Online Structure Learning for Markov Logic Networks

Tuyen N. Huynh and Raymond J. Mooney

Department of Computer Science, University of Texas at Austin,
1616 Guadalupe, Suite 2.408, Austin, Texas 78701, USA
{hntuyen,mooney}@cs.utexas.edu

Abstract. Most existing learning methods for Markov Logic Networks (MLNs) use batch training, which becomes computationally expensive and eventually infeasible for large datasets with thousands of training examples which may not even all fit in main memory. To address this issue, previous work has used online learning to train MLNs. However, they all assume that the model's structure (set of logical clauses) is given, and only learn the model's parameters. However, the input structure is usually incomplete, so it should also be updated. In this work, we present OSL—the first algorithm that performs both online structure and parameter learning for MLNs. Experimental results on two real-world datasets for natural-language field segmentation show that OSL outperforms systems that cannot revise structure.

1 Introduction

Statistical relational learning (SRL) concerns the induction of probabilistic knowledge that supports accurate prediction for multi-relational structured data [9]. Markov Logic Networks (MLNs) [28], sets of weighted clauses in first-order logic, are a recently developed SRL model that generalizes both full first-order logic and Markov networks which makes MLNs an expressive and powerful formalism. MLNs have also been successfully applied to a variety of real-world problems [4].

However, all existing methods for learning the structure (i.e. logical clauses) of an MLN [13,21,1,14,15] are batch algorithms that are effectively designed for training data with relatively few *mega-examples* [20]. A mega-example is a large set of connected facts, and mega-examples are disconnected and independent from each other. For instance, in WebKB [30], there are four mega-examples, each of which contains data about a particular university's computer-science department's web pages of professors, students, courses, research projects and the hyperlinks between them. However, there are many real-world problems with a different character — involving data with thousands of smaller structured examples. For example, a standard dataset for semantic role labeling consists of $90,750$ training examples where each example is a verb and all of its semantic arguments in a sentence [2]. In addition, most existing weight learning methods for MLNs employ batch training where the learner must repeatedly run inference over all training examples in each iteration, which becomes computationally expensive for datasets with thousands of training examples. To address this issue, previous work has applied online learning to set MLN weights [29,22,11]; however, to the best of our knowledge, there is no existing online *structure* learning algorithm for MLNs.

In this work, we present the first online structure learner for MLNs, called OSL, which updates both the structure and parameters. At each step, based on the model's incorrect predictions, OSL finds new clauses that fix these errors, then uses an adaptive subgradient method with l_1-regularization to update weights for both old and new clauses. Experimental results on natural language field segmentation on two real-world datasets show that OSL is able to find useful new clauses that improve the predictive accuracies of well-developed MLNs.

The remainder of the paper is organized as follows. Section 2 provides some background on MLNs and the field segmentation task. Section 3 presents our proposed algorithm. Section 4 reports the experimental evaluation on two real-world datasets. Section 5 and 6 discuss related and future work, respectively. Section 7 presents our conclusions.

2 Background

2.1 Terminology and Notation

There are four types of symbols in first-order logic: constants, variables, predicates, and functions [8]. Here, we assume that domains do not contain functions. Constants represent entities in the domain and can have types. Variables range over entities in the domain. Predicates represent properties and relations in the domain and each has a fixed number of arguments. Each argument can have a type specifying the type of constant that can fill it. We denote constants by strings starting with upper-case letters, and variables by strings starting with lower-case letters. A term is a constant or a variable. An atom is a predicate applied to terms. A ground atom is an atom all of whose arguments are constants. A positive literal is an atom, and a negative literal is a negated atom. A (possible) world is an assignment of truth values to all ground atoms in a domain. A formula consists of literals connected by logical connectives (i.e. $\vee$ and $\wedge$). A formula in clausal form, also called a clause, is a disjunction of literals.

For mathematical terms, we use lower case letters (e.g. η, λ) to denote scalars, bold face letters (e.g. $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{w}$) to denote vectors, and upper case letters (e.g. $\mathcal{W}$, $\mathcal{X}$) to denote sets. The inner product between vectors $\boldsymbol{w}$ and $\boldsymbol{x}$ is denoted by $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$. The $[a]_+$ notation denotes a truncated function at 0, i.e. $[a]_+ = max(a, 0)$.

2.2 MLNs

An MLN consists of a set of weighted first-order clauses. It provides a way of softening first-order logic by making situations in which not all clauses are satisfied less likely but not impossible [28]. More formally, let $\mathcal{X}$ be the set of all ground atoms, $\mathcal{C}$ be the set of all clauses in the MLN, w_i be the weight associated with clause $c_i \in \mathcal{C}$, $\mathcal{G}_{c_i}$ be the set of all possible groundings of clause c_i. Then the probability of a possible world $\mathbf{x}$ is defined as [28]:

$$P(\mathcal{X} = \mathbf{x}) = \frac{1}{\mathcal{Z}} \exp\left(\sum_{c_i \in \mathcal{C}} w_i \sum_{g \in \mathcal{G}_{c_i}} g(\mathbf{x})\right) = \frac{1}{\mathcal{Z}} \exp\left(\sum_{c_i \in \mathcal{C}} w_i n_i(\mathbf{x})\right)$$

where $g(\mathbf{x})$ is 1 if g is satisfied and 0 otherwise, $n_i(\mathbf{x}) = \sum\limits_{g \in \mathcal{G}_{c_i}} g(\mathbf{x})$ is the number of true groundings of c_i in the possible world $\mathbf{x}$, and $\mathcal{Z} = \sum\limits_{\mathbf{x} \in \mathcal{X}} \exp\left(\sum_{c_i \in \mathcal{C}} w_i n_i(\mathbf{x})\right)$ is the normalization constant.

In many applications, we know a priori which predicates provide evidence and which are used in queries, and the goal is to correctly predict query atoms given evidence atoms. If we partition the ground atoms in the domain into a set of evidence atoms $\mathcal{X}$ and a set of query atoms $\mathcal{Y}$, then the conditional probability of $\mathbf{y}$ given $\mathbf{x}$ is:

$$P(\mathcal{Y} = \mathbf{y} | \mathcal{X} = \mathbf{x}) = \frac{1}{\mathcal{Z}_{\mathbf{x}}} \exp\left(\sum_{c_i \in \mathcal{C}} w_i n_i(\mathbf{x}, \mathbf{y})\right)$$

where $n_i(\mathbf{x}, \mathbf{y})$ is the number of true groundings of c_i in the possible world ($\mathbf{x}$,$\mathbf{y}$) and $\mathcal{Z}_{\mathbf{x}} = \sum\limits_{\mathbf{y} \in \mathcal{Y}} \exp\left(\sum_{c_i \in \mathcal{C}} w_i n_i(\mathbf{x}, \mathbf{y})\right)$ is the normalization constant.

2.3 Natural Language Field Segmentation

In this work, we look at an information extraction task, called field segmentation [10], a sample real-world problem where the data contains many, relatively small, structured examples in the form of short documents. A document is represented as a sequence of tokens, and the goal is to segment the text into fields, i.e. label each token in the document with a particular field. For example, when segmenting advertisements for apartment rentals [10], the goal is to segment each ad into fields such as *Features*, *Neighborhood*, *Rent*, *Contact*, etc.. Below are descriptions of some key predicates:

– $Token(string, position, docID)$: the token at a particular position in a document such as $Token(Entirely, P4, Ad001)$
– $InField(field, position, docID)$: the field label of the token at a particular position in a document such as $InField(Features, P4, Ad001)$
– $Next(position, position)$: the first position is next to the second, such as $Next(P01, P02)$

$InField$ is a target predicate, the rest are evidence predicates.

3 Online Max-Margin Structure and Parameter Learning

In this section, we describe our new online max-margin learning algorithm, OSL, for updating both the structure and parameters of an MLN. In each step, whenever the model makes wrong predictions on a given example, based on the wrongly predicted atoms the algorithm finds new clauses that discriminate the ground-truth possible world from the predicted one, then uses an adaptive subgradient method with l_1-regularization to update weights for both old and new clauses. Algorithm 1 gives the pseudocode for OSL. Lines 3 to 20 are pseudocode for structure learning and lines 21 to 35 are pseudocode for parameter learning.

3.1 Online Max-Margin Structure Learning with Mode-Guided Relational Pathfinding

Most existing structure learning algorithms for MLNs only consider ground-truth possible worlds and search for clauses that improve the likelihood of those possible worlds. However, these approaches may spend a lot of time exploring unhelpful clauses that are true in most possible worlds. Therefore, instead of only considering ground-truth possible worlds, OSL also takes into account the predicted possible worlds, i.e. the most probable possible worlds predicted by the current model. At each step, if the predicted possible world is different from the ground-truth one, then OSL focuses on where the two possible worlds differ and searches for clauses that differentiate them. This is related to the idea of using implicit negative examples in inductive logic programming (ILP) [34]. In this case, each ground-truth possible world plays the role of a positive example in traditional ILP. Making a closed world assumption [8], any possible world that differs from the ground-truth possible world is incorrect and can be considered as a negative example (the predicted possible world in this case). In addition, this follows the max-margin training criterion which focuses on discriminating the true label (the ground-truth possible world) from the most probable incorrect one (the predicted possible world) [33].

At each time step t, OSL receives an example $\mathbf{x}_t$, produces the predicted label $\mathbf{y}_t^P = \arg\max_{\mathbf{y}\in\mathbf{Y}}\langle \mathbf{w}_{\mathcal{C}}, \boldsymbol{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y})\rangle$, then receives the true label $\mathbf{y}_t$. Given $\mathbf{y}_t$ and $\mathbf{y}_t^P$, in order to find clauses that separate $\mathbf{y}_t$ from $\mathbf{y}_t^P$, OSL first finds atoms that are in $\mathbf{y}_t$ but not in $\mathbf{y}_t^P$, $\Delta y_t = \mathbf{y}_t \setminus \mathbf{y}_t^P$. Then OSL searches the ground-truth possible world $(\mathbf{x}_t, \mathbf{y}_t)$ for clauses that are specific to the true ground atoms in Δy_t.

A simple way to find useful clauses specific to a set of atoms is to use relational pathfinding [27], which considers a relational example as a hypergraph with constants as nodes and true ground atoms as hyperedges connecting the nodes that are its arguments, and searches in the hypergraph for paths that connect the arguments of an input literal. A path of hyperedges corresponds to a conjunction of true ground atoms connected by their arguments and can be generalized into a first-order clause by variabilizing their arguments. Starting from a given atom, relational pathfinding searches for all paths connecting the arguments of the given atom. Therefore, relational pathfinding may be very slow or even intractable when there are a large (exponential) number of paths. To speed up relational pathfinding, we use mode declarations [23] to constrain the search for paths. As defined in [23], mode declarations are a form of language bias to constrain the search for definite clauses. Since our goal is to use mode declarations for constraining the search space of paths, we introduce a new mode declaration: $modep(r, p)$ for paths. It has two components: a recall number r which is a positive integer, and an atom p whose arguments are place-makers. A place-maker is either '+' (input), '-' (output), or '.' (don't explore). The recall number r limits the number of appearances of the predicate p in a path to r. The place-maker restricts the search of relational pathfinding. Only paths connecting 'input' or 'output' nodes will be considered. A ground atom can only added to a path if one of its arguments has previously appeared as 'input' or 'output' arguments in the path and all of its 'input' arguments are 'output' arguments of previous atoms. Here are some examples of mode declarations for paths:

$$modep(2, Token(.,+,.)) \quad modep(1, Next(-,-)) \quad modep(2, InField(.,-,.))$$

The above mode declarations require that a legal path contains at most two ground atoms of each of the predicates $Token$ and $InField$ and one ground atom of the predicate $Next$. Moreover, the second argument of $Token$ is an 'input' argument; the second argument of $InField$ and all arguments of $Next$ are 'output' arguments. Note that, in this case, all 'input' and 'output' arguments are of type position. These 'input' and 'output' modes constrain that the position constants in atoms of $Token$ must appeared in some previous atoms of $Next$ or $InField$ in a path. From the graphical model perspective, these mode declarations restrict the search space to linear chain CRFs [31] since they constrain the search to paths connecting ground atoms of two consecutive tokens. It is easy to modify the mode declarations to search for more complicated structure. For example, if we increase the recall number of $Next$ to 2 and the recall number of $InField$ to 3, then the search space is constrained to second-order CRFs since they constrain the searches to paths connecting ground atoms of three consecutive tokens. If we add a new mode declaration $modep(1, LessThan(-,-))$ for the predicate $LessThan$, then the search space becomes skip-chain CRFs [31]. Algorithm 2 presents the pseudocode for efficiently constructing a hypergraph based on mode declarations by only constructing the hypergraph corresponding to input and output nodes. Algorithm 3 gives the pseudocode for mode-guided relational pathfinding, $ModeGuidedFindPaths$, on the constructed hypergraph. It is an extension of a variant of relational pathfinding presented in [14].[1] Starting from each true ground atom $r(c_1, ..., c_r) \in \Delta y_t$, it recursively adds to the path ground atoms or hyperedges that satisfy the mode declarations. Its search terminates when the path reaches a specified maximum length or when no new hyperedge can be added. The algorithm stores all the paths encountered during the search. Below is a sample path found by the algorithm:

[1] In this variant, a path does not need to connect arguments of the input atom. The only requirement is that any two consecutive atoms in a path must share at least one argument.

$$\{InField(Size, P29, Ad001), Token(And, P29, Ad001), Next(P29, P30), \\ Token(Spacious, P30, Ad001)\ InField(Size, P30, Ad001)\}$$

A standard way to generalize paths into first-order clauses is to replace each constant c_i in a conjunction with a variable v_i. However, for many tasks such as field segmentation, it is critical to have clauses that are specific to a particular constant. In order to create clauses with constants, we introduce mode declarations for creating clauses: $modec(p)$. This mode declaration has only one component which is an atom p whose arguments are either 'c' (constant) or 'v' (variable). Below are some examples of mode declarations for creating clauses:

$$modec(Token(c, v, v)) \quad modec(Next(v, v)) \quad modec(InField(c, v, v)$$

Based on these mode declarations, OSL variablizes all constants in a conjunction except those are declared as constants. Then OSL converts the conjunction of positive literals to clausal form since this is the form used in Alchemy.[2] In MLNs, a conjunction of positive literals with weight w is equivalent to a clause of negative literals with weight $-w$. Previous work [14,15] found that it is also useful to add other variants of the clause by flipping the signs of some literals in the clause. Currently, we only add one variant—a Horn version of the clause by only flipping the first literal, the one for which the model made a wrong prediction. In summary, for each path, OSL creates two type of clauses: one with all negative literals and one in which only the first literal is positive. For example, from the sample path above, OSL creates the following two clauses:

$$\neg InField(Size, p1, a) \vee \neg Token(And, p1, a) \vee \neg Next(p1, p2) \vee \\ \neg Token(Spacious, p2, a) \vee \neg InField(Size, p2, a)$$

$$InField(Size, p1, a) \vee \neg Token(And, p1, a) \vee \neg Next(p1, p2) \vee \\ \neg Token(Spacious, p2, a) \vee \neg InField(Size, p2, a)$$

Finally, for each new clause c, OSL computes the difference in the number of true groundings of c in the ground-truth possible world $(\mathbf{x}_t, \mathbf{y}_t)$ and the predicted possible world $(\mathbf{x}_t, \mathbf{y}_t^P)$, $\Delta n_c = n_c(\mathbf{x}_t, \mathbf{y}_t) - n_c(\mathbf{x}_t, \mathbf{y}_t^P)$. Then, only clauses whose difference in number of true groundings is greater than or equal to a predefined threshold *minCountDiff* will be added to the existing MLN. The smaller the value of *minCountDiff*, the more clauses will be added to the existing MLN at each step.

3.2 Online Max-Margin l_1-Regularized Weight Learning

The above online structure learner may introduce a lot of new clauses in each step, and some of them may not be useful in the long run. To address this issue, we use l_1-regularization which has a tendency to force parameters to zero

[2] The standard software for MLNs: `alchemy.cs.washington.edu`

Algorithm 1. OSL

```
Input:  $\mathcal{C}$: initial clause set (can be empty)
    $mode$: mode declaration for each predicate
    $maxLen$: maximum number of hyperedges in a path
    $minCountDiff$: minimum number of difference in true groundings for selecting new clauses
    $\lambda, \eta, \delta$: parameters for the $l_1$-regularization adaptive subgradient method
    $\rho(\mathbf{y}, \mathbf{y}')$: label loss function
Note:   Index H maps from each node $\gamma_i$ to set of hyperedges $r(\gamma_1, ..., \gamma_i, ..., \gamma_n)$ containing $\gamma_i$
    $Paths$ is a set of paths, each path is a set of hyperedges
 1: Initialize: $\mathbf{w}_{\mathcal{C}} = 0, \mathbf{g}_{\mathcal{C}} = 0, nc = |\mathcal{C}|$
 2: for $i = 1$ to $T$ do
 3:   Receive an instance $\mathbf{x}_t$
 4:   Predict $\mathbf{y}_t^P = \arg\max_{\mathbf{y} \in \mathbf{Y}} \langle \mathbf{w}_{\mathcal{C}}, \boldsymbol{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}) \rangle$
 5:   Receive the correct target $\mathbf{y}_t$
 6:   Compute $\Delta y_t = \mathbf{y}_t \setminus \mathbf{y}_t^P$
 7:   if $\Delta y_t \neq \emptyset$ then
 8:     $H = CreateHG((\mathbf{x}_t, \mathbf{y}_t), mode)$
 9:     $Paths = \emptyset$
10:     for each true atom $r(c_1, ..., c_r) \in \Delta y_t$ do
11:       $V = \emptyset$
12:       for each $c_i \in \{c_1, ..., c_r\}$ do
13:         if isInputOrOutputVar($c_i$,mode) then
14:           $V = V \cup \{c_i\}$
15:         end if
16:       end for
17:       $ModeGuidedFindPaths(\{r(c_1, ..., c_r)\}, V, H, mode, maxLen, Paths)$
18:     end for
19:   end if
20:   $\mathcal{C}_{new} = CreateClauses(\mathcal{C}, Paths, mode)$
21:   Compute $\Delta\boldsymbol{n}_{\mathcal{C}}, \Delta\boldsymbol{n}_{\mathcal{C}_{new}}$:
22:     $\Delta\boldsymbol{n}_{\mathcal{C}} = \boldsymbol{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}_t) - \boldsymbol{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}_t^P)$
23:     $\Delta\boldsymbol{n}_{\mathcal{C}_{new}} = \boldsymbol{n}_{\mathcal{C}_{new}}(\mathbf{x}_t, \mathbf{y}_t) - \boldsymbol{n}_{\mathcal{C}_{new}}(\mathbf{x}_t, \mathbf{y}_t^P)$
24:   for $i = 1$ to $|\mathcal{C}|$ do
25:     $\mathbf{g}_{\mathcal{C},i} = \mathbf{g}_{\mathcal{C},i} + \Delta\boldsymbol{n}_{\mathcal{C},i} * \Delta\boldsymbol{n}_{\mathcal{C},i}$
26:     $\mathbf{w}_{\mathcal{C},i} = sign\left(\mathbf{w}_{\mathcal{C},i} + \frac{\eta}{\delta+\sqrt{\mathbf{g}_{\mathcal{C},i}}}\Delta\boldsymbol{n}_{\mathcal{C},i}\right)\left[\left|\mathbf{w}_{\mathcal{C},i} + \frac{\eta}{\delta+\sqrt{\mathbf{g}_{\mathcal{C},i}}}\Delta\boldsymbol{n}_{\mathcal{C},i}\right| - \frac{\lambda\eta}{\delta+\sqrt{\mathbf{g}_{\mathcal{C},i}}}\right]_+$
27:   end for
28:   for $i = 1$ to $|\mathcal{C}_{new}|$ do
29:     if $\Delta\boldsymbol{n}_{\mathcal{C}_{new,i}} \geq minCountDiffer$ then
30:       $\mathcal{C} = \mathcal{C} \cup \mathcal{C}_{new,i}$
31:       $nc = nc + 1$
32:       $\mathbf{g}_{\mathcal{C},nc} = \Delta\boldsymbol{n}_{\mathcal{C}_{new,i}} * \Delta\boldsymbol{n}_{\mathcal{C}_{new,i}}$
33:       $\mathbf{w}_{\mathcal{C},nc} = \left[\frac{\eta}{\delta+\sqrt{\mathbf{g}_{\mathcal{C},nc}}}(\Delta\boldsymbol{n}_{\mathcal{C}_{new,i}} - \lambda)\right]_+$
34:     end if
35:   end for
36: end for
```

Algorithm 2. CreateHG(D,mode)

```
Input:  $D$: a relational example
        mode: mode declaration file
 1: for each constant $c$ in $D$ do
 2:    $H[c] = \emptyset$
 3: end for
 4: for each true ground atom $r(c_1, ..., c_r) \in D$ do
 5:    for each constant $c_i \in \{c_1, ..., c_r\}$ do
 6:       if isInputOrOutputVar($c_i$,mode) then
 7:          $H[c_i] = H[c_i] \cup \{r(c_1, ..., c_r)\}$
 8:       end if
 9:    end for
10: end for
11: return $H$
```

Algorithm 3. $ModeGuidedFindPaths(CurrPath, V, H, mode, maxLen, Paths)$

```
 1: if $|CurrPath| < maxLen$ then
 2:    for each constant $c \in V$ do
 3:       for each $r(c_1, ..., c_r) \in H[c]$ do
 4:          if $canBeAdded(r(c_1, ..., c_r), CurrPath, mode)$ == success then
 5:             if $CurrPath \notin Paths$ then
 6:                $CurrPath = CurrPath \cup \{r(c_1, ..., c_r)\}$
 7:                $Paths = Paths \cup \{CurrPath\}$
 8:                $V' = \emptyset$
 9:                for each $c_i \in \{c_1, ..., c_r\}$ do
10:                   if $c_i \notin V$ and isInputOrOutputVar($c_i$,mode) then
11:                      $V = V \cup \{c_i\}$
12:                      $V' = V' \cup \{c_i\}$
13:                   end if
14:                end for
15:                $ModeGuidedFindPaths(CurrPath, V, H, mode, maxLen, Paths)$
16:                $CurrPath = CurrPath \setminus \{r(c_1, ..., c_r)\}$
17:                $V = V \setminus V'$
18:             end if
19:          end if
20:       end for
21:    end for
22: end if
```

by strongly penalizing small terms [18]. We employ a state-of-the-art online l_1-regularization method—ADAGRAD_FB which is a l_1-regularized adaptive subgradient method using composite mirror-descent update [6]. At each time step t, it updates the weight vector as follows:

$$\mathbf{w}_{t+1,i} = sign\left(\mathbf{w}_{t,i} - \frac{\eta}{H_{t,ii}}\mathbf{g}_{t,i}\right)\left[\left|\mathbf{w}_{t,i} - \frac{\eta}{H_{t,ii}}\mathbf{g}_{t,i}\right| - \frac{\lambda\eta}{H_{t,ii}}\right]_+ \quad (1)$$

where λ is the regularization parameter, η is the learning rate, $\mathbf{g}_t$ is the subgradient of the loss function at step t, and $H_{t,ii} = \delta + ||\mathbf{g}_{1:t,i}||_2 = \delta + \sqrt{\sum_{j=1}^{t}(\mathbf{g}_{j,i})^2}$ ($\delta \geq 0$). Note that, ADAGRAD_FB assigns a different step size, $\frac{\eta}{H_{t,ii}}$, for each component of the weight vectors. Thus, besides the weights, ADAGRAD_FB also needs to retain the sum of the squared subgradients of each component.

From the equation 1, we can see that if a clause is not relevant to the current example (i.e. $g_{t,i} = 0$) then ADAGRAD_FB discounts its weight by $\frac{\lambda\eta}{H_{t,ii}}$. Thus, irrelevant clauses will be zeroed out in the long run.

Regarding the loss function, we use the prediction-based loss function l_{PL} [11], a simpler variant of the max-margin loss:

$$l_{PL}(\mathbf{w}_{\mathcal{C}}, (\mathbf{x}_t, \mathbf{y}_t)) = \left[\rho(\mathbf{y}_t, \mathbf{y}_t^P) - \left\langle \mathbf{w}_{\mathcal{C}}, \left(\mathbf{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}_t^P)\right)\right\rangle\right]_+$$

The subgradient of l_{PL} is:

$$\mathbf{g}_{PL} = \mathbf{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}_t^{PL}) - \mathbf{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}_t) = -\left[\mathbf{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{n}_{\mathcal{C}}(\mathbf{x}_t, \mathbf{y}_t^{PL})\right] = -\Delta\mathbf{n}_{\mathcal{C}}$$

Substituting the gradient into equation 1, we obtain the following formulae for updating the weights of old clauses:

$$\mathbf{g}_{\mathcal{C},i} = \mathbf{g}_{\mathcal{C},i} + (\Delta \boldsymbol{n}_{\mathcal{C},i})^2$$

$$\mathbf{w}_{\mathcal{C},i} \leftarrow sign\left(\mathbf{w}_{\mathcal{C},i} + \frac{\eta}{\delta + \sqrt{\mathbf{g}_{\mathcal{C},i}}}\Delta\boldsymbol{n}_{\mathcal{C},i}\right)\left[\left|\mathbf{w}_{\mathcal{C},i} + \frac{\eta}{\delta + \sqrt{\mathbf{g}_{\mathcal{C},i}}}\Delta\boldsymbol{n}_{\mathcal{C},i}\right| - \frac{\lambda\eta}{\delta + \sqrt{\mathbf{g}_{\mathcal{C},i}}}\right]_+$$

For new clauses, the update formulae are simpler since all the previous weights and gradients are zero:

$$\mathbf{g}_{\mathcal{C},nc} = (\Delta\boldsymbol{n}_{\mathcal{C}_{new,i}})^2$$

$$\mathbf{w}_{\mathcal{C},nc} = \left[\frac{\eta}{\delta + \sqrt{\mathbf{g}_{\mathcal{C},nc}}}(\Delta\boldsymbol{n}_{\mathcal{C}_{new,i}} - \lambda)\right]_+$$

Lines 24 – 27 in Algorithm 1 are the pseudocode for updating the weights of existing clauses, and lines 28 – 35 are the pseudocode for selecting and setting weights for new clauses.

4 Experimental Evaluation

In this section, we conduct experiments to answer the following questions:

1. Starting from a given MLN, does OSL find new useful clauses that improve the predictive accuracy?
2. How well does OSL perform when starting from an empty knowledge base?
3. How does OSL compare to LSM, the state-of-the-art batch structure learner for MLNs [15] ?

4.1 Data

We ran experiments on two real world datasets for field segmentation: CiteSeer[17], a bibliographic citation dataset, and Craigslist [10], an advertisements dataset.

The CiteSeer dataset[3] contains 1,563 bibliographic citations. The dataset has four disjoint subsets consisting of citations in four different research areas. The task is to segment each citation into three fields: *Author*, *Title* and *Venue*.

The Craigslist dataset[4] consists of advertisements for apartment rentals posted on Craigslist. There are 8,767 ads in the dataset, but only 302 of them were labeled with 11 fields: *Available*, *Address*, *Contact*, *Features*, *Neighborhood*, *Photos*, *Rent*, *Restrictions*, *Roommates*, *Size*, and Utilities. The labeled ads are divided into 3 disjoint sets: training, development and test set. The number of ads in each set are 102, 100, and 100 respectively. We preprocessed the data using regular expressions to recognize numbers, dates, times, phone numbers, URLs, and email addresses.

4.2 Input MLNs

A standard model for sequence labeling tasks such as field segmentation is a linear-chain CRF [16]. Thus, we employ an initial MLN, named LC_0, which encodes a simple linear-chain CRF that uses only the current word as features:

$$\begin{gathered} Token(+t, p, c) \Rightarrow InField(+f, p, c) \\ Next(p1, p2) \wedge InField(+f1, p1, c) \Rightarrow InField(+f2, p2, c) \\ InField(f1, p, c) \wedge (f1 != f2) \Rightarrow \neg InField(f2, p, c). \end{gathered}$$

The plus notation indicates that the MLN contains an instance of the first clause for each (*token, field*) pair, and an instance of the second clause for each pair of fields. Thus, the first set of rules captures the correlation between tokens and fields, and the second set of rules represents the transitions between fields. The third rule constrains that the token at a position p can be part of at most one field.

For CiteSeer, we also use an existing MLN developed by Poon and Domingos [26], called the isolated segmentation model (ISM).[5] ISM is also a linear chain CRF but includes more features than the simple model above. Like LC_0, ISM also has rules that correlate the current words with field labels. For transition

[3] We used the versions created by Poon and Domingos [26], which can be found at `http://alchemy.cs.washington.edu/papers/poon07`

[4] `http://nlp.stanford.edu/~grenager/data/unsupie.tgz`

[5] `http://alchemy.cs.washington.edu/mlns/ie/ie_base.mln`

rules, ISM only captures transitions within fields and also takes into account punctuation as field boundaries:

$$Next(p1,p2) \wedge \neg HasPunc(p1,c) \wedge InField(+f,p1,c) \Rightarrow InField(+f,p2,c)$$

In addition, ISM also contains rules specific to the citation domain such as "the first two positions of a citation are usually in the author field", "initials tend to appear in either the author or the venue field". Most of those rules are features describing words that appear before or after the current word.

For Craigslist, previous work [10] found that it is only useful to capture the transitions within fields and take into account the field boundaries, so we create a version of ISM for Craigslist by removing clauses that are specific to the citation domain. Thus, the ISM MLN for Craiglist is a revised version of the LC_0 MLN. Therefore, we only ran experiments with ISM on Craigslist.

4.3 Methodology

To answer the questions above, we ran experiments with the following systems:

ADAGRAD_FB-LC_0: Use ADAGRAD_FB to learn weights for the LC_0 MLN.

OSL-M1-LC_0: Starting from the LC_0 MLN, this system runs a slow version of OSL where the parameter *minCountDiff* is set to 1, i.e. all clauses whose number of true groundings in true possible worlds is greater than those in predicted possible worlds will be selected.

OSL-M2-LC_0: Starting from the LC_0 MLN, this system runs a faster version of OSL where the parameter *minCountDiff* is set to 2.

ADAGRAD_FB-ISM: Use ADAGRAD_FB to learn weights for the ISM MLN.

OSL-M1-ISM: Like OSL-M1-LC_0, but starting from the ISM MLN.

OSL-M2-ISM: Like OSL-M2-LC_0, but starting from the ISM MLN.

OSL-M1-Empty: Like OSL-M1-LC_0, but starting from an empty MLN.

OSL-M2-Empty: Like OSL-M2-LC_0, but starting from an empty MLN.

Regarding label loss functions, we use Hamming (HM) loss which is the standard loss function for structured prediction [32,33].

For inference in training and testing, we used the exact MPE inference method based on Integer Linear Programming described by Huynh and Mooney [12]. For all systems, we ran one pass over the training set and used the average weight vector to predict on the test set. For Craigslist, we used the original split for training and test. For CiteSeer, we ran four-fold cross-validation (i.e. leave one topic out). The parameters λ, η, δ of ADAGRAD_FB were set to 0.001,1, and 1 respectively. For OSL, the mode declarations were set to constrain the search space of relational pathfinding to linear chain CRFs in order to make exact inference in training feasible; the maximum path length *maxLen* was set to 4; the parameters λ, η, δ were set to the same values in ADAGRAD_FB. All the

Table 1. Experimental results for CiteSeer

Systems	Avg. F_1	Avg. train. time (min.)	Avg. num. of non-zero clauses
ADAGRAD_FB-LC_0	82.62 ± 2.12	10.40	$2,896$
OSL-M2-LC_0	92.05 ± 2.63	14.16	$2,150$
OSL-M1-LC_0	94.47 ± 2.04	163.17	$9,395$
ADAGRAD_FB-ISM	91.18 ± 3.82	11.20	$1,250$
OSL-M2-ISM	95.51 ± 2.07	12.93	$1,548$
OSL-M1-ISM	96.48 ± 1.72	148.98	$8,476$
OSL-M2-Empty	88.94 ± 3.96	23.18	650
OSL-M1-Empty	94.03 ± 2.62	257.26	$15,212$

Table 2. Experimental results for Craigslist

Systems	F_1	Train. time (min.)	Num. of non-zero clauses
ADAGRAD_FB-ISM	79.57	2.57	$2,447$
OSL-M2-ISM	77.26	3.88	$2,817$
OSL-M1-ISM	81.58	33.63	$9,575$
OSL-M2-Empty	55.28	17.64	$1,311$
OSL-M1-Empty	71.23	75.84	$17,430$

parameters are set based on the performance on the Craigslist development set. We used the same parameter values for CiteSeer.

Like previous work [26], to measure the performance of each system, we used F_1, the harmonic mean of the precision and recall, at the token level.

4.4 Results and Discussion

Table 1 shows the average F_1 with their standard deviations, average training times in minutes, and average number of non-zero clauses for CiteSeer. All results are averaged over the four folds. First, either starting from LC_0 or ISM, OSL is able to find new useful clauses that improve the F_1 scores. For LC_0, comparing to the system that only does weight learning, the fast version of OSL, OSL-M2, increases the average F_1 score by 9.4 points, from 82.62 to 92.05. The slow version of OSL, OSL-M1, further improves the average F_1 score to 94.47. For ISM, even though it is a well-developed MLN, OSL is still able to enhance it. The OSL-M1-ISM achieves the best average F_1 score, 96.48, which is 2 points higher than the current best F_1 score achieved by using a complex joint segmentation model that also uses information from matching multiple citations of the same paper [26]. Overall, this answers question 1 affirmatively. Additionally, the results for OSL-M2-Empty and OSL-M1-Empty shows that OSL also performs well when learning from scratch. OSL-M1 even finds a structure that is more accurate than ISM's. All differences in F_1 between OSL and ADAGRAD_FB are statistically significant according to a paired t-test ($p < 0.05$). Overall, this also answers question 2 affirmatively.

Regarding training time, OSL-M2 takes on average a few more minutes than systems that only do weight learning. However, OSL-M1 takes longer to train since including more new clauses results in longer time for constructing the ground network, running inference, and computing the number of true groundings. The last column of Table 1 shows the average number of non-zero clauses in the final MLNs learned by different systems. These numbers reflect the size of MLNs generated by different systems during training.

Table 2 shows the experimental results for Craigslist. The Craigslist segmentation task is much harder than CiteSeer's due to the huge variance in the context of different ads. As a result, most words only appear once or twice in the training set. Thus the most important rules are those that correlate words with fields and those capturing the regularity that consecutive words are usually in the same field, which are already in ISM. In addition, most rules only appear once in a document. That is why OSL-M2 is not able to find useful clauses, but OSL-M1 is able to find some useful clauses that improve the F_1 score of ISM from 79.57 to 81.58. OSL also gives some promising results when starting from an empty MLN.

To answer question 3, we ran LSM on CiteSeer and Craigslist but the MLNs returned by LSM result in huge ground networks that made weight learning infeasible even using online weight learning. The problem is that these natural language problems have a huge vocabulary of words. Thus, failing to restrict clauses to specific words results in a blow-up in the size of the ground network. However, LSM is currently not able to learn clauses with constants. It is unclear whether it is feasible to alter LSM to efficiently learn clauses with constants since such constants may need to be considered individually which dramatically increases the search space. This problem also holds for other existing MLN structure learners [13,21,1,14].

Below are some sample useful clauses found by OSL-M2-ISM on CiteSeer:

– If the current token is in the *Title* field and it is followed by a period then it is likely that the next token is in the *Venue* field.

$$InField(Ftitle, p1, c) \wedge FollowBy(p1, TPERIOD, c) \wedge Next(p1, p2) \Rightarrow InField(Fvenue, p2, c)$$

– If the next token is 'in' and it is in the *Venue* field, then the current token is likely in the *Title* field

$$Next(p1, p2) \wedge Token(Tin, p2, c) \wedge InField(Fvenue, p2, c) \Rightarrow InField(Ftitle, p1, c)$$

When starting from an empty knowledge base, OSL-M2 is able to discover the regularity that consecutive words are usually in the same field:

$$Next(p1, p2) \wedge InField(Fauthor, p1, c) \Rightarrow InField(Fauthor, p2, c)$$
$$Next(p1, p2) \wedge InField(Ftitle, p1, c) \Rightarrow InField(Ftitle, p2, c)$$
$$Next(p1, p2) \wedge InField(Fvenue, p1, c) \Rightarrow InField(Fvenue, p2, c)$$

5 Related Work

Our work is related to previous work on online feature selection for Markov Random Fields (MRFs) [25,35]. However, our work differs in two aspects. First, this previous work assumes all the training examples are available at the beginning and only the features are arriving online, while in our work both the examples and features (clauses) are arriving online. Second, in this previous work, all potential features are given upfront, while our approach induces new features from each example. Thus, our work is also related to previous work on feature induction for MRFs [3,19], but these are batch methods.

The idea of combining relational pathfinding with mode declarations has been used in previous work [24,5]. However, how they are used is different. In [24], mode declarations were used to transform a bottom clause into a directed hypergraph where relational pathfinding was used to find paths. Similarly, in [5], mode declarations were used to validate paths obtained from bottom clauses. Here, mode declarations are first used to reduce the search space to paths that contain 'input' and 'output' nodes. Then they are used to test whether an hyperedge can be added to an existing path. Finally, they are used to create clauses with constants.

6 Future Work

OSL, especially OSL-M1, currently adds many new clauses at each step, which significantly increases the computational cost. However, since OSL creates clauses from all the paths encountered in the search, some of the short clauses are subclauses of the long ones. So it may be better to only keep the long ones since they have more information. Second, OSL currently does not use clauses in the existing MLN to restrict the search space. So it would be useful to exploit this information. Finally, it would be interesting to apply OSL to other learning problems that involve data with many structured examples. For instance, other natural-language problems such as semantic role labeling or computer-vision problems such as scene understanding [7].

7 Conclusions

In this work, we present OSL, the first online structure learner for MLNs. At each step, OSL uses mode-guided relational pathfinding to find new clauses that fix the model's wrong predictions. Experimental results on field segmentation on two real-world datasets show that OSL is able to find useful new clauses that improve the predictive accuracies of well-developed MLNs and also learned effective MLNs from scratch.

Acknowledgments. The authors thank IBM for the free academic license of CPLEX. This research is supported by ARO MURI grant W911NF-08-1-0242 and by a gift from Microsoft Research. Most of the experiments were run on the Mastodon Cluster, provided by NSF Grant EIA-0303609. The first author also thanks the Vietnam Education Foundation (VEF) for its sponsorship.

References

1. Biba, M., Ferilli, S., Esposito, F.: Discriminative structure learning of markov logic networks. In: Železný, F., Lavrač, N. (eds.) ILP 2008. LNCS (LNAI), vol. 5194, pp. 59–76. Springer, Heidelberg (2008)
2. Carreras, X., Màrquez, L.: Introduction to the CoNLL-2005 shared task: Semantic role labeling. In: Proc. of the 9th Conf. on Computational Natural Language Learning (CoNLL 2005), pp. 152–164 (2005)
3. Della Pietra, S., Della Pietra, V.J., Lafferty, J.D.: Inducing features of random fields. IEEE Trans. on Pattern Analysis and Machine Intelligence 19(4), 380–393 (1997)
4. Domingos, P., Lowd, D.: Markov Logic: An Interface Layer for Artificial Intelligence. Morgan & Claypool Publishers, San Francisco (2009)
5. Duboc, A.L., Paes, A., Zaverucha, G.: Using the bottom clause and mode declarations on FOL theory revision from examples. In: Železný, F., Lavrač, N. (eds.) ILP 2008. LNCS (LNAI), vol. 5194, pp. 91–106. Springer, Heidelberg (2008)
6. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Tech. rep., EECS Department, University of California, Berkeley (2010), `http://www.cs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html`
7. Fei-Fei, L., Li, L.J.: What, Where and Who? Telling the Story of an Image by Activity Classification, Scene Recognition and Object Categorization. In: Computer Vision: Detection, Recognition and Reconstruction, pp. 157–171 (2010)
8. Genesereth, M.R., Nilsson, N.J.: Logical foundations of artificial intelligence. Morgan Kaufmann, San Francisco (1987)
9. Getoor, L., Taskar, B. (eds.): Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
10. Grenager, T., Klein, D., Manning, C.D.: Unsupervised learning of field segmentation models for information extraction. In: Proc. of the 43nd Annual Meeting of the Asso. for Computational Linguistics, ACL 2005 (2005)
11. Huynh, T.N., Mooney, R.J.: Online max-margin weight learning with Markov Logic Networks. In: Proc. of the 2011 SIAM Int. Conf. on Data Mining (SDM 2011), pp. 642–651 (2011)
12. Huynh, T.N., Mooney, R.J.: Max-Margin Weight Learning for Markov Logic Networks. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009. LNCS, vol. 5781, pp. 564–579. Springer, Heidelberg (2009)
13. Kok, S., Domingos, P.: Learning the structure of Markov logic networks. In: ICML 2005 (2005)
14. Kok, S., Domingos, P.: Learning Markov logic network structure via hypergraph lifting. In: Proc. of 26th Int. Conf. on Machine Learning (ICML 2009), pp. 505–512 (2009)
15. Kok, S., Domingos, P.: Learning Markov logic networks using structural motifs. In: Proc. of 27th Int. Conf. on Machine Learning (ICML 2010), pp. 551–558 (2010)
16. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proc. of 18th Int. Conf. on Machine Learning (ICML 2001), pp. 282–289 (2001)
17. Lawrence, S., Giles, C.L., Bollacker, K.D.: Autonomous citation matching. In: Proc. of the 3rd Annual Conf. on Autonomous Agents, pp. 392–393 (1999)
18. Lee, S., Ganapathi, V., Koller, D.: Efficient structure learning of Markov networks using L_1-regularization. In: Adv. in Neu. Infor. Processing Systems (NIPS 2006), vol. 19, pp. 817–824 (2007)

19. McCallum, A.: Efficiently inducing features of conditional random fields. In: Proc. of 19th Conf. on Uncertainty in Artificial Intelligence (UAI 2003), pp. 403–410 (2003)
20. Mihalkova, L., Huynh, T., Mooney, R.J.: Mapping and revising Markov logic networks for transfer learning. In: Proc. of the 22nd Conf. on Artificial Intelligence (AAAI 2007), pp. 608–614 (2007)
21. Mihalkova, L., Mooney, R.J.: Bottom-up learning of Markov logic network structure. In: Proc. of 24th Int. Conf. on Machine Learning, ICML 2007 (2007)
22. Mihalkova, L., Mooney, R.J.: Learning to disambiguate search queries from short sessions. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009. LNCS, vol. 5782, pp. 111–127. Springer, Heidelberg (2009)
23. Muggleton, S.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
24. Ong, I.M., de Castro Dutra, I., Page, D., Costa, V.S.: Mode directed path finding. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 673–681. Springer, Heidelberg (2005)
25. Perkins, S., Theiler, J.: Online feature selection using grafting. In: Proc. of 20th Int. Conf. on Machine Learning (ICML 2003), pp. 592–599 (2003)
26. Poon, H., Domingos, P.: Joint inference in information extraction. In: Proc. of the 22nd Conf. on Artificial Intelligence (AAAI 2007), pp. 913–918 (2007)
27. Richards, B.L., Mooney, R.J.: Learning relations by pathfinding. In: Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI 1992), pp. 50–55 (1992)
28. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62, 107–136 (2006)
29. Riedel, S., Meza-Ruiz, I.: Collective semantic role labelling with Markov logic. In: Proc. of the 12th Conf. on Computational Natural Language Learning (CoNLL 2008), pp. 193–197 (2008)
30. Slattery, S., Craven, M.: Combining statistical and relational methods for learning in hypertext domains. In: Page, D.L. (ed.) ILP 1998. LNCS, vol. 1446, pp. 38–52. Springer, Heidelberg (1998)
31. Sutton, C., McCallum, A.: An introduction to conditional random fields for relational learning. In: Getoor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning, pp. 93–127. MIT Press, Cambridge (2007)
32. Taskar, B., Guestrin, C., Koller, D.: Max-margin Markov networks. In: Adv. in Neu. Infor. Processing Systems, NIPS 2003, vol. 16 (2004)
33. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: Proc. of 21st Int. Conf. on Machine Learning (ICML 2004), pp. 104–112 (2004)
34. Zelle, J.M., Thompson, C.A., Califf, M.E., Mooney, R.J.: Inducing logic programs without explicit negative examples. In: Swierstra, S.D. (ed.) PLILP 1995. LNCS, vol. 982, pp. 403–416. Springer, Heidelberg (1995)
35. Zhu, J., Lao, N., Xing, E.P.: Grafting-light: fast, incremental feature selection and structure learning of Markov random fields. In: Proc. of the 16th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2010), pp. 303–312 (2010)