A_2, composées des lettres appartenant au même alphabet Σ, il est naturel de penser à modifier l'ordre et/ou la nature des lettres de A_1 afin d'obtenir A_2 (ou le contraire). On connaît un ensemble de telles opérations de *correction d'erreurs*, constitué de la *substitution*, de l'*insertion*, et de la *suppression* d'éléments de Σ, suffisant pour transformer toute chaîne A_1 en chaîne A_2. En désignant la chaîne vide par ϵ, on notera (a,b) la *substitution* de la lettre a par la lettre b, (ϵ,a) l'*insertion* de la lettre a, et (a,ϵ) la *suppression* de la lettre a.

2.2 Distance de Levenshtein

On appelle alors *distance de Levenshtein* le nombre minimal d'opérations (prises dans cet ensemble) nécessaires pour tranformer A_1 en A_2. On appelle *dérivation corrective optimale* la suite d'opérations d'éditions utilisées pour calculer la distance de Levenshtein :

$$D(A_1, A_2) = e_1, e_2, .., e_n \text{ avec } e_k = (x_i, x_j),\ 1 \leq k \leq n,\ \forall x_i, x_j \in \{\Sigma \cup \epsilon\}$$

Un algorithme de programmation dynamique [KS83] permet de calculer la dérivation correctionelle optimale en un temps de l'ordre de $\mathcal{O}(|A_1|.|A_2|)$, avec $|A_1|$ (resp. $|A_2|$) le nombre de lettres qui composent A_1 (resp. A_2).

On peut aussi attribuer à ces opérations des *coûts unitaires* de la façon suivante :

$$w(x_i, x_j) = \begin{cases} 1 \ si \ x_i \neq x_j \\ 0 \ si \ x_i = x_j \end{cases} \ \forall x_i, x_j \in \{\Sigma \cup \epsilon\}$$

Dans cette optique, la distance de Levenshtein est également le coût minimal de la transformation de A_1 en A_2 selon des opérations avec coûts unitaires.

2.3 Distance d'édition

On peut vouloir généraliser le calcul précédent en partant de la remarque suivante : il est parfois utile d'attribuer une plus grande importance à une opération d'édition qu'à une autre. Dans ce cas les $w(x_i, x_j)$ ne seront plus des coûts unitaires. Il faudra cependant leur imposer d'être positfs ou nuls pour pouvoir traiter le problème par programmation dynamique. Ces valeurs définissent alors une *matrice de coûts* sur $\Sigma \cup \{\epsilon\} \times \Sigma \cup \{\epsilon\} - \{\epsilon\} \times \{\epsilon\}$

On a un résultat classique [KS83] pour cette extension : en mesurant le coût de passage de A_1 à A_2 pour une certaine dérivation corrective comme la somme des coûts des opérations élémentaires employées, on sait encore calculer par programmation dynamique la *distance d'édition* (ou *distance de Levenshtein pondérée*) entre ces deux séquences : c'est le minimum sur tous les coûts de passage entre ces deux séquences.

De même, on sait calculer la dérivation corrective optimale selon ces opérations pondérées. Une propriété complémentaire est que, si la matrice des coûts $w(x_i, x_j)$ a les propriétés d'une distance, alors la distance d'édition est aussi une distance au sens rigoureux du terme [WF74]. Dans la suite, nous garderons le terme de «distance» même si la propriété n'est pas assurée.

2.4 Distance corrective entre une séquence et un automate

Une autre extension est la suivante : étant donnés un automate fini $\mathcal{A}$ sur un alphabet Σ, une séquence B et une matrice de coûts unitaires, notons Z la phrase du langage accepté par $\mathcal{A}$ dont la distance de Levenshtein à B est la plus faible.

Il est également possible de calculer la dérivation corrective entre B et Z par une extension de l'algorithme précédent [KS83]. Cette dérivation peut, par extension, être appelée *dérivation corrective optimale* entre un automate et une phrase.

Plus récemment, on a démontré [AV98] que cet algorithme peut également être adapté pour calculer la dérivation corrective optimale entre un automate fini et une phrase, même quand la matrice de coûts n'est pas unitaire. Ce résultat était connu dans le cas particulier des automates sans cycle [Wag74], mais son extension au cas général est un résultat à notre avis remarquable.

3 Apprentissage d'un modèle syntaxique symbolique

Dans la suite de cet article nous nous focalisons donc sur le domaine particulier de l'apprentissage de l'inférence des grammaires regulières à partir d'exemples positifs. Cette section introduit un algorithme d'apprentissage des grammaires non stochastiques qui utilise la dérivation corrective optimale, propose un calcul de pondération des opérations d'édition puis décrit un protocole d'évaluation de la qualité du modèle appris.

3.1 Algorithme d'apprentissage par correction d'erreurs : ECGIA

Proposée par Rulot et Vidal [RV88] [RPV89], cette méthode[1] construit de façon incrémentale une grammaire regulière $G = (N, \Sigma, R, S)$,[2] équivalente à un automate non déterministe et sans cycles. Le modèle initial est construit à partir de la première séquence presentée, puis il est amélioré par l'ajout de nouvelles règles résultant de l'analyse corrective optimale des séquences suivantes de l'ensemble d'apprentissage.

L'exemple qui suit illustre l'apprentissage d'un automate (assimilable à une grammaire régulière) à partir des exemples {aab, abb, ab, baab}. Notons que cet algorithme utilise une matrice de coûts unitaires. Une extension simple de cet algorithme, non proposée par leurs auteurs, est d'utiliser une matrice de coûts non unitaire. Cette extension du modèle est faite en garantissant qu'aucun cycle n'est introduit dans l'automate résultant. Cette propriété permet de calculer la dérivation corrective optimale par programmation dynamique classique comme nous l'avons vu au paragraphe 2.4.

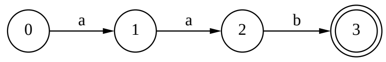

FIG. 1 – *Initialisation de l'automate avec 'aab'*

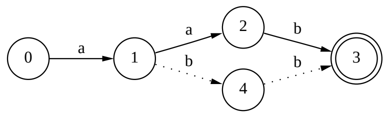

FIG. 2 – *"Parsing" avec 'abb' et substitution de a par b*

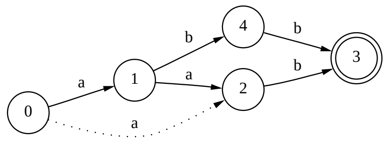

FIG. 3 – *"Parsing" avec 'ab' et suppression de a*

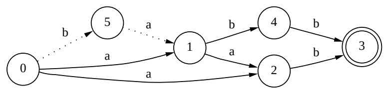

FIG. 4 – *"Parsing" avec 'baab' et insertion de b*

Sur cet exemple simple, on voit que la généralisation effectuée amène l'automate à accepter une phrase hors des exemples présentés : 'bbab'.

3.2 Evaluation du poids des opérations d'edition pendant l'apprentissage

Il est logique de vouloir étendre cet algorithme à l'inférence par dérivation corrective optimale avec une matrice de coûts variables afin de lier le coût d'édition à la fréquence de l'opération d'édition associée. Ceci est en particulier adapté à la langue naturelle où certaines opérations n'ont pas le même poids que les autres. Prenons par exemple la phrase :

«*euh, je voudrais obtenir le numéro des pompiers*»

1. ECGIA pour : Error Correcting Grammar Inference Algorithm
2. avec N : l'ensemble de non-terminaux, Σ : l'alphabet, R : l'ensemble de règles et S : non-terminal initial

Remplacer le mot *pompiers* par *urgences* est plus fréquent (autrement dit : «coûte moins») que par *hirondelles*. De la même façon il sera moins coûteux de supprimer le mot *euh* qu'un autre mot de cette phrase.

La matrice des coûts constitue alors un paramètre d'entrée de cet algorithme. Nous avons proposé dans [CM98] comment estimer les valeurs de cette matrice à partir du corpus d'apprentissage, en comptant les fréquences d'opérations d'édition utilisés lors de l'inférence de la grammaire. Nous utilisons pour cela une matrice M_{ECGI} de la forme suivante :

$M_{ECGI} =$

	ϵ	x_1	x_2	...	x_n
ϵ	0	$f(\epsilon, x_1)$	$f(\epsilon, x_2)$	...	$f(\epsilon, x_n)$
x_1	$f(x_1, \epsilon)$	$f(x_1, x_1)$	$f(x_1, x_2)$	...	$f(x_1, x_n)$
x_2	$f(x_2, \epsilon)$	$f(x_1, x_1)$	$f(x_1, x_2)$	...	$f(x_1, x_n)$
...	...	...	...	...	...
x_n	$f(x_n, \epsilon)$	$f(x_n, x_1)$	$f(x_n, x_2)$	...	$f(x_n, x_n)$

où $f(x_i, x_j)$ est le nombre de fois que l'opération (x_i, x_j) est utilisée pendant l'apprentissage et $n = |\Sigma|$. Remarquons que l'opération (ϵ, ϵ) n'est pas prise en compte.

Les *insertions* sont visibles sur la première ligne de la matrice, les *suppressions* sur la première colonne et les *substitutions* constituent les autres valeurs. En raisonnant en termes de coûts, il est logique de considérer qu'une opération qui arrive peu fréquemment doit être considérée comme plus coûteuse qu'une autre dont le nombre d'occurences est élevé. Cette remarque nous permet donc de transformer M_{ECGI} en matrice de coûts. La méthode que nous avons utilisée consiste à normaliser séparement les insertions, les suppressions et les substitutions selon les relations :

$w(\epsilon, x_i) = 1 - f(\epsilon, x_i)/max\{f(\epsilon, x_i)\}\ \forall x_i \neq \epsilon$
$w(x_i, \epsilon) = 1 - f(x_i, \epsilon)/max\{f(x_i, \epsilon)\}\ \forall x_i \neq \epsilon$
$w(x_i, x_j) = 1 - f(x_i, x_j)/max\{f(x_i, x_j)\},\ \forall x_i \neq \epsilon\ et\ \forall x_j \neq \epsilon$

3.3 Réapprentissage avec les nouveaux coûts

Nous pouvons alors réapprendre la grammaire G avec les nouvelles valeurs attribués aux opérations correctives, en utilisant cette fois l'extension de l'algorithme ECGIA proposée ci-dessus, puisque la matrice des coûts n'est plus unitaire.

Ce processus d'apprentissage, d'estimation de la matrice des coûts, puis de réapprentissage peut être réitéré. Nous avons constaté experimentalement [CM98] sur un corpus non trivial, issu du langage L_0 de Feldman [FLSW90] une convergence du processus au bout d'une dizaine d'itérations. Elle s'accompagne de la réduction en taille de la grammaire apprise (nombre de régles et de non-terminaux en décroissance).

La similitude de principe entre cette méthode et l'algorithme d'«Estimation-Maximisation» (EM) [DLR77] nous a amené à l'appeller «ECGI EM-like» bien que sa convergence n'a pas été démontrée à ce jour. Notons que la démonstration de convergence de l'algorihme «EM», par exemple pour les HMM (Chaînes de Markov cachées) s'appuye sur le fait que l'on réestime les probabilités sur une structure qui ne change pas lors des itérations, ce qui n'est pas le cas ici.

3.4 Evaluation de la qualité d'apprentissage

Il n'existe pas à notre connaissance de méthode générale d'évaluation de modèles de langages non-stochastiques en disposant uniquement des données positives. Il serait pourtant important de savoir évaluer dans l'absolu l'adéquation d'une grammaire aux données ou de pouvoir comparer deux méthodes d'apprentissage.

Pour cela, nous avons proposé dans [CM98] une méthode dont la caractéristique principale est la nécessité de supposer que l'on connaît le concept à apprendre, ou «grammaire cible». Elle est donc inutilisable dans le cas général. Elle reste cependant valide dans le cas de la comparaison de deux méthodes d'apprentissage où une grammaire artificielle peut être crée automatiquement ou proposée par un expert. Nous nous en servons donc dans le paragraphe suivant pour illustrer le gain apporté par l'algorithme «ECGI EM-like» par rapport au ECGIA classique.

3.5 Résultats

L'algorithme «ECGI EM-like» n'a pas pu etre appliqué sur les données réelles fournies par France Telecom (AGS : corpus des requêtes vocales, décrit dans : [SFC+96]), le lexique extrait de celui-ci étant trop important (env. 1000 entrées) par rapport au nombre des phrases fournies. Pour valider expérimentalement cette méthode nous avons eu recours à un corpus généré par la grammaire du langage L_0 (25 terminaux) mentionnée plus haut. Il a été divisé en 3 parties : corpus d'apprentissage de 1000 phrases, corpus de test non bruité de 1000 phrases et corpus de test de 1000 phrases avec 10% de bruit [3] introduit artificiellement. Les résultats de l'expérimentation sont resumées dans le tableau ci-dessous :

	#non-terminaux	#règles	corpus de test sans bruit	corpus de test avec 10% du bruit
$G(L_0)$	58	189	100%	59.0%
ECGI classique	572	1102	82.2%	51.3%
ECGI «EM-like»	534	1008	88.3%	53.1%

Sur les 531 phrases (53.1%) acceptées par la grammaire produite par ECGIA «EM-like» une a été rejetée par $G(L_0)$, contre deux sur les 513 phrases acceptées par la grammaire inferée par ECGIA classique. On voit donc que la nouvelle méthode apprend un modèle plus compact et plus proche du concept cible (meilleur taux d'acceptation sans sur-généralisation).

4 Apprentissage d'un modèle syntaxique stochastique

Dans cette section nous allons montrer comment le modèle vu précédemment peut être enrichi d'une distribution de probabilités ; puis nous présenterons une méthode fondée sur les techniques correctives permettant de l'évaluer. L'illustration de cette évaluation sera effectuée sur un corpus réel.

4.1 ECGIA stochastique

L'algorithme présenté dans la section 3 possède une extension stochastique, proposée par les mêmes auteurs [RV88].

Rappelons brièvement qu'un automate fini peut voir ses transitions affectées de valeurs réelles positives, sous la condition que la somme des valeurs sur les transitions partant d'un état soit égale à 1. L'automate est alors dit *probabilisé*, ou *stochastique*. On peut alors calculer pour toute phrase x la probabilité $p(x|G)$ d'avoir été engendrée par cet automate. On démontre que ce modèle est *consistant*, c'est à dire qu'il munit Σ^* d'une distribution de probabilités. En effet : $\sum_{x\in\Sigma^*} p(x|G) = 1 \quad (1)$.

Dans l'algorithme ECGIA stochastique, l'estimation de probabilités de chaque règle (ou transition) est faite pendant le processus d'apprentissage de la manière suivante :
Pour une règle quelconque $r \in R \;:\; A \rightarrow x_i B$:

- $p_t(r) = n_t(r)/m_t(A)$
- $n_t(r) = n_{t-1} + N_r(x) \quad \text{et} \quad n_{t_0}(r) = 1$
- $m_t(A) = m_{t-1}(A) + M_A(x) \quad \text{et} \quad m_{t_0'}(A) = 1 \quad \forall A \in N$

avec :
t : numéro d'ordre de la séquence x
t_0 , t_0' : numéros des étapes de création de r et de A
$N_r(x)$: nb. de fois (0 ou 1) que r a été utilisé pour reconnaître x
$M_A(x)$: nb. de fois (0 ou 1) que A a été utilisé pour reconnaître x
Nous obtenons alors une grammaire structurellement identique à celle apprise par ECGI non-stochastique, mais munie en plus de probabilités sur les règles, ce qui est équivalent à un automate stochastique.

3. Par «10% de bruit» nous entendons qu'en moyenne un mot sur 10 du corpus de test a été altereé : soit suprimé, soit remplacé par un autre, soit il a vu un nouveau mot tiré au hazard être introduit avant lui.

4.2 Evaluation de la qualité de l'apprentissage

Un moyen d'évaluation des modèles stochastiques : la perplexité

La *perplexité* est un des critères qui permettent d'évaluer la qualité d'un modèle syntaxique pourvu d'une distribution de probabilités [Dup96] [BJM83]. Elle peut être comprise comme le pouvoir de prédiction de ce modèle. Ce pouvoir est d'autant plus grand que le nombre de symboles proposés par le modèle pour correspondre au symbole suivant d'une séquence est petit. Si le modèle ne peut proposer aucun symbole (dans notre cas : si la probabilité d'une règle est nulle ou si elle n'existe pas) alors son pouvoir prédicitif est nul et sa perplexité est maximale. Formellement, la perplexité est définie par :

$$PP = 2^{\left[-\frac{1}{\|S\|}\sum_{i=1}^{|S|}\log_2 P(x_i)\right]}$$

avec :
$|S|$: le nombre de séquences de l'échantillon de test
$\|S\|$: la somme des longueurs des séquences de l'échantillon de test
$P(x_i)$: la probabilité que la i-ème séquence x_i ait été générée par le modèle
Ce calcul est conditionné d'une part par la consistance du modèle (ce dont nous sommes assurés), d'autre part par l'attribution d'une probabilité non nulle à toute séquence composée des symboles de Σ.

Dans le cas général, étant données une grammaire stochastique G et une séquence x, il est possible de calculer la probabilité pour que x soit générée par G. Cette probabilité peut valoir zéro, seulement si la syntaxe de la séquence n'est pas reconnue par la grammaire (la probabilité d'aucune règle de G n'étant égale à zéro).

Cependant, pour pallier le manque de données d'apprentissage, qui conduit à une mauvaise estimation statistique du modèle, on doit *lisser* ses paramètres. Ce lissage permet en outre que toute chaîne puisse être engendrée par le modèle avec une probabilité non nulle La technique de *lissage*, qui consiste ici à créer les transitions manquantes et à leur redistribuer une partie des valeurs des probabilités des règles existantes. Nous proposons une telle technique dans les paragraphes qui suivent.

Grammaire augmentée G'

Soit G une grammaire apprise par l'algorithme ECGI étendu au cas stochastique. Soit $G' = (N, \Sigma, R', S)$ la grammaire G augmentée par les *règles de correction d'erreurs* définies ainsi :

Insertion (de a) :	$A \to aA,\ \forall(A \to bB) \in P$
Substitution de b par a :	$A \to aB,\ \forall(A \to bB) \in P$
Suppression de b :	$A \to B,\ \ \forall(A \to bB) \in P$

Maintenant, la probabilité pour que $x \in \Sigma^*$ soit générée par G' est :

$$p_{G'}(x) = \sum_{\forall D'_G(x)} \Big(\prod_{\forall r_i \in D'_G(x)} p(r_i) \Big)$$

où $D'_G(x)$ est une dérivation correctionnelle (non-nécéssairement optimale) de G pour engendrer x.

La figure 5 présente une grammaire G (traits continus) à laquelle on a adjoint toutes les règles d'erreur (traits pointillés). Le mot vide ϵ (équivalent de la règle de suppression) est ici représenté par le caractère $\sim$. Remarquons qu'une séquence peut être acceptée de plusieurs façons.

Calculs des probabilités des règles de G'

On supposant que l'on connaît les probabilités des opérations correctionelles
$p(x_i, x_j)\ \forall x_i, x_j \in \{\Sigma \cup \epsilon\}$, nous pouvons écrire :

$$\forall\ A \to x_jB \in R' \quad p(A \to x_jB) = p(A \to x_iB \in R) * p(x_i, x_j) \quad (2)$$

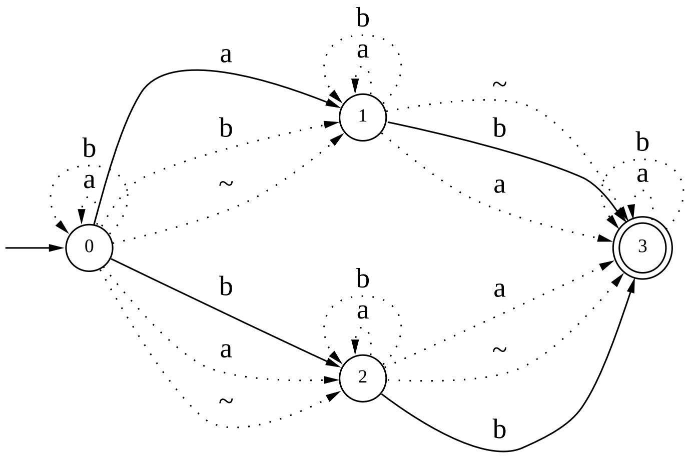

FIG. 5 – *Grammaire étendue G'*

Pour que le modèle reste consistant (i.e. pour qu'il vérifie (1)), il est nécéssaire que la somme des probabilités des règles dont la partie gauche est constituée d'un même non-terminal soit égale à 1 :

$$\sum_{\forall x \in \{\Sigma \cup \epsilon\}, \forall B \in N | \exists A \rightarrow x_i B \in R'} p(A \rightarrow x_i B) = 1 \quad \forall A \in N$$

Nous satisfaisons cette condition en ajoutant la contrainte :

$$\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} p(x_i, x_j) = 1, \forall x_i \in \{\Sigma \cup \epsilon\} \quad (3)$$

Le problème revient alors à calculer les probabilités des opérations correctionelles en respectant (3).

Estimation des probabilités des opérations corectionelles

Nous pouvons déduire les probabilités des opérations d'édition à partir de la matrice M_{ECGI} introduite dans 3.1 en la normalisant ligne par ligne (ce qui satisfait (3)). Deux problèmes se posent alors :

1. *Probabilités nulles des opérations* (x_i, x_j)

Nous obtenons les probabilités nulles pour les opérations d'édition qui n'ont pas été vues pendant l'apprentissage. Pour y remédier nous utilisons la technique de lissage d'*Absolute Discounting* [NE93] utilisées pour lisser des modèles de type N-grams que nous avons adaptées au lissage des probabilités d'édition de la manière suivante :

$$p(x_i, x_j) = \begin{cases} \frac{M_{ECGI}(x_i, x_j) - \lambda}{\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} M_{ECGI}(x_i, x_j)} & si\ M(x_i, x_j) > 0 \\ \frac{\lambda * (|\Sigma| - n_0)}{\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} M_{ECGI}(x_i, x_j)} * \Theta(x_i) & si\ M(\epsilon, x_j) = 0 \\ \frac{\lambda * (|\Sigma| - n_0 + 1)}{\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} M_{ECGI}(x_i, x_j)} * \Theta(x_i) & si\ M(x_i \neq \epsilon, x_j) = 0 \\ \Theta(x_i) & si \sum_{\forall x_j \in \Sigma} M_{ECGI}(x_i, x_j) = 0 \end{cases}$$

Avec :

n_0 **:** nombre de valeurs nulles sur une ligne de M_{ECGI}

$\Theta(x_i)$ **:** pourcentage du nombre d'opérations (x_j, x_i) $\forall x_i \neq x_j$ calculée pour les valeurs nulles de $M_{ECGI}(x_i, x_j)$ pour un x_i donn(on l'interprète comme l'importance des insertions des x_i à la place d'autres lettres).

La contrainte (3) reste satisfaite.

2. *Probabilités nulles des règles d'insertion*

Dans la grammaire apprise G, il n'y a pas de règles correspondant aux opérations d'insertion $(A \rightarrow x_i A)$. Par conséquence la rélation (2) leur attribue une probabilité nulle. La solution consiste à enlever une fraction β de probabilités à chacune des règles $A \rightarrow x_i B \in R'$ et l'attribuer à $p(A \rightarrow x_i A \in R')$ par :
$p(A \rightarrow x_i A \in R') = \beta * p(A \rightarrow x_i B \in R) * p(\epsilon, x_j)$ $\forall B \neq A$.

Chaque règle de $R^{'}$ se voit attribuer alors une probablité calculée selon :

$$\begin{array}{l} p(A \rightarrow x_i B \in R^{'}) = (1-\beta) * p(A \rightarrow x_i B \in R) * p(x_i, x_j) \\ p(A \rightarrow x_i A \in R^{'}) = \beta * p(A \rightarrow x_i B \in R) * p(\epsilon, x_j) \end{array} \quad \forall x_i \in \Sigma, \forall x_j \in \{\Sigma \cup \epsilon\}$$

La preuve de la consistance du modèle est developpée ci-dessous :

$\forall A, B \in N, A \neq B$:

$$\sum_{x_i \in \Sigma} p(A \rightarrow x_i B \in R^{'}) + p(A \rightarrow x_i A \in R^{'}) =$$

$$\sum_{x_i \in \Sigma} \sum_{x_j \in \{\Sigma \cup \epsilon\}} (1-\beta) * p(A \rightarrow x_i B \in R) * p(x_i, x_j) + \beta * p(A \rightarrow x_i B \in R) * p(\epsilon, x_j) =$$

$$\sum_{x_i \in \Sigma} \sum_{x_j \in \{\Sigma \cup \epsilon\}} p(A \rightarrow x_i B \in R) * p(x_i, x_j) =$$

$$\sum_{x_i \in \Sigma} p(A \rightarrow x_i B \in R) = 1$$

car G est un modèle consistant. En conséquence $G^{'}$ est aussi un modèle consistant.

Nous obtenons ainsi un système complet d'inférence et d'évaluation des grammaires stochastiques par utilisation des techniques correctives. Remarquons que cette méthode s'applique également avec l'algorithme «ECGI EM-like», puisque le calcul des $p(x_i, x_j)$ intervient uniquement après l'itération finale.

4.3 Résultats

Avec cette méthode le problème posé par le langage L_0 est devenu "trop facile" (la perplexité calculée sur le corpus de test est de l'ordre de 4). De plus l'importance du lissage est pratiquement négligeable (une grande majorité des phrases de test est acceptée par l'automate appris). Nous avons donc effectué des tests sur les données de France Telecom : le corpus AGS mentionné dans 3.5. Un bref resumé de ses caractéristiques se trouve dans le tableau ci-dessous :

Corpus original	Nombre d'exemples	Taille du lexique	Nombre de mots par phrase
Apprentissage	9850	866	5.04
Test	724	373	4.95

Sur la figure 6 nous avons representé la double variation des paramètres : chaque courbe correspond à une valeur de β, une courbe représente la perplexité en fonction de λ.

La valeur de perplexité la plus faible ($\approx$ 40) correspond au couple : $\beta = 0.05$, $\lambda = 0.6$. La valeur de β illustre la faible importance de l'opération d'insertion (cependant nécessaire car la perplexité augmente pour les valeurs $beta < 0.05$). La valeur obtenue pour λ est typique pour le lissage par *Absolute Discounting*.

Les données d'apprentissage et de test, dans la version originale fournie par le CNET, proviennent de locuteurs différents. Pour se placer dans des conditions d'apprentissage statistiquement plus satisfaisantes, nous avons mélangé ces deux corpus, puis tiré au hasard le même nombre de phrases pour l'apprentissage et pour le test. Les résultats sur cette «version 2» du corpus sont visualisés sur la figure 7.

Il est sans soute possible de comparer ces résultats avec ceux obtenus par un modèle de N-grams lissés avec "back-off" et "non-shadowing", modèle décrit par exemple dans [DR97] : au premier abord on obtient une perplexité d'environ 18 pour des modèles bigrammes et d'environ 14 pour des modèles trigrammes. Il faut cependant rester prudent car l'équivalence totale de l'ensemble des conditions expérimentales entre nos mesures et les leurs reste à vérifier. Rappelons que la "syntaxe" d'un modèle n-gramme est markovienne : la probabilité d'apparition d'un mot n'est conditionnée que par les $n-1$ mots précédents.

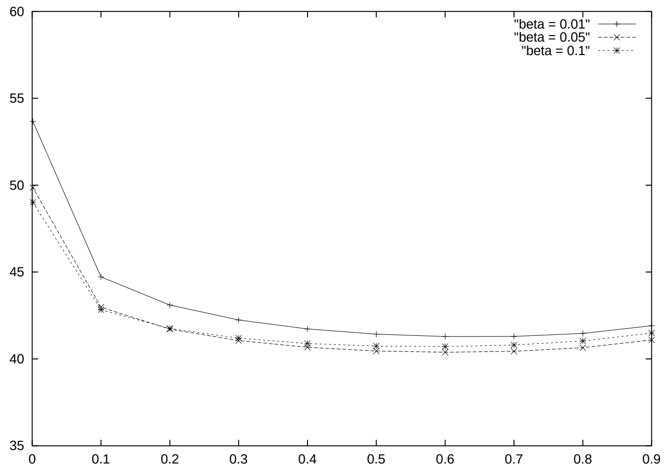

FIG. 6 – *Perplexité de la grammaire G' lissée par les techniques correctives sur le corpus AGS en version originale. La valeur de β apparaît dans l'etiquette des graphes, en haut à droite de la ligne*

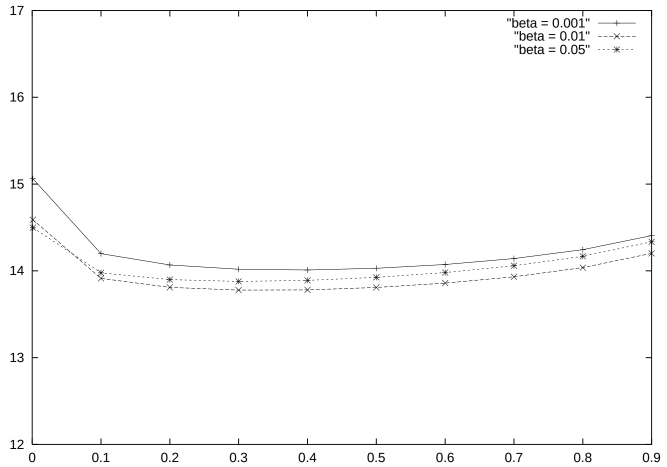

FIG. 7 – *Perplexité de la grammaire G' lissée sur le corpus AGS en version 2*

Les "règles de grammaires" sous-jacentes ne révèlent donc pas de structure syntaxique au sens habituel, contrairement à ce que l'on espère d'un automate appris par inférence grammaticale.

5 Conclusion

Nous avons proposé dans cet article un système homogène d'apprentissage et d'évaluation de modèles de langages à partir d'un corpus. La notion centrale utilisée est celle de *distance corrective* entre deux phrases ou entre une phrase et un automate fini. De cette notion dérive l'apprentissage à partir d'exemples positifs d'une grammaire régulière, non stochastique ou stochastique. Nous avons montré comment l'apprentissage itératif de la grammaire et des poids caractéristiques de la distance corrective permettent d'améliorer l'algorithme classique d'inférence de la grammaire non stochastique. Nous avons proposé ensuite une méthode d'apprentissage d'un automate stochastique, toujours basée sur l'utilisation de la distance corrective. Ceci permet, après lissage, de quantifier la qualité de la généralisation obtenue en mesurant la perplexité de la grammaire sur l'ensemble de test. Le lissage d'automates stochastiques, en particulier obtenus par inférence grammaticale, est un problème nouveau dont l'importance a en particulier été montrée dans [TH98]. La nouvelle méthode proposée ici possède la caractéristique d'être homogène avec la méthode d'apprentissage. Les résultats obtenus sur le corpus AGS, composé de phrases réelles prononcées en situation de dialogue oral homme-machine, montrent que cette modélisation syntaxique est pertinente. Comme elle est destinée à fournir l'étage grammatical du système de reconnaissance vocale qui se trouve en entrée du système de dialogue, il est intéressant de comparer ses résultats avec le modèle syntaxique courant, un modèle de bigrammes.

Ce travail peut être poursuivi dans plusieurs directions :

- L'évaluation d'un modèle syntaxique non stochastique nécessite un modèle du langage cible, ce qui est une contrainte difficile à admettre. Une méthode d'évaluation de la qualité de généralisation libérée de cette contrainte est en cours de constitution.
- Il sera intéressant d'introduire des connaissances linguistiques dans ce travail, en particulier en travaillant sur des catégories grammaticales regroupant les mots. Le corpus AGS devrait alors avoir un comportement meilleur vis à vis de l'algorithme non stochastique itératif (EM-like).
- L'intégration au système de dialogue du CNET est le but final : la mesure de perplexité ne suffit pas à valider complètement une grammaire.

En conclusion, il nous semble que traiter par un concept commun (la correction d'erreur) l'apprentissage de l'automate, la réestimation des poids, l'affectation de probabilité aus règles apprises, puis le lissage de l'automate appris est non seulement satisfaisant d'un point de vue de principe, mais donne des résultats intéressants sur un corpus réel de langue naturelle.[4]

Références

[AAA98] *Applying Machine Learning to Discourse Processing.* – 1998. AAAI Press, Technical Report SS-98-01.

[AV98] Amengual (J.-C.) et Vidal (E.). – Efficient error-correcting viterbi parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-20, n° 10, October 1998.

[BJM83] Bahl (L. R.), Jelinek (F.) et Mercer (R. L.). – A maximum likelihood approach to continious speech recognition. *IEEE trans. on Pattern Analysis and Machine Intelligence*, vol. 5, 1983, pp. 179–190.

[CM98] Chodorowski (J.) et Miclet (L.). – Applying grammar inference in learning a language model for oral dialogue. *In : International Colloquium on Grammatical Inference*, pp. 102–113. – 1998. Springer-Verlag.

[DLR77] Dempster (A.P.), Laird (N.M.) et Rubin (D.B.). – Maximum likehood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, vol. 39, n° ser. B, 1977, pp. 1–38.

[DR97] Dupont (P.) et Rosenfeld (R.). – *Lattice based language models.* – Rapport technique n° CMU-CS-97-173, Carnegie Mellon University, 1997.

[Dup96] Dupont (P.). – *Utilisation et apprentissage de modèles de langages pour la reconnaissance de la parole continue.* – Thèse de PhD, ENST, 1996.

[DvdBW97] Daelemans (W.), van den Bosch (A.) et Weijters (T.). – Empirical learning of natural language processing tasks. *In : Proceedings of 9th European Conference on Machine Learning, Workshop on* Empirical Learning of Natural Language Processing Tasks. – 1997.

[FLSW90] Feldman (J. A.), Lakoff (G.), Stolcke (A.) et Weber (S. Hollbach). – *Miniature Language Acquisition : A touchstone for cognitive science.* – Rapport technique, ICSI, 1990. TR-90-009.

[ICG98] *Grammatical Inference.* – 1998. Lecture Notes in Artificial Intelligence 1433, Springer-Verlag.

[ICM97] *Empirical Learning of Natural Language Processing Tasks.* – 1997.

4. Outre les aspects liés à l'inférence proprement dite, nous avons présenté le lissage d'automates produits par l'algorithme ECGI qui ont la particularité d'être sans cycles. Un travail simultané et indépendant aborde une généralisation de ce problème de lissage au cas où les automates comportent des cycles (P. Dupont, J.-C. Amengual : Smoothing probabilistic automata: an error-correcting approach, Rapport technique EURISE 9806, Université Jean Monnet, Saint-Etienne, 1998)

[KS83] Kruskal (J. B.) et Sankoff (D.). – *Time Warps, String Edits, and Macromolecules : the Theory and Practice of Sequence Comparaison.* – Addison-Wesley, 1983.

[MC95] Mooney (R. J.) et Califf (M. E.). – Induction of first-order decision lists : Results on learning the past tense of english verbs. *J.A.I.R.*, vol. 3, 1995, pp. 1–24.

[NE93] Ney (H.) et Essen (U.). – Estimating small probabilities by leaving-one-out. *In : European Conference on Speech Communication and Technology (Eurospeech'93)*, pp. 2239–2242. – Berlin, Germany, 1993.

[NMTM98] Nigam (K.), McCallum (A.), Thrun (S.) et Mitchell (T.). – Learning to classify text from labeled and unlabeled documents. *5th National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[OW94] Ostendorf (M.) et Wightman (C. W.). – Automatic labelling of prosodic patterns. *IEEE Transactions on Speech and Audio Processing*, vol. 2, n° 4, 1994, pp. 469–481.

[RPV89] Rulot (H.), Prieto (N.) et Vidal (E.). – Learning accurate finite-state structural models of words: the ecgi algorithm. *In : ICASSP'89*, pp. 643–646. – 1989.

[RV88] Rulot (H.) et Vidal (E.). – An efficient algorithm for the inference of circuit-free automata. *Syntactic and Structural Pattern Recognition*, 1988, pp. 173–184.

[SFC+96] Sadek (D.), Ferrieux (A.), Cozannet (A.), Bretier (P.), Panaget (F.) et Simonin (J.). – Effective human-computer cooperative spoken dialogue : the ags demonstrator. *In : Proceedings of ICSLP*, pp. 542–545. – 1996.

[TH98] Thollard (F.) et Higuera (de la) (C.). – The importance of smoothing in learning deterministic stochastic automatons, 1998.

[Wag74] Wagner (Robert A.). – Order-n correction for regular languages. *Communication of the ACM*, vol. 17, n° 5, May 1974.

[WF74] Wagner (Robert A.) et Ficher (Michael J.). – The string-to-string correction problem. *Journal of the Association for Computing Machinery*, vol. 21, n° 1, January 1974, pp. 168–173.