Apprentissage génératif de la structure de réseaux logiques de Markov à partir d'un graphe des prédicats

Quang-Thang Dinh*, Matthieu Exbrayat*, Christel Vrain* **

*LIFO, Université d'Orléans, Rue Léonard de Vinci,
B.P. 6759, 45067 ORLEANS Cedex 2, France
{thang.dinh,matthieu.exbrayat,christel.vrain}@univ-orleans.fr
http://www.univ-orleans.fr/lifo/
** Université Pierre et Marie Curie - Paris 6 - LIP6
4 Place Jussieu - 75252 Paris cedex 05

Résumé. Les Réseaux Logiques de Markov (MLNs) combinent l'apport statistique des Réseaux de Markov à la logique du premier ordre. Dans cette approche, chaque clause logique se voit affectée d'un poids, l'instantiation des clauses permettant alors de produire un Réseau de Markov. L'apprentissage d'un MLN consiste à apprendre d'une part sa structure (la liste de clauses logiques) et d'autre part les poids de celles-ci. Nous proposons ici une méthode d'apprentissage génératif de Réseau Logique de Markov. Cette méthode repose sur l'utilisation d'un graphe des prédicats, produit à partir d'un ensemble de prédicats et d'une base d'apprentissage. Une méthode heuristique de variabilisation est mise en œuvre afin de produire le jeu de clauses candidates. Les résultats présentés montrent l'intérêt de notre approche au regard de l'état de l'art.

1 Introduction

L'Apprentissage Statistique Relationnel (SRL) (Getoor et Taskar (2007)) consiste à combiner le pouvoir descriptif de l'apprentissage relationnel à la souplesse de l'apprentissage statistique. Diverses approches ont été proposées au cours des quinze dernières années, tels que les programmes logiques stochastiques (Muggleton (1996)), PRISM (Sato et Kameya (1997)), MACCENT (Dehaspe (1997)), les modèles relationnels probabilistes (PRM) (Friedman et al. (1999)), les programmes logiques bayésiens (BLP) (Kersting et De Raedt (2001)) et les réseaux relationnels dépendants (Neville et Jensen (2004)). Les réseaux logiques de Markov (Richardson et Domingos (2006)), qui constituent l'une des approches les plus récentes de ce domaine, reposent sur la combinaison de la logique du premier ordre avec les réseaux de Markov. Un réseau de Markov (Pearl (1988)) est un graphe, dont les nœuds représentent des variables aléatoires et dont les arêtes expriment les dépendances conditionnelles entre ces variables. Chaque clique du graphe correspond ainsi à un ensemble de variables conditionnellement dépendantes. On associe à chaque clique un poids. Un tel réseau permet ensuite d'inférer la valeur d'une ou plusieurs variables. Un réseau logique de Markov, ou MLN, est constitué d'un ensemble de clauses logiques pondérées. Ces clauses sont constituées d'atomes, lesquels

peuvent être vus comme des prototypes pour la construction d'un réseau de Markov. En effet, si l'on dispose d'un ensemble de constantes, on peut produire, en instantiant les clauses, un ensemble d'atomes clos qui constitueront les nœuds d'un réseau de Markov. Les nœuds issus d'une même instanciation de clause seront liés, et les cliques ainsi produites seront affectées du poids de la clause dont elles dérivent.

L'apprentissage d'un MLN peut être décomposé en deux phases, consistant à apprendre respectivement la structure (i.e. les clauses en logique du premier ordre) et les paramètres (i.e. le poids de ces clauses) de ce réseau. Les premières stratégies développées se limitaient à l'apprentissage des paramètres, la structure étant fournie à l'algorithme et donc considérée comme proposée par un expert ou apprise lors d'une étape antérieure. De telles approches pouvaient conduire à des résultats sous-optimaux quand la structure fournie ne reflétait pas ou mal les dépendances essentielles du domaine étudié. En conséquence, les travaux récents sur les MLN se sont concentrés sur des méthodes d'apprentissage globales intègrant les étapes d'apprentissage de la structure et des poids. Néanmoins, cette intégration reste délicate en raison du très vaste espace de recherche induit par la logique du premier ordre. Peu d'approches pratiques ont donc été proposées, les principales reposant sur l'approche top-down (Kok et Domingos (2005)), l'approche bottom-up (BUSL, Mihalkova et Mooney (2007)),la descente de gradient (Iterated Local Search, Biba et al. (2008)),la généralisation d'hypergraphes (Learning via Hyper-graph Lifting, Kok et Domingos (2009)), l'apprentissage à partir de motifs (Learning using Structural Motifs, Kok et Domingos (2010)), la propositionalisation (Heuristic Generative Structure learning for MLNs, Dinh et al. (2010)) et l'apprentissage à partir de réseaux bayésiens (Moralized Bayes Net, Khosravi et al. (2010)).

Du point de vue des objectifs, les réseaux logiques de Markov permettent d'envisager l'apprentissage génératif (apprentissage pour tous les prédicats du domaine) et l'apprentissage discriminant (valeur d'un seul prédicat). Dans cet article nous nous concentrerons sur l'apprentissage génératif, notre principale contribution étant un algorithme innovant pour l'apprentissage génératif de la structure d'un MLN. Cet algorithme construit d'abord un graphe des prédicats, mettant en évidence les liens entre atomes partageant des arguments de même type et synthétisant les chemins pouvant exister dans la base d'apprentissage. La construction de clauses candidates pour le MLN se fait ensuite sur la base de ce graphe. Chaque chemin est transformé en clause par le biais d'une technique heuristique de variabilisation. L'algorithme produit des clauses de longueur croissante, en commençant par les clauses de longueur 2. L'intérêt de chaque clause est ensuite évalué, les plus pertinentes étant ajoutées dans le réseau logique de Markov final.

Cet article est organisé comme suit : nous rappelons et introduisons quelques définitions en section 2. Nous décrivons notre algorithme en section 3. Les expérimentations et leurs résultats sont présentés en section 4. Nous concluons en section 5.

2 Préliminaires

Nous rappelons ici quelques définitions de la logique du première ordre qui seront utilisées au long de cet article. Nous considérons un langage du premier ordre sans fonction, composé d'un ensemble $\mathcal{P}$ de symboles de prédicats, d'un ensemble C de constantes et d'un ensemble de variables. Un *atome* est une expression $p(t_1, \ldots, t_k)$, où p est un prédicat et les t_i sont des variables ou des constantes. Un *littéral* est un atome (littéral positif) ou la négation d'un atome

(littéral négatif) ; nous parlerons de littéral clos lorsqu'il ne comporte que des constantes et de littéral avec variables quand il ne contient que des variables. Une *clause* est une disjonction de littéraux. La *longueur* d'une clause *c*, notée *len(c)*, correspond au nombre de littéraux formant *c*. Deux littéraux (resp. avec variables) sont dits *connectés* s'ils partagent au moins une constante (resp. une variable). Une clause est dite *connectée* quand il existe un ordonnancement de ses littéraux $L_1 \vee \ldots \vee L_p$ tel que pour chaque littéral L_j, $j = 2 \ldots p$, il existe une variable (ou une constante) apparaissant dans L_j et L_i, avec $i < j$. Un monde (aussi appelé parfois interprétation de Herbrand) est une affectation de valeur de vérité à tous les atomes clos. Une *base de données* est une spécification partielle d'un monde dont chaque atome est vrai, faux ou (implicitement) inconnu. Dans cet article, nous utilisons l'hypothèse du monde clos : tout atome qui n'est pas dans la base de données est considéré comme *faux*.

Un réseau logique de Markov est un ensemble de formules du premier ordre pondérées $\{F_i, w_i\}$. Associé à un ensemble de constantes, il définit un réseau de Markov (Pearl (1988)) dont chaque nœud correspond à un atome clos et chaque caractéristique (clique) correspond à une instanciation d'une formule. Le poids d'une clique est le poids de la formule qui l'a produite. La probabilité d'observer un monde *x* est : $P(X = x) = \frac{1}{Z} \exp(\sum_i w_i n_i(x))$ où $n_i(x)$ est le nombre d'instanciations vraies de la formule F_i dans x, et w_i est le poids associé à cette formule. *Z* est une constante de normalisation.

Soit $Y = \{Y_1, \ldots, Y_n\}$ l'ensemble des atomes dont on souhaite inférer la valeur et $X = \{X_1, \ldots, X_m\}$ l'ensemble des atomes de valeur connue ; la log-vraisemblance conditionnelle (CLL) de *Y* sachant *X* est : $\log P(Y = y|X = x) = \sum_{j=1}^{n} \log P(Y_j = y_j|X = x)$. La pseudo-log-vraisemblance (PLL) du monde x est donnée par : $\log P(X = x) = \sum_{l=1}^{n} \log P(X_l = x_l|MB_x(X_l))$ où X_l est un atome clos et x_l sa valeur de vérité (0 or 1) dans x et $MB_x(X_l)$ est l'état de la couverture de Markov de X_l dans x. Rappelons que la couverture de Markov d'un atome est constituée des voisins de cet atome dans le réseau qui présentent une dépendance conditionnelle directe avec celui-ci.

Dans un cadre génératif, on cherche un ensemble de formules pondérées optimisant la pseudo-log-vraisemblance (PLL). La technique d'optimisation de la PLL la plus courante se nomme *L-BFGS* (Sha et Pereira (2003)) et repose sur la méthode de quasi-Newton.

Le calcul de la PLL est fortement influencé par la présence de clauses contenant un grand nombre de variables, et présentant donc un grand nombre d'instanciations. La PLL est conc couramment remplacée par une mesure dérivée, introduite par Kok et Domingos (2005), appelée pseudo log-vraisemblance pondérée (WPLL), définie par :

$$\log P(X = x) = \sum_{r \in R} c_r \sum_{k=1}^{g_r} \log P(X_{r,k} = x_{r,k}|MB_x(X_{r,k})),$$

où *R* est l'ensemble des prédicats, g_r est le nombre d'atomes clos de *r*, et $x_{r,k}$ est la valeur de vérité (0 ou 1) du *k*-ème atome clos de *r*. La valeur de c_r dépend des buts de l'utilisateur. Dans nos expériences, nous avons choisi la même valeur que Kok et Domingos (2005) : $c_r = 1/g_r$.

3 Apprentissage génératif de la structure reposant sur un graphe de prédicats

Dans cette section, nous présentons notre approche pour l'apprentissage de structures de MLN à partir d'une base de données DB et d'un MLN donné en entrée (éventuellement vide).

Nous définissons d'abord la notion de graphe de prédicats et nous décrivons ensuite comment nous l'utilisons pour apprendre la structure d'un MLN.

3.1 Graphe des prédicats

Nous considérons un ensemble $\mathcal{P}$ de *m* prédicats $\{p_1, \ldots, p_m\}$ et une base de données DB composée de littéraux clos construits sur ces prédicats.

Définition 1 *Un atome modèle d'un prédicat p est une expression p($type_1$, ... , $type_n$), où $type_i$, $1 \leq i \leq n$, indique le type du i-ème argument.*

Définition 2 *Un lien entre deux atomes modèles $p_i(a_{i_1}, \ldots, a_{i_q})$ et $p_j(a_{j_1}, \ldots, a_{j_k})$ est une liste ordonnée de couples de positions u et v tels que les types des arguments à la position u dans p_i et v dans p_j sont identiques. Il est décrit par : $link(p_i(a_{i_1}, ..., a_{i_q}), p_j(a_{j_1}, ..., a_{j_k})) = \langle i_c\ j_d | \ldots \rangle$, où $a_{i_c} = a_{j_d}$, $1 \leq c \leq q, 1 \leq d \leq k$.*

Un lien entre deux prédicats p_i et p_j dénoté par $link(p_i, p_j)$ est l'ensemble de tous les liens possibles entre leurs atomes modèles $p_i(a_{i_1}, \ldots, a_{i_q})$ et $p_j(a_{j_1}, \ldots, a_{j_k})$.

Quand deux atomes modèles ne partagent pas d'argument, il n'existe pas de lien entre eux.

Définition 3 *Une formule satisfaisant un lien $link(p_i(a_{i_1}, \ldots, a_{i_q}), p_j(a_{j_1}, ..., a_{j_k})) = \langle si_1\ sj_1 | \ldots | si_c\ sj_c | \rangle$ est de la forme $p_i(V_{i_1}, \ldots, V_{i_q}) \wedge p_j(V_{j_1}, \ldots, V_{j_k}))$, où $V_{i_{si_d}} = V_{j_{sj_d}}$, pour $1 \leq d \leq c$ et les autres variables V_t, $t = i_1, \ldots, i_q, j_1, ..., j_k$ sont toutes distinctes.*

Naturellement, les définitions de lien et de formule satisfaisant un lien peuvent être appliquées à la négation. Par exemple, $link(p_i(a_{i_1}, ..., a_{i_q}), !p_j(a_{j_1}, ..., a_{j_k}))$ est le lien entre $p_i(a_{i_1}, ..., a_{i_q})$ et la négation de $p_j(a_{j_1}, ..., a_{j_k})$. Seuls les arguments sont pris en compte, et donc $link(p_i, !p_j) = link(p_i, p_j)$.

Exemple 1 *Nous considérons un langage composé de deux prédicats* AdvisedBy *et* Professor *respectivement avec comme atomes modèles* AdvisedBy(person, person) *et* Professor(person). *L'argument (le type)* person *apparaît à la position* 0 *de* Professor(person) *et aux positions* 0 *et* 1 *de* AdvisedBy(person, person). *Des liens possibles entre eux sont donc* $\langle 0\ 0 \rangle$ *ou encore* $\langle 0\ 1 \rangle$. *Une formule satisfaisant ce dernier lien est* Professor(A) $\wedge$ AdvisedBy(B, A).
De même, nous avons link(AdvisedBy, AdvisedBy)= $\{\langle 0\ 0 \rangle, \langle 0\ 1 \rangle, \langle 1\ 0 \rangle, \langle 1\ 1 \rangle, \langle 0\ 0 \mid 1\ 0 \rangle, \langle 0\ 1 \mid 1\ 0 \rangle, \langle 0\ 0 \mid 1\ 1 \rangle\}$.
Notons que plusieurs liens ne sont pas considérés. Par exemple, le lien $link(AdvisedBy, AdvisedBy) = \langle 0\ 0 \mid 1\ 1 \rangle$ conduit à une formule composée de deux littéraux identiques. Le lien $\langle 1\ 0 \mid 0\ 1 \rangle$ est similaire au lien $\langle 0\ 1 \mid 1\ 0 \rangle$, dans la mesure où ils conduisent à deux formules identiques à un renommage des variables près. Le lien $\langle 1\ 1 \mid 0\ 0 \rangle$ est aussi similaire au lien $\langle 0\ 0 \mid 1\ 1 \rangle$. Il en est de même pour $link(Professor, Professor) = \langle 0\ 0 \rangle$, $link(Professor, !Professor) = \langle 0\ 0 \rangle \ldots$

Définition 4 *Un graphe non-orienté de $\mathcal{P}$ pour une base de données DB est une paire ordonnée* G=(V, E) *composé d'un ensemble* V *de nœuds et d'un ensemble* E *d'arêtes, où :*

i. Chaque nœud $v_i \in V$ correspond à un prédicat de $\mathcal{P}$ ou à sa négation, $|V| = 2 \times |\mathcal{P}|$

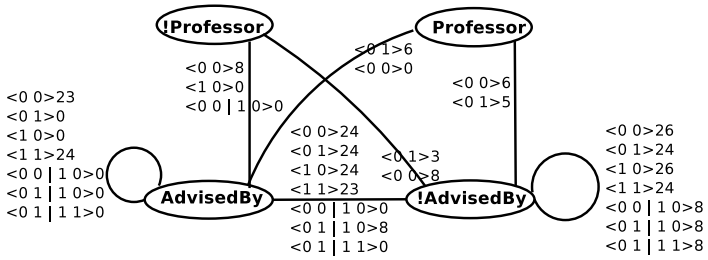

FIG. 1 – *Exemple de graphe des prédicats.*

Algorithme 1 : GSLP(DB, MLN, maxLength)

Créer le graphe des prédicats *G = (V, E)*;
Ajouter toutes les clauses unitaires au MLN et apprendre leur poids;
for *longueur=2* **to** *maxLength* **do**
  $CC \longleftarrow CreerClausesCandidates(DB, G, CC)$; `// Algorithm 2`
  $AjouterClausesCandidatesMlnElaguer(CC, MLN)$;
end
$Retourner(MLN)$;

ii. S'il existe un lien $link(p_i, p_j)$ *entre deux prédicats* p_i *et* p_j*, alors il existe une arête entre les nœuds* v_i *et* v_j *à laquelle sont associées deux étiquettes :* l-label *contenant le lien(par exemple <0 0>) et* l-cardinalité *donnant le nombre d'instantiations vraies de la formule binaire satisfaisant ce lien.*

iii. A chaque nœud $v_i \in V$*, est associé un poids défini par :* $v_i.weight = k * \frac{\sum_{p=1}^{q} t_{ip}}{q}$*, où* q *est le nombre d'arêtes incidentes à* v_i*,* t_{ip} *est la l-cardinalité de la* p*-ième arête et* k *est un coefficient réel d'ajustement.*

Exemple 2 *La figure 1 donne le graphe de prédicats pour le langage donné dans l'exemple 1. Nous avons un lien* link(Professor, AdvisedBy) = $\langle 0, 1 \rangle$*. Si nous supposons que nous avons 6 instantiations vraies de la formule correspondante dans la base de données, nous trouvons alors une arête entre* Professor *et* AdvisedBy *avec* l-label = $\langle 0, 1 \rangle$ *et* l-cardinalité = 6*. Inversement, nous avons* link(AdvisedBy, Professor)= $\langle 1, 0 \rangle 6$*. Quand* $k = 1.0$*, le poids du nœud* AdvisedBy *est* $AdvisedBy.weight = 1.0 * (23 + 24)/7 \simeq 6.714$*, si 23 et 24 sont les nombres d'instantiations de la formule correspondant aux liens* $\langle 0\ 0 \rangle$ *et* $\langle 1\ 1 \rangle$ *(et si les autres liens n'ont pas d'instantiations).*

3.2 Structure de l'algorithme

Etant données une base de données DB consistante, un MLN (éventuellement vide) et un nombre entier *maxLength*, spécifiant la longueur maximale des clauses, nous présentons maintenant un algorithme, appelé *GSLP* (Generative Structure Learning based on graph of Predicates) permettant d'apprendre génératívement la structure du MLN.

Les approches développées dans Kok et Domingos (2005), Biba et al. (2008) et Huynh et Mooney (2008) explorent de manière intensive l'espace des clauses produisant beaucoup de clauses inutiles, alors que celles développées dans Mihalkova et Mooney (2007), Kok et Domingos (2009), Dinh et al. (2010) recherchent tous les chemins dans la base de données, nécessitant un temps de calcul important. Nous préférons engendrer des clauses potentiellement intéressantes en nous fondant sur deux observations. D'abord, il nous semble inutile de considérer des clauses connectées qui n'ont pas d'instances dans la base de données. Ensuite, une formule connectée $A_1 \wedge ... \wedge A_n$ est d'autant plus utile qu'elle couvre un plus grand nombre d'atomes dans la base de données. Autrement dit, des clauses intéressantes sont surtout celles qui sont assez fréquentes en termes du nombre d'instantiations. On retrouve ainsi la notion traditionnelle de couverture en ILP. Il est évident que si une formule connectée est fréquente, donc ses sous-formules connectées sont au moins aussi fréquentes qu'elle (des remarques semblables sont utilisés sur beaucoup de stratégies connues d'exploration). Nous proposons ainsi de produire les clauses candidates en commençant par les clauses binaires (formées de deux atomes connectés) ; les clauses de 3-atomes peuvent alors être formées à partir des clauses binaires, puis celles de 4-atomes et ainsi de suite. Le problème à résoudre est alors celui de la recherche d'un tel ensemble de clauses binaires connectées $A_i \wedge A_j$ à partir desquelles on peut étendre progressivement l'ensemble de clauses candidates.

Notre algorithme de génération des clauses repose sur le graphe des prédicats. Pour chaque prédicat, nous créons deux nœuds (correspondant au prédicat et à sa négation), et le graphe contient donc seulement $2 \times m$ nœuds, où *m* est le nombre de prédicats du langage. Le nombre de nœuds incidents à un nœud dépend du nombre d'arguments (du prédicat correspondant) et du nombre de relations entre les types des arguments des prédicats. Il reste en général raisonnable. Nous calculons seulement la l-cardinalité des liens, réduisant ainsi le calcul du nombre d'instantiations à des formules binaires. La recherche de chemins dans le graphe est alors beaucoup plus efficace que la recherche de chemins dans un ensemble d'atomes, particulièrement quand on a un grand nombre de prédicats et quand beaucoup d'arguments sont partagés dans les atomes dans le base de données.

Notons cependant que pour chaque chemin nous connaissons seulement les prédicats qui apparaissent dans la clause et des informations sur des variables qui doivent être unifiées. Beaucoup de clauses (avec des variables différentes) peuvent être construites. Nous utilisons une technique heuristique (expliquée dans la sous-section 3.3) pour variabiliser un chemin afin d'obtenir *une unique clause*.

À chaque étape, ayant un ensemble de candidats de même longueur, notre algorithme ajoute chaque clause au MLN courant et apprend les poids du MLN correspondant. La pseudo-log vraisemblance est alors calculée ; si le poids de cette clause est plus grand qu'un seuil *minWeight* et s'il y a amélioration de cette mesure, le MLN candidat deviendra le MLN appris courant. Parce que l'ajout d'une clause peut influencer les poids des clauses ajoutées précédentes, une fois que toutes les clauses candidates ont été considérées, nous tentons d'élaguer des clauses du MLN : une clause avec un poids inférieur au seuil *minWeight* sera enlevée du MLN si son retrait permet d'augmenter la pseudo-log vraisemblance.

Dans un MLN, le poids associé à une formule (une clause) reflète l'importance de cette formule : plus le poids est élevé, plus grande est la différence de la log-probabilité entre un monde qui satisfait la formule et un monde ne le satisfaisant pas, toutes choses par ailleurs étant égales. Concernant des clauses unitaires (contenant un seul prédicat), leur poids, très

Algorithme 2 : CreerClausesCandidates(DB, G, CC)

Initialisation : $SCC = \emptyset$;
foreach *clause* $c \in CC$ **do**
  $SP \longleftarrow ChercherCheminDansGraphe(c, G)$;
  foreach *chemin* $p \in SP$ **do**
    $tc \longleftarrow Variabiliser(p)$;
    Evaluer(tc);
    $SCC \longleftarrow tc$ si (($tc.gain > 0$) et ($tc.weight > minWeight$));
  end
end
$Return(SCC)$;

schématiquement, capture la distribution marginale de ces prédicats alors que les clauses plus longues modélisent les dépendances des prédicats. C'est la raison pour laquelle il est utile d'ajouter tout d'abord toutes les clauses unitaires au MLN. C'est la première étape de notre algorithme et nous appelons ce premier réseau le MLN unitaire.

Nous présentons la structure de GSLP dans l'Algorithme 1. Dans la section suivante, nous décrivons plus en détail la façon de générer un ensemble de clauses candidates pour chaque étape de notre algorithme.

3.3 Créer les clauses candidates

Comme mentionné ci-dessus, nous utilisons le graphe des prédicats pour construire les chemins à variabiliser afin de produire des clauses. Comme beaucoup de systèmes en ILP, nous cherchons un ensemble de clauses candidates, avec un taux de couverture important : nous considérons seulement les arêtes $e(v_i, v_j)$ dont la l-cardinalité est plus grande que le minimum de $v_i.weight$ et $v_j.weight$. Rappelons que le *poids* d'un nœud est proportionnel à la moyenne des l-cardinalités de ses arêtes incidentes et que le coefficient k apparaissant dans le poids est utilisé comme paramètre d'ajustement. De telles arêtes sont ensuite variabilisées afin d' obtenir des clauses connectées binaires. Pour construire des clauses plus longues, nous étendons chaque chemin $p_1, ..., p_k$ (correspondant à une clause candidate) avec une arête incidente à au moins un nœud de l'ensemble $\{p_1, ..., p_k\}$, conduisant ainsi à un chemin $p_1, ..., p_k, p_{k+1}$ et vérifiant la propriété précédente. Ce processus est répété pour toutes les arêtes. Une clause est évaluée en apprenant les poids du MLN composé du MLN unitaire et de cette clause puis en calculant la mesure (WPLL) associée. Si la mesure de ce MLN est plus grande que celle du MLN unitaire et si le poids de la clause est plus grand que le seuil *minWeight*, on considère cette clause comme un clause candidate. L'algorithme 2 donne les différentes étapes pour construire les clauses candidates.

Considérons maintenant le processus de variabilisation. Généralement, on variabilise une clause en remplaçant les constantes par des variables. Dans notre cas, nous ne disposons que d'un chemin dans le graphe de prédicats, i.e., une liste d'arêtes contenant pour seules informations les prédicats et les positions des arguments partagés. Il n'est pas envisageable de construire toutes les clauses satisfaisant ce chemin pour des raisons de temps de calcul : plus on produit de clauses et plus il faut de temps pour les évaluer et apprendre le MLN final.

Nous variabilisons heuristiquement un chemin pour produire une unique clause connectée. Afin de réduire le nombre de variables distinctes dans une clause, nous traitons les prédicats par ordre de fréquence décroissante dans le chemin. Pour chaque prédicat, nous effectuons la variabilisation d'abord pour les variables partagées, ensuite pour les variables non partagées. Un argument d'un prédicat sera variabilisé par une nouvelle variable si sa position n'apparaît pas dans le l-label d'un lien du chemin impliquant ce prédicat.

Exemple 3 *Le chemin* $!advisedBy\ \langle 0,1\rangle\ !advisedBy$ *et* $!advisedBy\ \langle 0,0\rangle\ student$ *conduit à la clause* $!advisedBy(A,B) \vee !advisedBy(C,A) \vee student(A)$. *Par convention le lien entre* $student$ *et* $advisedBy$ *est appliqué au premier* $advisedBy$.

A partir d'un chemin $p_1, ..., p_k$ correspondant à une clause c_k, nous cherchons un chemin $p_1, ..., p_k, p_{k+1}$, puis le variabilisons pour créer une clause c_{k+1}. Si c_k est une sous-clause de c_{k+1} elle est ignorée.

4 Expérimentations

4.1 Bases de test

Nous avons utilisé trois jeux de données disponibles en ligne [1] : IMDB, Uw-cse et Cora. IMDB décrit une base de films et comporte 1540 atomes construits à partir de 10 prédicats et 316 constantes. Uw-cse décrit un département académique et comporte 2673 atomes construits à partir de 15 prédicats et 1323 constantes. Cora est la plus grande de ces trois bases. Elle décrit une collection de citations de publications en informatique. Elle est bâtie sur 10 prédicats et 3079 constantes et comporte 70367 atomes. A la différence des deux bases précédents, une partie des atomes sont faux (IMDB et Uw-cse ne fournissent que des atomes vrais).

4.2 Systèmes et méthodologie

Nous comparons *GSLP* à deux méthodes de l'état de l'art : LSM et HGSM :

– LSM (Learning using Structural Motifs, Kok et Domingos (2010)) repose sur l'observation que la base de données relationnelle contient des schémas qui sont des variations de mêmes motifs structurels. Un mécanisme de *parcours aléatoire* est utilisé pour identifier des objets fortement connectés, puis les regrouper, avec les relations associées, au sein d'un motif. La recherche de clauses se fait ensuite sur la base des motifs ainsi créés. Pour apprendre le MLN final, LSM ajoute toutes les clauses au MLN, apprend leurs poids optimaux et supprime celles dont le poids est inférieur à un seuil donné θ_{wt}.

– HGSM (Heuristic Generative Structure learning for MLNs, Dinh et al. (2010)) suit une approche bottom-up. Sur la base des exemples fournis, HGSM bâtit un ensemble de tables booléennes donnant une vue synthétique des atomes connectés. Chaque colonne d'une telle table correspond à un littéral avec variable, et chaque ligne indique un ensemble de connections observées (ou non), entre ces littéraux, à partir d'un atome clos donné. A partir de ces tables, les littéraux dépendants sont identifiés grâce à l'algorithme GSMN (Grow-Shrink Markov Networks, Bromberg et al. (2009)), et sont regroupés pour former des clauses candidates. Les meilleures clauses candidates sont intégrées au MLN.

1. http ://alchemy.cs.washington.edu

	IMDB			UW-CSE			CORA		
	CLL	AUC	Tps (h)	CLL	AUC	Tps (h)	CLL	AUC	Tps (h)
GSLP	**-0.160 ± 0.03**	**0.789 ± 0.06**	1.59	**-0.053± 0.06**	**0.502± 0.07**	4.62	**-0.059 ± 0.05**	**0.886 ± 0.07**	7.82
HGSM	-0.183 ± 0.03	0.692 ± 0.07	3.06	-0.103± 0.04	0.311± 0.02	8.68	-0.087 ± 0.05	0.762 ± 0.06	64.15
LSM	-0.191 ± 0.02	0.714 ± 0.06	**1.23**	-0.068± 0.07	0.426± 0.07	**2.88**	-0.065 ± 0.02	0.803 ± 0.08	**6.05**

TAB. 1 – *CLL et AUC pour les bases IMDB, Uw-cse et Cora. Les valeurs présentées correspondent à une moyenne sur l'ensemble des prédicats.*

Les trois algorithmes sont implémentés au dessus de la librairie Alchemy, développée par l'équipe de P. Domingos. Notons l'existence d'un troisième algorithme récent, nommé MBN (Khosravi et al. (2010)), et permettant l'apprentissage de la structure d'un MLN dans un cadre génératif. Cependant, les principes sous-jacents à cet algorithme ne permettent pas la mise en place de la validation croisée. Nous envisageons d'étudier ultérieurement les adaptations possibles qui permettraient sa mise en œuvre.

Chaque jeu de données est découpé en 5 parties pour effectuer une validation croisée. Pour chaque algorithme et chaque jeu de données, nous avons mesuré, pour chaque prédicat, la CLL (log-vraisemblance conditionnelle) et l'aire sous la courbe (AUC) de précision-rappel, ces deux mesures constituant le standard utilisé dans la plupart des publications du domaine. La CLL mesure directement la qualité des probabilités produites. Pour un prédicat donné, elle correspond à la moyenne des valeurs observées pour tous les atomes clos correspondant. L'AUC montre la capacité de l'algorithme à inférer avec peu de positifs dans le jeu de données. Pour la calculer, pour un prédicat donné, on fait varier le seuil de CLL au delà duquel un atome est considéré vrai, et on note les différents couples de précision et de rappel observés. La valeur des atomes cibles a été inférée avec l'algorithme MC-SAT (Poon et Domingos (2006)). Enfin, nous avons utilisé le code fourni par (Davis et Goadrich (2006)) pour calculer l'AUC.

Les paramètres pour LSM et HGSM sont ceux utilisés respectivement par Kok et Domingos (2010) et Dinh et al. (2010). Le nombre maximal de littéraux par clause est fixé à 5 pour tous les algorithmes. Afin de limiter l'espace de recherche dans GSLP, le coefficient d'ajustement k est fixé à 1, limitant ainsi le nombre de nœuds du graphe exploré. Nous limitons le nombre de prédicats semblables dans chaque clause à 3 afin de conserver des temps de calcul raisonnables lors de l'apprentissage des poids. Les test ont été conduits sur une machine disposant d'un processeur Dual-core AMD 2.4 GHz et 4GB de mémoire.

4.3 Résultats

Nous avons effectué une validation croisée en apprenant sur quatre parties et en inférant, pour chaque littéral clos (atome vrai ou faux suivant le signe du littéral) de la cinquième partie, sa probabilité d'être vrai. Ce processus est itéré 5 fois, en changeant la partie utilisée pour le test. A partir de ces valeurs, nous avons pu calculer pour chaque prédicat, la CLL et l'AUC précision-rappel. Nous avons ensuite moyenné ces deux informations non seulement sur les différentes parties, mais aussi sur l'ensemble des prédicats de chaque base (cette pratique étant la norme dans les publications du domaine).

Le tableau 1 présente la CLL et l'AUC, ainsi que le temps d'exécution (apprentissage et inférence) pour les trois algorithmes. On peut noter que, concernant LSM, nos résultats diffèrent légèrement de ceux de Kok et Domingos (2010), bien que nous ayons utilisé les mêmes paramètres. Cela s'explique par trois raisons. Tout d'abord, concernant IMDB et Cora, les auteurs de LSM ont omis quelques prédicats d'égalité tandis que nous les avons gardé. Ensuite, nous avons autorisé des clauses comportant jusqu'à 5 prédicats au lieu de 4. Enfin, comme nous nous plaçons dans un cadre génératif, les poids dans des formules ont été appris une seule fois pour une optimisation globale du réseau, alors que Kok et Domingos (2010) réapprennent les poids pour chaque prédicat, afin d'optimiser le réseau de manière discriminante.

Nous pouvons observer que pour ces trois jeux de données, GSLP surpasse LSM et HGSM en termes de CLL et d'AUC. Cela se vérifie non seulement pour les valeurs moyennes présentées dans cette table, mais aussi sur des valeurs moyennes par prédicat dans chaque fragment. GSLP est plus rapide que HGSM, particulièrement pour Cora. Il se concentre en effet sur les seules *bonnes* arêtes (correspondant à des connexions fréquentes) et crée seulement une clause pour chaque chemin. Son espace de recherche est donc bien moins vaste que celui de HGSM. En revanche, GSLP est un peu plus lent que LSM. Ce dernier repose sur une vue synthétique de chaque jeu de données appelée hyper-graphe généralisé (lifted hypergraph). Ces hyper-graphes généralisés sont calculés une seule fois lors de l'apprentissage et sont utilisés ensuite pendant l'étape de validation croisée, qui s'en trouve considérablement accélérée.

Le tableau 2 détaille les valeurs moyennes de CLL et d'AUC pour chaque prédicat de Uw-cse, pour les algorithmes GSLP et LSM. On peut constater que GSLP se comporte mieux que LSM pour la plupart des prédicats. En revanche, LSM produit des résultats remarquables pour les prédicats comme *SameCourse*, *SamePerson*, *SameProject*. Notons qu'il s'agit de prédicats très particuliers, dont les atomes sont vrais si et seulement si leurs deux arguments sont identiques.

Ce processus d'évaluation a été l'occasion, pour nous, de mettre en avant divers points sur lesquels nous envisageons de nous pencher à l'avenir :

- La valeur du coefficient d'ajustement *k* a un impact clair sur le MLN calculé. Avec *k* petit, on considère plus d'arêtes et donc plus de clauses. Le MLN généré est en général plus grand et présente un meilleur score global (WPLL). Néanmoins, cet accroissement de la taille rend l'inférence très coûteuse, voire infaisable avec les algorithmes disponibles. L'équilibre entre performance globale (en termes de WPLL) et temps de calcul constituera un point intéressant à étudier.
- La structure apprise est globalement satisfaisante, mais les inférences qui en résulte sont nettement meilleurs pour certains prédicats que pour d'autres. La compréhension de cette variabilité et sa réduction constituent une perspective intéressante.
- On retrouve fréquement des clauses candidates qui diffèrent seulement par leur variabilisation. Par exemple, $P(X,Y) \vee Q(X,Y)$ et $P(X,Y) \vee Q(Y,X)$ différent seulement sur la position de *X* et *Y*. Un traitement approprié de telles configurations devrait permettre de réduire les temps d'exécution.

5 Conclusion

Nous avons présenté GSLP, un algorithme d'apprentissage de la structure d'un Réseau Logique de Markov dans un cadre génératif. Cet algorithme crée un graphe des prédicats et

	GSLP		LSM	
PREDICATES	CLL	AUC	CLL	AUC
ADVISEDBY	**-0.015**	**0.228**	-0.020	0.010
COURSELEVEL	**-0.311**	**0.801**	-0.321	0,581
HASPOSITION	**-0,057**	**0,821**	**-0,057**	0,568
INPHASE	**-0,092**	**0,449**	-0,160	0,170
PROFESSOR	**-0,069**	0,965	-0,084	**1,000**
PROJECTMEMBER	**-0,001**	**0,001**	**-0,001**	0,0005
PUBLICATION	**-0,078**	**0,234**	-0,130	0,037
SAMECOURSE	-0,009	0,921	**-0,002**	**1,000**
SAMEPERSON	-0,010	0,922	**-0,002**	**1,000**
SAMEPROJECT	-0,005	0,952	**-0,001**	**1,000**
STUDENT	**-0,066**	**0,987**	-0,141	0,961
TA	**-0,008**	**0,025**	**-0,008**	0,002
TEMPADVISEDBY	**-0,008**	**0,019**	**-0,008**	0,006
YEARSINPROGRAM	**-0,004**	**0,187**	-0,008	0,051
TAUGHTBY	**-0,059**	**0,014**	-0,078	0,004
MOYENNE	**-0,053**	**0,502**	-0,068	0,426

TAB. 2 – *Résultats par prédicat pour Uw-cse.*

utilise une technique heuristique de variabilisation pour produire des clauses candidates. Les expériences menées à ce jour tendent à montrer sa supériorité par rapport à l'état de l'art. Elles nous ont également permis d'identifier plusieurs directions pour nos recherches ultérieures. Nous envisageons maintenant d'appliquer GSLP à des domaines plus grands et plus riches, et notamment des domaines comportant plus de prédicats afin d'exploiter les avantages de notre approche. Nous étudions également les possiblités de le comparer à d'autres algorithmes récents tel MBN.

Références

Biba, M., S. Ferilli, et F. Esposito (2008). Structure learning of markov logic networks through iterated local search. In *ECAI '08*, pp. 361–365. IOS Press.

Bromberg, F., D. Margaritis, et V. Honavar (2009). Efficient markov network structure discovery using independence tests. *J. Artif. Int. Res.*, 449–484.

Davis, J. et M. Goadrich (2006). The relationship between precision-recall and roc curves. In *ICML '06*, pp. 233–240. ACM.

Dehaspe, L. (1997). Maximum entropy modeling with clausal constraints. In *ILP '97*, pp. 109–124. Springer-Verlag.

Dinh, Q. T., M. Exbrayat, et C. Vrain (2010). Generative structure learning for markov logic networks. In *STAIRS '10*. IOS Press.

Friedman, N., L. Getoor, D. Koller, et A. Pfeffer (1999). Learning probabilistic relational models. pp. 1300–1309.

Getoor, L. et B. Taskar (2007). *Introduction to Statistical Relational Learning*. The MIT Press.

Huynh, T. N. et R. J. Mooney (2008). Discriminative structure and parameter learning for markov logic networks. In *ICML '08*, pp. 416–423. ACM.

Kersting, K. et L. De Raedt (2001). Bayesian logic programs. Technical report.

Khosravi, H., O. Schulte, T. Man, X. Xu, et B. Bina (2010). Structure learning for markov logic networks with many descriptive attributes. In *ICML-10*.

Kok, S. et P. Domingos (2005). Learning the structure of markov logic networks. In *ICML '05*, pp. 441–448. ACM.

Kok, S. et P. Domingos (2009). Learning markov logic network structure via hypergraph lifting. In *ICML '09*, pp. 505–512. ACM.

Kok, S. et P. Domingos (2010). Learning markov logic networks using structural motifs. In *ICML-10*, pp. 551–558. Omnipress.

Mihalkova, L. et R. J. Mooney (2007). Bottom-up learning of markov logic network structure. In *ICML '07*, pp. 625–632. ACM.

Muggleton, S. (1996). Stochastic logic programs. In I. Press (Ed.), *Advances in Inductive Logic Programming*, pp. 254–264.

Neville, J. et D. Jensen (2004). Dependency networks for relational data. In *ICDM '04*, pp. 170–177. IEEE Computer Society.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc.

Poon, H. et P. Domingos (2006). Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI'06*, pp. 458–463. AAAI Press.

Richardson, M. et P. Domingos (2006). Markov logic networks. *Mach. Learn.*, 107–136.

Sato, T. et Y. Kameya (1997). Prism : a language for symbolic-statistical modeling. In *IJCAI'97*, pp. 1330–1335.

Sha, F. et F. Pereira (2003). Shallow parsing with conditional random fields. In *NAACL '03*, pp. 134–141. Association for Computational Linguistics.

Summary

Markov Logic Networks combine Markov Networks and first-order logic by attaching weights to first-order formulas and viewing them as templates for features of Markov Networks. Learning a MLN can be decomposed into structure and weights learning. In this paper we present a new algorithm to learn generatively the structure of Markov Logic Networks. The algorithm uses a graph of predicates, which is built from a set of predicates and a training database. A heuristical variabilization technique is used in order to produce candidate clauses. According to our first experiments, this approach appears to be promising.