Champs Conditionnels Aléatoires pour l'Annotation d'Arbres

Florent Jousse, Rémi Gilleron, Isabelle Tellier, Marc Tommasi

Champs conditionnels aléatoires pour l'annotation d'arbres

Florent Jousse, Rémi Gilleron, Isabelle Tellier, Marc Tommasi

Universités de Lille
LIFL - équipe GRAppA & INRIA Futurs - Projet MOSTRARE
jousse@grappa.univ-lille3.fr
{remi.gilleron, isabelle.tellier, marc.tommasi}@univ-lille3.fr

Résumé : Avec en vue la transformation de documents semi-structurés de type XML, nous nous intéressons au problème de l'annotation de tels documents par apprentissage statistique, à partir d'exemples de documents déjà annotés. Afin de modéliser la probabilité d'une annotation connaissant un document, nous nous plaçons dans le cadre des champs conditionnels aléatoires. Ce modèle a déjà fait ses preuves pour l'annotation de séquences : nous l'adaptons ici aux arbres ordonnés d'arité non bornée. Nous étudions l'expressivité du nouveau modèle ainsi introduit en le comparant aux automates d'arbres stochastiques (ou grammaires régulières probabilistes d'arbres). Nous présentons aussi en détail l'algorithme de recherche de l'annotation la plus probable et l'algorithme d'inférence pour ce modèle. Ces algorithmes sont implantés dans une librairie Tree CRF écrite en JAVA. Ces travaux sont des préliminaires qui nous permettront par la suite d'étudier les applications du modèle pour la transformation de documents.

Mots-clés : Données semi-structurées, Annotation, Modèles conditionnels, Champs conditionnels aléatoires

1 Introduction

L'explosion des sources d'information numériques rend pressant le besoin d'intégrer des données sémantiquement riches, hétérogènes et réparties. Cette intégration repose principalement sur des technologies XML et des standards tels que XPATH, XSLT, XQUERY. Cependant, l'écriture de requêtes et de transformations reste du domaine du spécialiste alors que les besoins se multiplient. Automatiser la génération de transformations rendrait de grands services. C'est l'objectif à long terme que nous visons, en le considérant comme un problème d'apprentissage à partir d'exemples formés de couples (document origine, document transformé). Dans un premier temps, nous nous limitons aux transformations qui préservent la structure des arbres. Cette limitation n'est pas si stricte qu'on pourrait le penser, car la sémantique de l'alphabet d'annotation peut permettre d'exprimer des transformations plus riches. Par exemple, on peut annoter un arbre par des opérations de suppression ou d'ajout d'un nœud qui, une fois exécutées, modi-

fieront sa structure. L'annotation peut ainsi être vue comme un pré-traitement pour la transformation.

Le traitement de données structurées par des algorithmes d'apprentissage en est encore à ses débuts. Les efforts se sont surtout concentrés jusqu'à présent sur l'annotation de séquences : ainsi, de nombreux travaux ont porté sur les modèles de Markov cachés (HMMs, (Rabiner & Juang, 1986)) et les grammaires régulières stochastiques (Carrasco & Oncina, 1994). Dans ces modèles dit "génératifs", on modélise à la fois la séquence à annoter et l'annotation. On est donc amené à résoudre un problème plus difficile que le problème original : apprendre une probabilité jointe plutôt que d'apprendre la probabilité de l'annotation sachant la séquence d'entrée. Pour répondre à cette critique, des modèles alternatifs ont été proposés. Le problème de l'annotation a ainsi été considéré comme un problème de classification supervisée à sortie structurée (Taskar *et al.*, 2005; Tsochantaridis *et al.*, 2004). Le problème de l'annotation peut également être considéré comme un modèle à maximum d'entropie. C'est dans ce cadre qu'ont été introduits les champs conditionnels aléatoires (Lafferty *et al.*, 2001) qui ont principalement été utilisés pour des séquences (Sha & Pereira, 2003; McCallum & Li, 2003; Pinto *et al.*, 2003).

Certains travaux récents ont pourtant commencé à aborder l'apprentissage de structures arborescentes, via des automates d'arbres stochastiques (Habrard *et al.*, 2005) ou des modèles de Markov arborescents. Nous pouvons également citer les travaux portant sur les réseaux bayésiens dynamiques (Denoyer & Gallinari, 2004) pour les documents semi-structurés : cette approche présente l'avantage de pouvoir prendre également en compte le contenu textuel des documents semi-structurés. Mais ces modèles restent génératifs. Or, il ne nous semble pas réaliste et pertinent d'espérer modéliser ainsi des données semi-structurées dans un contexte de masses de données hétérogènes. Nous avons donc choisi de nous orienter vers des modèles non génératifs.

Dans cet article, nous proposons donc d'adapter le modèle des champs conditionnels aléatoires à l'annotation d'arbres, en prenant en compte la structure de ces arbres. Les structures des documents semi-structurés, abstraction faite des attributs et des références, sont en effet des arbres ordonnés d'arité non bornée. Nous proposons un modèle, appelé Tree CRF, de champs conditionnels aléatoires adapté à ces structures. En particulier, la structure choisie pour le graphe d'indépendances permet d'exprimer des propriétés provenant de relations verticales (entre père et fils) mais aussi horizontales (entre frères d'un même père). Pour rendre ce modèle opérationnel, nous détaillons les algorithmes permettant la recherche de l'annotation la plus probable et l'inférence des paramètres d'un Tree CRF. Ces algorithmes exploitent des techniques issues de la programmation dynamique et des méthodes de descente gradient adaptées, afin de les rendre aussi efficaces que possible. La complexité des calculs est en effet connue comme problématique dans les champs conditionnels aléatoires. Les algorithmes proposés sont implantés dans une librairie JAVA et les premières expériences réalisées nous laissent espérer des temps de calcul raisonnables, c'est-à-dire, de l'ordre de grandeur de ceux obtenus sur les séquences. Notons enfin qu'un premier modèle de champs conditionnels aléatoires pour les arbres a été proposé par (Cohn & Blunsom, 2005). Mais ce modèle est restreint aux arbres ordonnés d'arité bornée, et seules les propriétés verticales (issues de relations père-fils) peuvent y être exprimées.

La suite de l'article est organisée comme suit : en partie 2, nous exposons le modèle des CRFs en général et son utilisation pour l'annotation de séquences en particulier. En partie 3, nous présentons notre adaptation des CRFs pour les arbres. Pour illustrer son expressivité, nous montrons qu'il permet de définir les mêmes probabilités conditionnelles qu'un automate d'arbres stochastique. Enfin, la partie 4 est consacrée à l'adaptation des algorithmes classiques d'apprentissage dans les CRFs au cas de notre modèle.

2 Champs conditionnels aléatoires

2.1 Définition des CRFs

Les champs conditionnels aléatoires (Conditional Random Fields ou CRFs) (Lafferty *et al.*, 2001) sont des modèles graphiques non dirigés ayant pour objectif de définir une distribution de probabilités sur les annotations y étant donnée une observation x. Ils sont définis comme suit.

Soit $\mathcal{G} = (V, E)$ un graphe non dirigé (appelé graphe d'indépendances) où V est l'ensemble des nœuds (vertices) et E l'ensemble des arcs (edges), et X et Y deux champs aléatoires décrivant respectivement l'observation et son annotation, de sorte que pour chaque nœud $v \in V$ il existe une variable aléatoire Y_v dans Y. On dit que (X, Y) est un champ conditionnel aléatoire si chaque variable aléatoire Y_v respecte la propriété de Markov suivante :

$$\forall v, p(Y_v|X, \{Y_w, w \neq v\}) = p(Y_v|X, \{Y_w, (V, W) \in E\})$$

c'est-à-dire chaque variable aléatoire Y_v dépend uniquement de X et de ses voisins dans le graphe d'indépendances.

D'après le théorème de Hammersley-Clifford (Hammersley & Clifford, 1971), cette condition d'indépendance permet d'écrire la probabilité d'une annotation y étant donnée une observation x comme un produit de fonctions de potentiel $\psi_c(y_c, x)$ sur tous les sous-graphes complètement connectés, appelés cliques, du graphe d'indépendances.

$$p(y|x) = \frac{1}{Z(x)} \prod_{c \in \mathcal{C}} \psi_c(y_c, x)$$

où $\mathcal{C}$ est l'ensemble des cliques de $\mathcal{G}$, y_c est la configuration prise par les variables aléatoires de la clique c dans l'observation y et $Z(x)$ est un coefficient de normalisation défini comme suit :

$$Z(x) = \sum_{y} \prod_{c \in \mathcal{C}} \psi_c(y_c, x)$$

Pour les CRFs, Lafferty, McCallum et Pereira (Lafferty *et al.*, 2001) ont proposé de définir la forme de ces fonctions de potentiel comme l'exponentielle d'une somme pondérée de fonctions f_k appelées *features*, les λ_k étant les poids associés à chacune de ces features :

$$\psi_c(y_c, x) = \exp\Big(\sum_k \lambda_k f_k(y_c, x, c)\Big)$$

Les features sont des fonctions à valeurs réelles. C'est à travers elles que toutes les connaissances du domaine sont intégrées dans le modèle. Dans la plupart des cas, les features sont des fonctions binaires valant 1 si un phénomène est observé, 0 sinon. Ces features prennent en paramètres les valeurs prises par les variables aléatoires de la clique sur laquelle elles s'appliquent (y_c), ainsi que l'ensemble de l'observation x. Par conséquent, la valeur prise par une variable aléatoire peut dépendre de toute l'observation x. Par exemple, dans le cas de l'annotation d'une séquence, le choix de l'étiquette associée au dernier élément de la séquence peut être lié à la *valeur* du premier élément de cette séquence.

À ces features sont associés des poids $\Lambda = \{\lambda_k\}$. Ces poids sont les paramètres du modèle. Ils permettent d'attacher plus ou moins d'importance à certaines features, ou même d'indiquer que le phénomène caractérisé par une feature ne doit pas se produire (si le poids est négatif).

Un CRF est donc défini par un graphe d'indépendances $\mathcal{G}$ et un ensemble de *features* f_k, auxquelles sont associés des poids λ_k. La probabilité conditionnelle d'une annotation connaissant une observation, telle que définie par un CRF, s'exprime alors par :

$$p(y|x) = \frac{1}{Z(X=x)} \exp\Big(\sum_{c\in\mathcal{C}} \sum_{k} \lambda_k f_k(y_c, x, c)\Big) \tag{2.1}$$

où $Z(x)$ se réécrit :

$$Z(x) = \sum_{y} \exp\Big(\sum_{c\in\mathcal{C}} \sum_{k} \lambda_k f_k(y_c, x, c)\Big) \tag{2.2}$$

Le premier problème associé aux CRFs est le *problème de l'annotation* qui consiste à rechercher l'annotation la plus probable selon (2.1) associée à une observation. Le second problème est celui de *l'inférence ou de l'apprentissage* du CRF, qui consiste à estimer les paramètres $\Lambda = \{\lambda_k\}$ qui maximisent la vraisemblance du modèle par rapport à un échantillon d'observations annotées. Ces paramètres peuvent être appris en utilisant une méthode classique de maximisation de la log-vraisemblance. Les paramètres optimaux ne pouvant pas être calculés de façon analytique, des méthodes de descente de gradient sont utilisées. H. Wallach (Wallach, 2002) a montré que la méthode la plus performante dans ce contexte est l'algorithme BFGS à mémoire limitée (L-BFGS) (Byrd *et al.*, 1995).

2.2 Cas des CRFs sur les séquences

Dans la littérature, les CRFs ont pour l'instant été utilisés essentiellement dans le cas de l'annotation de séquences.Ils ont ainsi servi dans des tâches aussi variées que l'annotation syntaxique (Part-Of-Speech tagging), le Shallow Parsing (Sha & Pereira, 2003), la reconnaissance d'entités nommées (McCallum & Li, 2003), l'extraction d'informations (Pinto *et al.*, 2003), parmi d'autres. Dans ces travaux sur les séquences, le graphe d'indépendances utilisé est une "first-order linear chain" (Fig. 1).

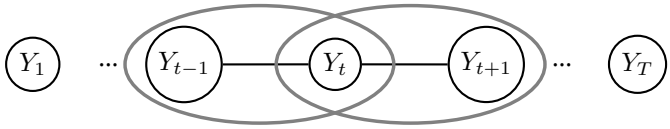

FIG. 1 – First-order linear chain model

Dans ce type de modèle, la probabilité d'une annotation s'exprime comme suit :

$$p(y|x) = \frac{1}{Z(x)} \exp\Big(\sum_{t=2}^{T} \sum_{k} \lambda_k f_k(y_{t-1}, y_t, x, t)\Big)$$

où T est la longueur de la séquence x. Les features sont de la forme $f_k(y_{t-1}, y_t, x, t)$ car les cliques du graphe d'indépendances sont les paires de nœuds (Y_{t-1}, Y_t). On note ici que, par souci de simplification, on ne considère que les cliques à deux noeuds dans l'écriture des formules et algorithmes. En effet, les cliques à un noeud peuvent être traitées de façon similaire.

L'intérêt principal de travailler avec un graphe d'indépendances aussi simple que celui-ci est qu'il permet de mettre en oeuvre des techniques de programmation dynamique pour un calcul efficace des deux tâches principales des CRFs que sont la recherche de l'annotation la plus probable et l'apprentissage des paramètres du modèle. Nous donnons ici un aperçu de ces deux algorithmes.

La recherche de l'annotation la plus probable consiste à trouver l'annotation $\hat{y}$ maximisant la probabilité $p(\hat{y}|x)$, étant donnée une observation x et un CRF dont les paramètres λ_k sont connus.

$$\hat{y} = \arg\max_y p(y|x) = \arg\max_y \sum_{t=2}^{T} \sum_{k} \lambda_k f_k(y_{t-1}, y_t, x, t)$$

Il est évidemment impossible de calculer naïvement cette valeur pour toutes les annotations y possibles. Toutefois, la forme du graphe permet de mettre en place l'algorithme de Viterbi d'une façon similaire à son utilisation dans le cas des HMMs. Pour cela, on définit le coefficient $\delta_t(y_t)$ comme étant le "score", *ie.* la somme pondérée des features sur toute la séquence, de la meilleure annotation de $x_1 \dots x_t$ où x_t est annoté par y_t. Sa formule de récurrence est définie comme suit :

$$\delta_{t+1}(y_{t+1}) = \max_{y_t} \delta_t(y_t) \exp\Big(\sum_{k} \lambda_k f_k(y_t, y_{t+1}, x, t+1)\Big)$$

Cette variable permet d'obtenir aisément le score de la meilleure annotation

$$\max_y \sum_{t=2}^{T} \sum_{k} \lambda_k f_k(y_{t-1}, y_t, x, t) = \max_{y_T} \delta_T(y_T)$$

En mémorisant le chemin de Viterbi correspondant, on obtient la meilleure annotation $\hat{y}$ de l'observation x.

La tâche d'apprentissage d'un CRF consiste, étant donné un ensemble d'apprentissage $S = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$, à trouver les poids Λ qui maximisent la log-vraisemblance du modèle

$$\mathcal{L}_\Lambda = \sum_{i=1}^{N} \log\left(p_\Lambda(y^{(i)}|x^{(i)})\right) \tag{2.3}$$

où $p_\Lambda(y|x)$ est la probabilité de l'annotation y sachant l'observation x dans le CRF dont les paramètres sont Λ.

L'algorithme de maximisation de la log-vraisemblance utilisé étant une descente de gradient (L-BFGS), $\mathcal{L}_\Lambda$ est calculé à chaque étape avec des paramètres Λ différents. Il est donc nécessaire de calculer cette fonction efficacement. La partie coûteuse de ce calcul est le coefficient de normalisation $Z(x)$ (*cf.* eq. (2.2)) qui intervient dans la probabilité $p(y|x)$. En effet, celui-ci est la somme des probabilités non normalisées ($p(y|x) \times Z(x)$) pour toutes les annotations possibles de x. Une méthode de programmation dynamique similaire à l'algorithme *forward* des HMMs est donc employée. Pour cela, on définit les coefficients *forward* $\alpha_t(y_t)$. $\alpha_t(y_t)$ est la probabilité non normalisée de toutes les annotations possibles de la séquence $x_1 \ldots x_t$ où x_t est annoté par y_t. La formule de récurrence de ce coefficient est :

$$\alpha_{t+1}(y_{t+1}) = \sum_{y_t} \alpha_t(y_t) \exp\Big(\sum_k \lambda_k f_k(y_t, y_{t+1}, x, t+1)\Big)$$

En utilisant ce coefficient, on peut calculer $Z(x)$ de la façon suivante :

$$Z(x) = \sum_{y_T} \alpha_T(y_T)$$

3 Tree CRFs

Nous proposons maintenant d'adapter le modèle des CRFs au cas de l'annotation d'arbres ordonnés d'arité non bornée. Ces arbres permettent en effet de modéliser les données semi-structurées si on ne prend pas en compte les attributs et les références. Dans de tels arbres, tout nœud peut être repéré par une position qui est une séquence d'entiers et à tout nœud est associé un label. Par ailleurs, les fils d'un nœud sont ordonnés et un nœud ayant un label peut avoir un nombre quelconque de fils. Par exemple, $u(l, l, l, u(l, l))$ est un arbre d'arité non bornée. L'ensemble des positions des nœuds est l'ensemble $\{\epsilon, 1, 2, 3, 4, 41, 42\}$ où ϵ désigne la racine. La racine a pour label u et a quatre fils, le nœud 4 a pour label u et a deux fils. Dans la suite, étant donné un arbre y, nous noterons n une position, $n.i$ son i-ème fils et y_n le label du nœud n.

3.1 Caractéristiques du modèle

Nous devons, dans un premier temps, proposer un graphe d'indépendances adapté aux arbres. Comme nous considérons des arbres ordonnés d'arité non bornée (de type XML), ce modèle doit permettre l'expression de récurrences verticales (en profondeur)

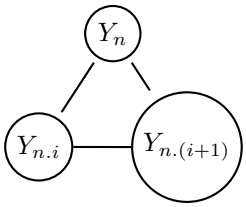

FIG. 2 – Modèle d'indépendances fils-frère.

et horizontales (en largeur). Dans cet objectif, nous proposons un graphe où une clique maximale, représentée dans la figure 2, est composée de trois nœuds : un père et deux de ses fils consécutifs. Par exemple, les cliques maximales pour $u(l,l,l,u(l,l))$ sont définies par les triplets de positions : $(\epsilon,1,2)$, $(\epsilon,2,3)$, $(\epsilon,3,4)$, $(\epsilon,41,42)$. Dans le calcul de la probabilité conditionnelle d'un arbre d'annotation sachant un arbre d'entrée, toutes les cliques maximales à trois noeuds qui respectent le schéma "père - deux fils consécutifs" seront considérées. Par souci de simplification, nous ne considérerons dans la suite que les features s'appliquant sur les cliques maximales à tois noeuds dans les formules et l'expression des algorithmes. La prise en considération des features sur les cliques à un noeud et à deux noeuds dans les formules et algorithmes est aisée.

Du graphe d'indépendances que nous venons de définir, nous déduisons que les features d'un Tree CRF sont de la forme $f_k\big(y_n, y_{n.i}, y_{n.(i+1)}, x, (n, n.i, n.(i+1))\big)$. Elles prennent en paramètre un triplet de positions (celle du père n et celles de ses deux fils consécutifs $n.i$ et $n.(i+1)$), les étiquettes dans l'annotation y de ce triplet de positions et l'arbre observé x dans sa totalité. Nous simplifierons l'écriture d'une feature en oubliant le triplet de positions. Une feature sera donc écrite sous la forme $f_k\big(y_n, y_{n.i}, y_{n.(i+1)}, x\big)$. On peut noter qu'une clique maximale est repérée par la donnée des positions n et $n.i$. À ces features sont associés des poids que l'on notera λ_k. Dans la suite de cet article, nous utiliserons par commodité les notations suivantes :

$$\Lambda F_{n,n.i} = \sum_k \lambda_k f_k(y_n, y_{n.i}, y_{n.(i+1)}, x))$$

où Λ est le vecteur de poids et $F_{n,n.i}$ le vecteur de features. Pour simplifier les formules, on dira que $\Lambda F_{n,n.i} = 0$ si $n.i$ est le dernier fils de n (*ie.* il n'existe donc pas de nœud $n.(i+1)$).

Maintenant que nous avons défini le graphe d'indépendances de notre modèle ainsi que les features utilisées, nous pouvons écrire la formule du calcul de la probabilité conditionnelle de l'arbre annotation y sachant x :

$$p(y|x) = \frac{1}{Z(x)} \prod_{c \in \mathcal{C}} \exp \Lambda F_{n,n.i} \tag{3.1}$$

où $\mathcal{C}$ est l'ensemble des cliques de $\mathcal{G}$.

Le coefficient de normalisation $Z(x)$ est donc :

$$Z(x) = \sum_{y} \Big(\prod_{c \in \mathcal{C}} \exp \Lambda F_{n,n.i} \Big)$$

La section 4 décrit précisément les adaptations algorithmiques nécessaires pour les tâches d'annotation et d'apprentissage. Mais dans un premier temps, nous allons comparer l'expressivité de nos Tree CRFs à celle des automates d'arbres stochastiques.

3.2 Comparaison des Tree CRFs et des Automates d'Arbres Stochastiques

Dans l'introduction des CRFs sur les séquences, les auteurs (Lafferty *et al.,* 2001) montraient qu'il est possible de représenter un HMM sous la forme d'un CRF. De la même manière, nous allons montrer ici comment représenter un automate d'arbres stochastique à l'aide d'un Tree CRF afin de donner une idée de l'expressivité du modèle que nous avons défini.

Comme il n'existe pas de définition consensuelle des automates d'arbres stochastiques dans le cas des arbres d'arité non bornée, nous nous restreignons ici au cas des arbres d'arité bornée. Pour simplifier les notations, nous considérons des automates sur les arbres d'arité 0 ou 2. C'est, bien sûr, sur le problème de l'annotation que nous souhaitons comparer les automates stochastiques et les Tree CRFs. Une annotation d'un arbre x par un automate stochastique est un arbre y de même structure sur l'alphabet des états qui correspond à un calcul de l'automate d'arbre stochastique. La recherche de l'annotation la plus probable n'a de sens que s'il existe plusieurs calculs de l'automate. Les automates envisagés ici sont donc stochastiques et non déterministes. On peut noter également que nous utilisons des automates d'arbres ascendants. En renversant les règles, on peut les considérer sous un point de vue génératif, ce qui correspond à des grammaires régulières stochastiques d'arbres.

Un automate A d'arbres stochastique est un quintuplet $(\Sigma, Q, \Delta, p, \mathit{final})$ où $\Sigma = \Sigma_0 \cup \Sigma_2$ est un ensemble fini de symboles de fonction d'arité 0 ou 2, Q est un ensemble fini d'états, Δ est un ensemble de règles de la forme : $a \rightarrow q$ où $a \in \Sigma_0$ et $q \in Q$ ou $g(q_1, q_2) \rightarrow q$ où $g \in \Sigma_2$ et $q, q_1, q_2 \in Q$, $p : \Delta \rightarrow [0, 1]$ est la fonction qui associe à toute règle un réel dans $[0, 1]$, et la fonction $\mathit{final} : Q \rightarrow [0, 1]$ associe à chaque état sa probabilité d'être final. On dit qu'un automate d'arbres stochastique est normalisé si, pour tout $q \in Q$, la somme des probabilités des transitions arrivant sur q vaut 1, et si la somme des probabilités des états finaux vaut 1.

Le calcul de la probabilité conditionnelle d'une annotation y sachant une observation x dans un automate d'arbre stochastique A est :

$$p_A(y|x) = \frac{p_A(y, x)}{\sum_y p_A(y, x)}$$

où $p_A(y, x)$ est la probabilité du calcul réussi y sur l'arbre x obtenue en multipliant les poids de toutes les règles utilisées dans le calcul y et la probabilité de l'état en racine de y d'être terminal.

Nous allons maintenant représenter un automate d'arbres stochastique avec un Tree CRF. Premièrement, l'alphabet $\mathcal{Y}$ des labels du Tree CRF correspond à l'ensemble d'états Q de l'automate. Chaque transition de l'automate est représentée par une feature dans le Tree CRF. Pour chaque règle de la forme $a \to q$, on crée une feature à un noeud $f_{a \to q}$ de la forme :

$$f_{a \to q}(y_n, x) = \begin{cases} 1 & \text{si } x_n = a \text{ et } y_n = q \\ 0 & \text{sinon} \end{cases}$$

Pour chaque règle de la forme $g(q_1, q_2) \to q$, on crée une feature à trois noeuds $f_{g(q_1,q_2) \to q}$ de la forme :

$$f_{g(q_1,q_2) \to q}(y_n, y_{ni}, y_{n(i+1)}, x) = \begin{cases} 1 & \text{si } x_n = g, y_n = q, y_{ni} = q_1 \text{ et } y_{n(i+1)} = q_2 \\ 0 & \text{sinon} \end{cases}$$

À chacune des features ainsi créées, on associe comme poids le logarithme de la probabilité de la règle correspondante dans l'automate. Toutefois, créer uniquement les features correspondant aux règles de l'automate ne suffit pas. En effet, dans un automate d'arbres stochastique, les transitions non définies sont en fait des transitions dont la probabilité est nulle. Elles doivent donc être transposées comme telles. Notre Tree CRF est ainsi complété par des features pour chaque transition de probabilité nulle. On associe à ces features le poids $\log(0) = -\infty$.

Enfin, la dernière composante de l'automate à représenter est la probabilité d'un état d'être final. On représente cette propriété par une feature à un noeud de la forme suivante :

$$f_{\mathit{final}(q)}(y_n, x) = \begin{cases} 1 & \text{si } y_n = q \text{ et } n \text{ est la racine} \\ 0 & \text{sinon} \end{cases}$$

Le poids associé à cette feature est aussi le logarithme de la probabilité de cet état d'être final. De la même façon que pour les transitions, si un état q a une probabilité nulle d'être final, il faut créer une feature $f_{\mathit{final}(q)}$ et lui donner comme poids $-\infty$.

Le Tree CRF ainsi défini permet d'obtenir la même distribution de probabilité conditionnelle que dans l'automate. En effet, la probabilité dans un CRF est de la forme suivante :

$$p_{\mathit{crf}}(y|x) = \frac{1}{Z(x)} \prod_{c \in \mathcal{C}} \prod_{k} \exp(\lambda_k f_k)$$

Soient A un automate stochastique et crfA le Tree CRF obtenu par la construction précédente. Pour chaque clique, seule la feature correspondant à la règle qui s'applique vaut 1, et à la racine s'applique la feature correspondant à la probabilité de l'état d'être final. Pour un arbre x et un calcul réussi y de A, on a :

$$p_{\mathit{crfA}}(y|x) = \frac{1}{Z(x)} \exp(\lambda_{\mathit{final}(y_\epsilon)}) \prod_{\mathit{regle}} \exp(\lambda_{\mathit{regle}})$$

où $\prod_{\mathit{regle}} \exp(\lambda_{\mathit{regle}})$ est le produit sur toutes les règles utilisées dans le calcul y de l'exponentielle du poids associé à la règle. Or on a choisi $\lambda_{\mathit{final}(q)} = \log(\mathit{final}(q))$ et

$\lambda_{regle} = \log(p(regle))$. On en déduit que :

$$p_{\mathrm{crfA}}(y|x) = \frac{1}{Z(x)} final(y_\epsilon) \times \prod_{regle} p(regle) = \frac{1}{Z(x)} p_A(y,x)$$

Or $Z(x) = \sum_y \prod_{c\in\mathcal{C}} \prod_k \exp(\lambda_k f_k)$. Pour tout calcul réussi, nous avons montré l'égalité entre $p_A(y,x)$ et le produit sur toutes les cliques. Pour tout y ne correspondant pas un calcul réussi, $p_A(y,x) = 0$, et, avec le choix des poids pour les transitions non définies, le produit sur toutes les cliques vaut 0. Par conséquent, $\sum_y \prod_{c\in\mathcal{C}} \prod_k \exp(\lambda_k f_k) = \sum_y p_A(y,x)$, et donc $Z(x) = \sum_y p_A(y,x)$. On en déduit donc que la probabilité définie par le Tree CRF $crfA$ est la même que celle de l'automate A :

$$p_{crfA}(y|x) = p_A(y|x) \tag{3.2}$$

Pour les arbres d'arité fixée, les probabilités conditionnelles pouvant être définies par des automates d'arbres probabilistes peuvent être définies par des Tree CRFs. La réciproque est fausse car on peut exprimer dans les features des propriétés non régulières sur les arbres d'entrée. Cette étude doit être étendue aux arbres ordonnés d'arité non bornée bien qu'il n'existe pas de définition consensuelle d'automates d'arbres stochastiques dans ce cas.

4 Algorithmes pour les Tree CRFs

Cette section se concentre sur l'adaptation des algorithmes de programmation dynamique pour les tâches d'annotation et d'apprentissage des paramètres dans les Tree CRFs. De la même façon que pour les CRFs sur les séquences, les algorithmes employés s'inspirent d'algorithmes classiques. Nous commencerons par décrire l'adaptation de l'algorithme de Viterbi à nos Tree CRFs. Puis nous présenterons un algorithme permettant des calculs efficaces lors de la phase d'apprentissage.

4.1 Recherche de la meilleure annotation d'un arbre

La recherche de l'annotation la plus probable consiste à trouver l'annotation $\hat{y}$ qui maximise $p(\hat{y}|x)$ étant donnée une observation x et un Tree CRF dont on connaît les paramètres. Nous voulons donc trouver :

$$\hat{y} = \arg\max_y p(y|x)$$

De la formule (3.1), on déduit :

$$\hat{y} = \arg\max_y \left(\sum_{c\in\mathcal{C}} \Lambda F_{n,n.i} \right)$$

Calculer ce score pour chaque annotation y possible étant trop coûteux, on utilise l'algorithme de Viterbi. Pour cela, on définit $\delta_n(y_n)$ comme étant le "score" de la meilleure

annotation du sous-arbre partant du nœud n, où le nœud n est annoté par y_n. On appelle score la somme pondérée des features sur toutes les cliques de l'arbre.

$$\delta_n(y_n) = \begin{cases} 0 & \text{si } n \text{ est une feuille} \\ \max\limits_{y_{n.1} \cdots y_{n.m}} \left(\sum\limits_{i=1}^{m} \left(\delta_{n.i}(y_{n.i}) + \Lambda F_{n,n.i} \right) \right) & \text{sinon} \end{cases}$$

où m est le nombre de fils du nœud n.

Toutefois, dans des arbres d'arité supérieure à 2 et tout particulièrement pour les arbres d'arité non bornée, la présente définition de $\delta_n(y_n)$ ne suffit pas à obtenir un algorithme efficace. En effet, son calcul nécessite de trouver un maximum pour toutes les combinaisons d'annotations possibles pour les fils du nœud n, ce qui fait, en notant $\mathcal{Y}$ l'alphabet des étiquettes et $\|\mathcal{Y}\|$ sa taille, $\|\mathcal{Y}\|^m$ si n a m fils. En plus de la programmation dynamique en profondeur déjà présente, nous introduisons donc une programmation dynamique en largeur avec le coefficient $\delta'_{n,k}(y_n, y_{n.k})$ qui s'inspire du δ utilisé dans le cas des CRFs sur les séquences. On le définit comme le score de la meilleure annotation du sous-arbre partant de n, où on n'a gardé que les k premiers fils de n et où n est annoté par y_n et $n.k$ par $y_{n.k}$.

$$\delta'_{n,k}(y_n, y_{n.k}) = \begin{cases} \delta_{n.m}(y_{n.m}) & \text{si } k = m \\ \delta_{n.k}(y_{n.k}) & \text{sinon} \\ + \max\limits_{y_{n.(k+1)}} \left(\Lambda F_{n,n.i} + \delta'_{n,(k+1)}(y_{n.(k+1)}) \right) & \end{cases}$$

$\delta_n(y_n)$ peut alors se réécrire de la façon suivante :

$$\delta_n(y_n) = \begin{cases} 0 & \text{si } n \text{ est une feuille} \\ \max\limits_{y_{n.1}} \delta'_{n,1}(y_{n.1}) & \text{sinon} \end{cases}$$

Il suffit alors de rechercher $max_{y_\epsilon} \delta_\epsilon(y_\epsilon)$, et de retrouver le chemin de Viterbi correspondant pour obtenir la meilleure annotation $\hat{y}$.

4.2 Apprentissage des paramètres du modèle

Comme pour les CRFs sur les séquences, la phase d'apprentissage d'un Tree CRF consiste à trouver les paramètres Λ qui maximisent la log-vraisemblance du modèle par rapport à un ensemble d'apprentissage S. On utilise une formule de log-vraisemblance légèrement différente (à un coefficient multiplicateur près) de la formule (2.3) :

$$\mathcal{L}_\Lambda = \sum_{(x,y) \in S} \tilde{p}(x, y) \log p_\Lambda(y|x) \tag{4.1}$$

où $\tilde{p}(x, y)$ est la fréquence d'apparition du couple (x, y) dans S.

En remplaçant $p_\Lambda(y|x)$ par sa définition (*cf.* eq. (3.1)), on obtient :

$$\mathcal{L}_\Lambda = \sum_{x,y} \tilde{p}(x, y) \Big[\prod_{c \in \mathcal{C}} \exp \Lambda F_{n,n.i} \Big] - \sum_{x} \tilde{p}(x) \log Z(x)$$

La fonction de log-vraisemblance ainsi obtenue est concave, ce qui signifie qu'il est possible de trouver un maximum global qui correspond aux paramètres pour lesquels la dérivée s'annule. Il est toutefois impossible de calculer analytiquement cette dérivée par rapport à l'ensemble des paramètres du modèle. On utilise donc une méthode de descente de gradient (L-BFGS) pour laquelle il est nécessaire de calculer les dérivées partielles de $\mathcal{L}_\Lambda$ par rapport à chaque paramètre du Tree CRF. Ces dérivées partielles s'expriment :

$$\frac{\partial \mathcal{L}_\Lambda}{\partial \lambda_k} = \sum_{x,y} \tilde{p}(x,y) \sum_c f_k - \sum_x \tilde{p}(x) \sum_y p_\Lambda(y|x) \sum_c f_k \tag{4.2}$$

$$= E_{\tilde{p}(x,y)}[f_k] - E_{p_\Lambda(y|x)}[f_k] \tag{4.3}$$

où $E_{\tilde{p}(x,y)}[f_k]$ est l'espérance de la feature f_k par rapport à la distribution empirique $\tilde{p}(x,y)$ et $E_{p_\Lambda(y|x)}[f_k]$ est l'espérance de f_k par rapport à la distribution du Tree CRF. Si le calcul de $E_{\tilde{p}(x,y)}[f_k]$ est assez immédiat, le calcul de $E_{p_\Lambda(y|x)}[f_k]$ quant à lui pose un problème en raison de la somme sur toutes les annotations y possibles.

Les deux difficultés lors de la phase d'apprentissage des paramètres sont donc le calcul du coefficient de normalisation $Z(x)$ qui intervient dans le calcul de $p_\Lambda(y|x)$, et le calcul des dérivées partielles par rapport à chaque paramètre du Tree CRF. Pour effectuer ces calculs de manière efficace, nous avons mis en œuvre une méthode de programmation dynamique s'inspirant de l'algorithme *Inside-Outside* utilisé dans les grammaires algébriques probabilistes (Probabilistic Context Free Grammars ou PCFGs) pour la récursion en profondeur et de l'algorithme Forward-Backward des HMMs pour la récursion en largeur. Ces deux algorithmes fondateurs sont décrits en détails dans (Manning & Schütze, 1999).

Nous abordons tout d'abord le problème du calcul de $Z(x)$. Pour cela, on définit le coefficient *inside* $\beta_n(y_n)$ comme la somme des probabilités non normalisées de toutes les annotations possibles du sous-arbre partant du nœud n, où n est annoté par y_n.

$$\beta_n(y_n) = \begin{cases} 1 & \text{si } n \text{ est une feuille} \\ \displaystyle\sum_{y_{n.1},\ldots,y_{n.m}} \left(\prod_{i=1}^{m} \beta_{n.i}(y_{n.i}) \exp(\Lambda F_{n,n.i}) \right) & \text{sinon} \end{cases}$$

Comme pour l'adaptation de l'algorithme de Viterbi décrite précédemment, cette formule ne suffit pas dans le cas des arbres d'arité supérieure à 2, en raison de la somme sur toutes les combinaisons d'étiquettes possibles pour les fils du nœud n. Nous introduisons donc de la programmation dynamique en largeur en nous inspirant des coefficients *backward* utilisés dans les HMMs. Nous appelons ce coefficient $\beta'_{n,k}(y_{n.k})$.

On définit $\beta'_{n,k}(y_{n.k})$ comme la probabilité non normalisée du sous-arbre partant de n dont on n'a gardé que les k premiers fils de n et où n est annoté par y_n et $n.k$ par $y_{n.k}$.

$$\beta'_{n,k}(y_n, y_{n.k}) = \begin{cases} \beta_{n.m}(y_{n.m}) & \text{si } k = m \\ \beta_{n.k}(y_{n.k}) \displaystyle\sum_{y_{n.(k+1)}} \left(\exp(\Lambda F_{n,n.k}) \beta'_{n,(k+1)}(y_{n.(k+1)}) \right) & \text{sinon} \end{cases}$$

Par conséquent, $\beta_n(y_n)$ peut être réécrit comme suit :

$$\beta_n(y_n) = \begin{cases} 1 & \text{si } n \text{ est une feuille} \\ \sum_{y_{n.1}} \beta'_{n,1}(y_{n.1}) & \text{sinon} \end{cases}$$

En utilisant le coefficient ainsi défini, il est possible de calculer efficacement $Z(x)$ de la façon suivante :

$$Z(x) = \sum_{y_\epsilon} \beta_\epsilon(y_\epsilon) \tag{4.4}$$

La complexité du calcul de $Z(x)$ est en $O(|x| \times \|\mathcal{Y}\|^3)$. Elle est donc linéaire par rapport à la taille de l'arbre d'entrée. Elle est cubique par rapport à la taille de l'alphabet des étiquettes, il faudra donc veiller à choisir des alphabets de taille raisonnable.

Le problème de la complexité du calcul des dérivées partielles de $\mathcal{L}_\Lambda$ peut être résolu de deux façons différentes. La première consiste à approximer ce calcul. En effet, il est possible d'obtenir une valeur approchée de la dérivée partielle en λ_k en faisant varier λ_k d'une valeur ϵ positive très proche de zéro et en calculant la différence :

$$\frac{\partial \mathcal{L}_\Lambda}{\partial \lambda_k} = \frac{\mathcal{L}_{\Lambda^+} - \mathcal{L}_{\Lambda^-}}{2\epsilon}$$

où Λ^+ est le vecteur des paramètres du modèle où λ_k a été remplacé par $\lambda_k + \epsilon$ et Λ^- est le vecteur des paramètres du modèle où λ_k a été remplacé par $\lambda_k - \epsilon$. C'est cette méthode qui est implantée actuellement.

L'autre méthode possible est de calculer la valeur exacte de ces dérivées partielles en utilisant une technique de programmation dynamique. Elle a été utilisée par (Lafferty *et al.*, 2001) dans le cas des CRFs sur les séquences à l'aide des coefficients *forward* et *backward*. Nous l'adaptons ici au cas des arbres. La difficulté réside dans le calcul de l'espérance d'une feature f_k par rapport au modèle.

Nous définissons donc le coefficient *outside* $\alpha_n(y_n)$ qui correspond à la somme des probabilités non normalisées de toutes les annotations possibles du complémentaire du sous-arbre partant du nœud n, où n est annoté par y_n.

$$\alpha_n(y_n) = \begin{cases} 1 & \text{si } n \text{ est la racine} \\ \sum_{y_{n'}} \Bigg(\alpha_{n'}(y_{n'}) & \\ \quad \sum_{y_{n'.1} \cdots y_{n'.m} (y_{n'.i} = y_n)} \Big(\prod_{j=1}^{m} \exp(\Lambda F_{n',n'.j}) \prod_{j=1 (j \neq i)}^{m} \beta_{n'.j}(y_{n'.j}) \Big) \Bigg) & \end{cases}$$

Pour la programmation dynamique en largeur, on introduit le coefficient $\beta''_{n,k}(y_{n.k})$ s'inspirant des coefficients *forward* utilisés dans les HMMs. Il est défini comme suit :

$$\beta''_{n,k}(y_n, y_{n.k}) = \begin{cases} 1 & \text{si } k = 1 \\ \sum_{y_{n(k-1)}} \Big(\beta_{n(k-1)}(y_{n(k-1)}) \exp \Lambda F_{n,n(k-1)} \beta''_{n,k-1}(y_n, y_{n(k-1)}) \Big) & \end{cases}$$

A l'aide de ce coefficient, on peut ré-écrire $\alpha_n(y_n)$:

$$\alpha_n(y_n) = \begin{cases} 1 & \text{si } n \text{ est la racine} \\ \sum_{y_{n'}} \left(\alpha_{n'}(y_{n'}) \frac{\beta''_{n',i}(y_{n'},y_n)\beta'_{n',i}(y_{n'},y_n)}{\beta_n(y_n)} \right) & \text{sinon } (n = n'.i) \end{cases}$$

On peut remarquer que d'après les définitions des coefficients *inside* et *outside* définis précédemment, $\alpha_n(y_n)\beta_n(y_n)$ représente la somme des probabilités non normalisées des annotations y pour lesquelles le nœud n est annoté par y_n. Si l'on étend le raisonnement, la formule (4.5) correspond à la somme des probabilités non normalisées des annotations y où les nœuds n, ni et $n(i+1)$ sont annotés par y_n, y_{ni} et $y_{n(i+1)}$. Ces deux remarques servent d'intuition pour la reformulation de l'espérance d'une feature par rapport au Tree CRF.

$$P(y_n, y_{ni}, y_{n(i+1)}) = \alpha_n(y_n) \exp \Lambda F_{n,ni} \beta''_{n,ni}(y_n, y_{ni}) \beta_{ni}(y_{ni}) \beta'_{n,n(i+1)}(y_n, y_{n(i+1)}) \tag{4.5}$$

Il est alors possible d'écrire l'espérance d'une feature f_k par rapport au modèle de la façon suivante :

$$E_{p_\Lambda(y|x)}[f_k] = \sum_x \tilde{p}(x) \sum_c \sum_{(y_n, y_{n.i}, y_{n.(i+1)})} f_k(y_n, y_{n.i}, y_{n.(i+1)}) \frac{P(y_n, y_{ni}, y_{n(i+1)})}{Z(x)} \tag{4.6}$$

Ceci nous permet de calculer efficacement la dérivée partielle de la fonction de log-vraisemblance à l'aide des formules (4.3) et (4.6).

4.3 Implantation

Il existe déjà une librairie, le "CRF Project" [1], implantant en JAVA les CRFs pour l'annotation de séquences. Toutefois, à ce jour, aucune librairie n'offre la possibilité d'effectuer de l'annotation d'arbres avec des CRFs. Nous avons donc mis en œuvre l'implantation d'une librairie Tree CRF en JAVA. Celle-ci utilise les algorithmes décrits précédemment afin de proposer un outil rapide pour :

- calculer la probabilité conditionnelle d'une annotation y sachant un arbre x.
- trouver l'annotation $\hat{y}$ la plus probable étant donné un arbre x et un Tree CRF dont on connaît les paramètres.
- apprendre les paramètres d'un Tree CRF étant donné un échantillon d'observations annotées. L'algorithme d'apprentissage utilisé est la descente de gradient L-BFGS fournie par le "RISO Project" [2].

Dans l'implantation de cette librairie, il est nécessaire de faire attention à deux points essentiels. Dans un premier temps, les valeurs des coefficients *inside* décrits plus haut sont mémorisées dans des matrices de taille $|x| \times \|\mathcal{Y}\|$. Il est donc essentiel d'optimiser le temps d'accès à ces matrices. Pour cela, nous nous sommes inspirés de la représentation des matrices dans la librairie COLT [3].

[1] `http://crf.sourceforge.net/`

[2] `http://riso.sourceforge.net/`

[3] `http://hoschek.home.cern.ch/hoschek/colt/`

Il est aussi nécessaire de veiller à limiter la complexité des features. En effet, au sein d'une feature, il est possible d'effectuer des calculs potentiellement coûteux sur l'observation x, comme d'évaluer la taille du sous-arbre ou de tester la présence d'un nœud dans x, etc. Afin d'éviter d'effectuer ces calculs à chaque appel de la feature, ceux-ci sont pré-calculés lors du chargement en mémoire de l'arbre x.

5 Conclusion

Dans cet article, nous avons proposé une adaptation des CRFs, un modèle qui a fait ses preuves pour l'annotation de séquences, au cas de l'annotation d'arbres ordonnés d'arité non bornée : les Tree CRFs. Nous avons donné un aperçu de l'expressivité de ce modèle en le comparant aux automates d'arbres stochastiques. Les Tres CRFs se comportent vis-à-vis de ces automates de la même façon que les CRFs sur les séquences se comportent vis-à-vis des HMMs. Enfin, nous avons décrit les algorithmes de recherche de l'annotation la plus probable et d'apprentissage des paramètres du modèle. L'implantation actuelle nous a permis de tester ces algorithmes et leurs performances sur des données artificielles de petite taille. Ce travail préliminaire doit donc être poursuivi pour être validé sur des données semi-structurées de grande taille. Il doit également être confronté à diverses tâches.

Nous pourrons tout d'abord nous confronter à diverses tâches d'extraction d'informations à partir de données semi-structurées de type XML ou HTML. Nous pensons que la souplesse apportée par la façon de définir des features nous permettra d'obtenir des résultats au moins comparables à ceux obtenus par les systèmes existants dans le domaine pour les tâches d'extraction classique (extraction d'un ou plusieurs champs). Nous estimons que l'intérêt principal des Tree CRFs sera de pouvoir traiter des problèmes d'extraction pour lesquels la sortie est elle-même semi-structurée.

Nous pourrons aussi tester notre modèle sur les tâches de transformation de documents semi-structurés. Pour cela, nous devrons définir une sémantique à notre alphabet d'annotations indiquant à chaque nœud l'opération de transformation à effectuer. De cette manière, la transformation s'effectuera en deux parties, la première consistant en l'annotation de l'arbre à l'aide du Tree CRF, la seconde effectuant les transformations indiquées par les annotations des nœuds.

Enfin, nous avons montré comment il est possible de représenter un automate d'arbres stochastique par un Tree CRF. Une des difficultés posée par les champs conditionnels aléatoires réside dans le choix des features. Une perspective de notre travail est la combinaison des approches : utiliser des techniques d'inférence grammaticale pour générer les features (les règles de l'automate) puis utiliser les algorithmes présentés ici pour apprendre les paramètres du Tree CRF.

Références

BYRD R. H., LU P., NOCEDAL J. & ZHU C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, **16**(5), 1190–1208.

CARRASCO R. & ONCINA J. (1994). Learning stochastic regular grammars by means of a state merging method. In *ICGI'04 : Proceedings of the seventh International Colloquium on Grammatical Inference*, p. 139–152.

COHN T. & BLUNSOM P. (2005). Semantic role labelling with tree conditional random fields. In *CoNLL'05 : Proceedings of The Ninth Conference on Natural Language Learning"*.

DENOYER L. & GALLINARI P. (2004). Bayesian network model for semi-structured document classification. *Information Processing and Management*, **40**(5), 807–827.

HABRARD A., BERNARD M. & SEBBAN M. (2005). Detecting irrelevant subtrees to improve probabilistic learning from tree-structured data. *Fundamenta Informaticae*, **66**(1-2), 103–130.

HAMMERSLEY J. & CLIFFORD P. (1971). Markov fields on finite graphs and lattices. Unpublished.

LAFFERTY J., MCCALLUM A. & PEREIRA F. (2001). Conditional random fields : Probabilistic models for segmenting and labeling sequence data. In *ICML'01 : Proceedings of the 18th International Conf. on Machine Learning*, p. 282–289.

MANNING C. D. & SCHÜTZE H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts : The MIT Press.

MCCALLUM A. & LI W. (2003). Early results for named entity recognition with conditional random fields. In *CoNLL'2003 : Proceedings of The Seventh Conference on Natural Language Learning*.

PINTO D., MCCALLUM A., LEE X. & CROFT W. (2003). Table extraction using conditional random fields. In *SIGIR'03 : Proceedings of the 26th ACM SIGIR*.

RABINER L. R. & JUANG B. H. (1986). An introduction to hidden markov models. *IEEE ASSP Magazine*, **3**(1), 4–16.

SHA F. & PEREIRA F. (2003). Shallow parsing with conditional random fields. In *Technical Report CIS TR MS-CIS-02-35, University of Pennsylvania, 2003.*

TASKAR B., CHATALBASHEV V., KOLLER D. & GUESTRIN C. (2005). Learning structured prediction models : a large margin approach. In *ICML'05 : Proceedings of the 22nd international conference on Machine learning*, p. 896–903.

TSOCHANTARIDIS I., HOFMANN T., JOACHIMS T. & ALTUN Y. (2004). Support vector learning for interdependent and structured output spaces. In *ICML'04 : Proceedings of the 21st international conference on Machine learning*, p. 823–830.

WALLACH H. (2002). Efficient training of conditional random fields. Master's thesis, University of Edinburgh.