A Soft Version of Predicate Invention Based on Structured Sparsity

William Yang Wang, Kathryn Mazaitis, William W. Cohen
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.
{yww,krivard,wcohen}@cs.cmu.edu

Abstract

In predicate invention (PI), new predicates are introduced into a logical theory, usually by rewriting a group of closely-related rules to use a common invented predicate as a "subroutine". PI is difficult, since a poorly-chosen invented predicate may lead to error cascades. Here we suggest a "soft" version of predicate invention: instead of explicitly creating new predicates, we *implicitly* group closely-related rules by using structured sparsity to regularize their parameters together. We show that soft PI, unlike hard PI, consistently improves over previous strong baselines for structure-learning on two large-scale tasks.

1 Introduction

In relational learning, *predicate invention* (PI) is a method in which new predicates are introduced into a logical theory. Most PI techniques simplify a logical theory by allowing a group of closely-related rules to be combined somehow, using the invented predicate.

Although not often, but practically PI may be viewed from a sparse structure learning perspective: early PI algorithms from the inductive logic programming community often leverage similar patterns from first-order logic representations, and then invent new predicates to compress the first-order formulas to form compact theories. For example, in learning logical rules for the domain of family relations, one might have learned the following clauses:

$$\textit{daughter(X,Z), father(Z,Y)} \Rightarrow \textit{sister(X,Y)}$$
$$\textit{daughter(X,Z), mother(Z,Y)} \Rightarrow \textit{sister(X,Y)}$$

A PI system like CHAMP [Kijsirikul *et al.*, 1992] would create a new predicate by combining these similar rules: e.g., it might invent a predicate "*parent*", along with an appropriate definition (as the disjunction of *father* and *mother*), and then compress the above clauses into[1]:

$$\textit{daughter(X,Z), parent(Z,Y)} \Rightarrow \textit{sister(X,Y)}$$

The difficulty with applying such PI invention methods is that they are somewhat prone to errors when data is noisy. Past approaches to PI have avoided these problems by some combination of clean data and computationally-intensive search[2] over structure space [Kemp *et al.*, 2006; Kok and Domingos, 2007].

Intuitively, PI is motivated by the principle that parsimonious explanations of the data are likely generalize well. A similar bias towards parsimonious theories is made by methods such as Lasso [Tibshirani, 1996], which "sparsify" a model by pushing the weights for some features to zero. Sparsity-encouraging regularization methods are a common tool in analyzing complex, high-dimensional datasets, and have been useful in domains including text classification Forman [2003] and vision [Olshausen and Field, 1997; Wright *et al.*, 2010]. Sparsity-encouraging regularization methods are often viewed as "softer" substitutes for feature selection.

It is natural to conjecture that sparse regularization might make PI more robust, by effectively removing "noisy" invented predicates. Notice, however, that an invented predicate P couples the performance of all rules that use P: in the example above, for instance, the performance of the *parent* rules are coupled to the performance of the *sister* rule that calls it. This suggests that *structured sparsity* methods such as the group Lasso [Friedman *et al.*, 2010; Yuan and Lin, 2006] might be more useful for PI.

Further reflection suggests another connection between structured sparsity and PI. Consider the set of rules that would be simplified by an invented predicate. PI compresses a theory by replacing this set with a smaller one, thus reducing the number of parameters to learn. A structured sparsity regularizer that regularizes together the parameters for this set also reduces the number of parameters to learn, in a very analogous way. We call this soft version of predicate invention *soft PI*. Soft PI does not explicitly creating new symbols to compress the existing theory: instead, soft PI relies on modifying the learner, via a regularizer, to exploit the same commonalities.

In this paper, we explore the connections between PI and sparsity-encouraging regularizers. More specifically, we use the iterated structural gradient (ISG) approach [Wang *et al.*,

[1] We use the predicate symbol "parent" for clarity–a real invented predicate would have a meaningless name, like *invented16*.

[2] We note that structure search is especially expensive in probabilistic logics, where inference is generally non-trivial.

```
about(X,Z) :- handLabeled(X,Z)            # base.
about(X,Z) :- sim(X,Y),about(Y,Z)         # prop.
sim(X,Y) :- links(X,Y)                    # sim,link.
sim(X,Y) :-
   hasWord(X,W),hasWord(Y,W),
   linkedBy(X,Y,W)                        # sim,word.
linkedBy(X,Y,W) :- true                   # by(W).
```

Table 1: A simple program in ProPPR. See text for explanation.

2014a] to identify potentially useful rules. We then apply CHAMP-like heuristics to identify groups of clauses that could be compressed with invented predicates. We compare "hard PI" methods, in which the invented predicates are actually introduced, with "soft PI", in which we impose a group Lasso penalty regularization term to learn parameters for a final set of clauses, with structured sparsity. We compare these approaches with non-structured sparse and non-sparse regularizers, as well as an alternative structured sparsity regularizer, namely a sparse graph Laplacian regularization exploits pair-wise relationships between rules.

Our approach is built on top of ProPPR [Wang *et al.*, 2013], a new, scalable first-order probabilistic logic, which is fast enough to support probabilistic inference on large problems [Wang *et al.*, 2014b]. The methods we explore here are highly scalable: when using a parallel stochastic gradient descent learner with lazy proximal structured sparsity updates, learning takes only a few minutes to process 20,000 examples against a 10,000-tuple database. We also scale the group Lasso approach to a version of the NELL [Carlson *et al.*, 2010] KB with 100K facts, while achieving good performances on the "long tail" of inferences in the KB.

To summarize, our contributions are as follows. (1) We present a new freely available family relation dataset with more than 30,000 of facts for SRL research. This is analogous to a much smaller dataset introduced by Hinton [Hinton, 1986]. (2) We introduce an overlapping proximal group Lasso algorithm to regularize together related clauses. (3) We show that this soft version of PI can outperform strong PI-free baselines in the NELL KB completion task, and the family-relation learning task.

2 Background on ProPPR

Below we will give an informal description of ProPPR, based on a small example. More formal descriptions can be found elsewhere [Wang *et al.*, 2013].

ProPPR (for Programming with Personalized PageRank) is a stochastic extension of the logic programming language Prolog. A simple program in ProPPR is shown in Table 1. Roughly speaking, the upper-case tokens are variables, and the ":-" symbol means that the left-hand side (the *head* of a rule) is implied by the conjunction of conditions on the right-hand size (the *body*). In addition to the rules shown, a ProPPR program would include a *database* of *facts*: in this example, facts would take the form *handLabeled(page,label)*, *hasWord(page,word)*, or *linkedBy(page1,page2)*, representing labeled training data, a document-term matrix, and hyperlinks, respectively. The condition "true" in the last rule is "syntactic sugar" for an empty body.

In ProPPR, a user issues a query, such as "about(a,X)?", and the answer is a set of possible bindings for the free variables in the query (here there is just one such varable, "X"). To answer the query, ProPPR builds a *proof graph*. Each node in the graph is a list of conditions $R_1, \ldots, R_k$ that remain to prove, interpreted as a conjunction. To find the children of a node $R1, \ldots, Rk$, you look for either

1. database facts that match R_1, in which case the appropriate variables are bound, and R_1 is removed from the list, or;
2. a rule $A \leftarrow B_1, \ldots, B_m$ with a head A that matches R_1, in which case again the appropriate variables are bound, and R_1 is replaced with the body of the rule, resulting in the new list $B_1, \ldots, B_m, R_2, \ldots, R_k$.

In Prolog, this proof graph is constructed on-the-fly in a depth-first, left-to-right way, returning the first solution found, and backtracking, if requested, to find additional solutions. In ProPPR, however, we will define a *stochastic process on the graph*, which will generate a score for each node, and hence a score for each answer to the query. The stochastic process used in ProPPR is *personalized PageRank* [Page *et al.*, 1998; Csalogny *et al.*, 2005], also known as random-walk-with-restart. Intuitively, this process upweights solution nodes that are reachable by *many short proofs* (i.e., short paths from the query node.) Formally, personalized PageRank is the fixed point of the iteration

$$\mathbf{p}^{t+1} = \alpha \chi_{v_0} + (1 - \alpha) W \mathbf{p}^t \tag{1}$$

where $\mathbf{p}[u]$ is the weight assigned to u, v_0 is the seed (i.e., query) node, χ_{v_0} is a vector with $\chi_{v_0}[v_0] = 1$ and $\chi_{v_0}[u] = 0$ for $u \neq v$, and the parameter α is the reset probability. W is a matrix of transition probabilities, i.e., $W[v, u]$ is the probability of transitioning from node u to a child node v:

$$W[v, u] = \frac{1}{Z} f\big(\theta \cdot \phi_{[v,u]}\big) \tag{2}$$

Here Z is an appropriate normalizing constant, θ is the weight vector associated with the features $\phi_{[v,u]}$ on edge $[v, u]$. The edge strength functions f used in this study are rectified linear united (ReLU) [Nair and Hinton, 2010] and the hyperbolic tangent function (tanh) [Glorot and Bengio, 2010].

Like Prolog, ProPPR's proof graph is also constructed on-the-fly, but rather than using depth-first search, we use PageRank-Nibble, a fast approximate technique for incrementally exploring a large graph from a an initial "seed" node [Andersen *et al.*, 2008]. PageRank-Nibble takes a parameter ϵ and will return an approximation $\hat{\mathbf{p}}$ to the personalized PageRank vector $\mathbf{p}$, such that each node's estimated probability is within ϵ of correct. ProPPR can be viewed as a scalable extension of stochastic logic programs[Muggleton, 1996; Cussens, 2001; Van Daele *et al.*, 2014].

We close this background section with some final brief comments about ProPPR.

Scalability. ProPPR is currently limited in that it uses memory to store the fact databases, and the proof graphs constructed from them. ProPPR uses a special-purpose scheme based on sparse matrix representations to store facts which are triples, which allows it to accommodate databases with hundreds of millions of facts in tens of gigabytes.

With respect to run-time, ProPPR'S scalability is improved by the fast approximate inference scheme used, which is typically an order of magnitude faster than power iteration for moderate-sized problems [Wang *et al.*, 2013], and much faster on larger problems. Experimentation and learning are also sped up because with PageRank-Nibble, each query is answered using a "small"—size $O(\frac{1}{\alpha\epsilon})$—proof graph. Many operations required in learning and experimentation can thus be easily parallelized on a multi-core machine, by simply distributing different proof graphs to different threads.

Parameter learning. The personalized PageRank scores are defined by a transition probability matrix W. ProPPR allows "feature generators" to be attached to its rules, as indicated by the code after the hashtags in the example program: for instance, when matching the rule "sim(X,Y) :- links(X,Y)" to a condition such as "sim(a,X)" the two features "sim" and "link" are generated, and when matching the rule "linkedBy(X,Y,W) :- true" to the condition "linkedBy(a,c,sprinter)" the feature "by(sprinter)" is generated. Since edges in the proof graph correspond to rule matches, the edges can also be labeled by features, and a weighted combination of these features can be used to define a total weight for each edge, which finally can be normalized used to define the transition matrix W. Learning can be used to tune these weights to data; ProPPR's learning uses a parallelized SGD method, in which inference on different examples is performed in different threads, and weight updates are synchronized.

Structure learning. Prior work [Wang *et al.*, 2014a] has studied the problem of learning a ProPPR theory, rather than simply tuning parameters in an existing theory, a process called *structure learning*. In particular, inspired by recent advances in inductive logic programming [Muggleton *et al.*, 2014], Wang *et al.* [2014a] propose a scheme called the *structural gradient* which scores every rule in some (possibly large) user-defined space $\mathcal{R}$ of potential rules, and then adds high-scoring rules to a theory. In more detail, the space of potential rules $\mathcal{R}$ is defined by a "second-order abductive theory", which conceptually constructs proofs using all rules in $\mathcal{R}$. The second-order theory is defined in such a way such that each parameter in the second-order theory corresponds to a rule in $\mathcal{R}$, so the gradient of the parameter vector corresponds to a scoring scheme for the rules in $\mathcal{R}$. The structure learning via parameter learning idea of ProPPR's structure learning method is broadly related to joint structure and parameter learning of Markov Logic Networks [Khot *et al.*, 2011]. The iterated structural gradient method that incrementally refines the hypothesized structure space is also closely related to a learn-and-join algorithm for learning Markov Logic Networks [Khosravi *et al.*, 2010].

3 Hard Predicate Invention

In previous work [Wang *et al.*, 2014a] involving the structural gradient method, the space of rules $\mathcal{R}$ includes rules over a fixed set if predicate symbols that are known in advance. In some cases it is useful to invent new predicates and define them. For instance, in learning a definition of *aunt* using the pre-defined predicates *mother*, *father*, a system might include the rules:

aunt(X,Y) :- sister(X,Z),mother(Z,Y).
aunt(X,Y) :- sister(X,Z),father(Z,Y).

A potentially more compact definition for *aunt* might be found by inventing the new predicate *invented1* and defining it as the disjunction of *mother* and *father*. Constructing and defining new predicate symbols in this way is called predicate invention (PI).

Existing PI approaches involve creating new predicates based on similarities [Wogulis and Langley, 1989] and differences [Muggleton and Buntine, 1992] between learned rules. However, many PI systems are not robust enough to handle noisy data, as when incorrect predicates are invented, errors may eaily cascade. In this paper we evaluated several variations of a CHAMP-style analysis [Kijsirikul *et al.*, 1992] to invent predicates based on sets of similar rules. We consider pairs of rules to be similar if they have the following format.

- R1 is "p(X,Y) :- q(X,Z),r(Z,Y)" and R2 is "p(X,Y) :- q(X,Z),s(Z,Y)", i.e., they are length-two chains that differ only in the last predicate of the RHS, or
- R1 is "p(X,Y) :- q(X,Y)" and R2 is "p(X,Y) :- s(X,Y)", i.e., they are length-one chains that differ only in the last predicate of the RHS, or
- R1 is "p(X,Y) :- q(Y,X)" and R2 is "p(X,Y) :- s(Y,X)", which is the inverse relation case.

In our preliminary experiments on learning family relations (detailed settings and quantitative results will be shown in Section 5.), the results that "hard PI" were produced as follows:

nephew(X,Y) :- invented1(Y,X).
invented1(X,Y) :- uncle(X,Y).
invented1(X,Y) :- aunt(X,Y).
uncle,X,Y) :- invented2(Y,X).
invented2(X,Y) :- nephew(X,Y).
invented2(X,Y) :- niece(X,Y).
aunt(X,Y) :- sister(X,Z),invented1(Z,Y).
sister(X,Y) :- niece(X,Z),invented1(Z,Y).
. . .
brother(X,Y) :- invented1(X,Y).
uncle(X,Y) :- uncle(X,Z),invented1(Z,Y).

Although the majority of the compressed rules produced by hard PI are intuitively meaningful, the last two rules are not.

4 Structured Sparsity for Soft Predicate Invention

From the example in the previous section, we see that a drawback of hard predicate invention is that, given noisy inputs, incorrect clauses may be generated. In this section, we present a structured sparsity based alternative to hard PI: instead of creating new symbols, our approach groups similar concepts together, and exploit regularization-based structured sparsity technique to explore closely-related concepts and rules.

Element-Wise Regularization In this subsection, we briefly review past work on regularization techniques. Here we define the weight parameter vector **w**, and each weight element in **w** corresponds to a first-order logic clause candidate, according to prior work on structure learning using parameter learning for first-order logic [Wang *et al.*, 2014a]. Note that ProPPR's default regularization term $\mu||\mathbf{w}||_2^2$, which is the Ridge estimator [Le Cessie and Van Houwelingen, 1992], will not be producing sparse estimates. The noisy estimates may not be ideal, since the incorrect first-order logic program may lead to more errors in the downstream applications. To solve this issue, we consider the following Lasso [Tibshirani, 1996] formulation, where objective function is:

$$\min\Big(-\ell + \mu||\mathbf{w}||_1\Big)$$

Unlike the Ridge estimator, the above Lasso penalty will now produce sparse estimates, even though the objective function is now non-differentiable. To optimize the above function, we use a proximal operator: each weight component w is shrunk towards 0 by a shrinkage value σ,

$$signum(w) \cdot max(0, |w| - \sigma)$$

where in our lazy L_1 regularization update is

$$\sigma = \delta\sqrt{2}\mu\beta.$$

here, δ is the total number of accumulated regularization update, and β is the learning rate. We use a Lazy L_1 update algorithm [Carpenter, 2008] for optimization. The main idea is that, the regularization updates for all features in each example are slow and unnecessary, and we can cache the regularization updates of relevant features, then update the accumulated regularization changes.

Structured Graph Laplacian Regularization One challenge associated with PI is that the reuse of invented predicates makes them hard to remove by simple element-wise regularizers. To better incorporate the dependencies among the features in each logic clause, we introduce a pair-wise graph Laplacian penalty [Belkin *et al.*, 2006]:

$$\min\Big(-\ell + \zeta\sum_{(p,q)}||w_p - w_q||A_{(p,q)}\Big)$$

Here, ζ is the regularization coefficient that controls the strength of the structured penalty, and $A_{(p,q)}$ is an adjacency matrix that indicates the pair-wise similarity among logic clauses, and the basic idea is to push similar logic clauses to have similar weights after learning. For example, consider the following clauses:

sister(X,Y) :- daughter(X,Z), father(Z,Y).
sister(X,Y) :- daughter(X,Z), mother(Z,Y).

Since they share the same goal and the same first predicate on the right hand side (RHS), it make sense for them to have similar weights. To construct the sparse A matrix , we use a CHAMP-style analysis [Kijsirikul *et al.*, 1992]: for all pairs of clauses that share the same goal and have $|RHS| = 1$ cases, we connect them in the adjacency graph. For all pairs of clauses that share the same goal and have $|RHS| = 2$ cases, if the first predicates on the RHS are the same, we connect the two clauses together in A. For instance, instead of inventing a hard predicate for "nephew or niece", we instead assign a weight value of 1 for this pair of rules in the affinity matrix A.

To implement this approach, we then define a degree matrix D to be the total number of connections for each entry in A, and the graph Laplacian to be $L = D - A$, which transforms the optimization into:

$$\min\Big(-\ell + \zeta\mathbf{w}^T L\mathbf{w}\Big)$$

where ℓ is the loss. Like ridge regression, this regularizer is a quadratic penalty.

Sparse Laplacian Regularization To incorporate sparsity, we may also consider this alternative sparse Laplacian regularization formula:

$$\min\Big(-\ell + \mu||\mathbf{w}||_1 + \zeta\mathbf{w}^T L\mathbf{w}\Big)$$

where the sparsity-inducing L_1 term was added.

Structured Sparsity via Group Lasso A problem with Laplacian regularization is that while it pushes similar pairs of clauses to have similar weights, it will not typically remove groups in incorrect related clauses by pushing their weights to zero. To solve this problem, we introduce a sparse group Lasso [Friedman *et al.*, 2010; Yuan and Lin, 2006] formulation:

$$\min\Big(-\ell + \mu||\mathbf{w}||_1 + \zeta\sum_{l=1}^{L}||\mathbf{w}_l||_2\Big)$$

where L is the total number of feature groups, and $\mathbf{w}_l$ is the parameter vector for the l-th group. This is now a structured sparsity learning problem, where we can introduce group sparsity. To generate the groups, we utilize the same A matrix in the Laplacian regularization: each row in A corresponds to a feature group, and different groups may have overlapping features. The benefit of using sparse group Lasso is that it will drive the weights for the entire group of features to zero if the group is not useful or noisy, which appears to be critical for soft PI. Again, we take the first-order derivatives of the group Lasso term, and use the same proximal operator algorithm to solve the sparse group Lasso optimization. We also consider a "group lasso" alternative of the above formulation when we remove the element-wise L_1 term. Note that Nishino *et al.* [2014] is among the first to study a projected gradient approach for learning sparse parameters in relational parameter learning, but the problems of learning sparse structures and predicate invention are not discussed.

5 Experiments

In this section, we evaluate the effectiveness of the proposed approach on two datasets: a new, large family relation dataset[3], which features the Kings and Queens of Europe, including Great Britain's royal family up to 1992; as well as

[3] This is motivated by Hinton's classic kinship dataset, which includes only two families, each with twelve individuals, and twelve binary relations between these individuals.

Methods	MAP(ReLU)	AUC-PR(ReLU)	#c	Time	MAP(tanh)	AUC-PR(tanh)	#c	Time
Hard Predicate Invention	.708 ± .012	.713 ± .011	79	09:05	.713 ± .008	.720 ± .007	79	09:26
No Predicate Invention	.744 ± .010	.751 ± .011	101	11:24	.764 ± .018	.771 ± .015	101	12:25
+ Ridge	.752 ± .009	.758 ± .010	101	11:05	.785 ± .010	.794 ± .008	101	11:47
+ Lasso ▲	.763 ± .015	.773 ± .020	95	10:23	.792 ± .018	.798 ± .017	95	10:45
Soft Predicate Invention								
+ Laplacian ▲	.766 ± .013	.782 ± .013	101	11:23	.770 ± .011	.781 ± .010	101	12:01
+ Group Lasso ▲	.761 ± .015	.777 ± .014	86	09:25	.768 ± .013	.779 ± .012	88	09:56
+ Sparse Laplacian ▲	.773 ± .007	.777 ± .005	95	10:14	.790 ± .007	.798 ± .007	95	10:36
+ Sparse Group Lasso ▲	**.801 ± .015**	**.812 ± .012**	65	08:01	**.807 ± .012**	**.813 ± .014**	63	07:55

Table 2: The MAP results from non-iterated structural gradients for KB completion on the royal family dataset. #c: the averaged number of logic clauses with non-zero weights across all runs. Time: the averaged runtime (minutes) of inference on the test. ▲ indicates that comparing to the no PI baseline, the p-values of the repetitive McNemar tests are all $< .0001$.

the NELL subsets that include up to 100K grounded facts extracted from the Web. In particular, we focus on the task of structure learning for *knowledge base completion* [Wang *et al.*, 2014a; Cropper and Muggleton, 2014], where the goal is to learn first-order logic program to reconstruct the KB, given only partially complete background database. For comprehensive empirical comparisons of ProPPR's structure learning scheme to the Markov Logic Networks' structure learning baseline, we refer the readers to prior work [Wang *et al.*, 2014a].

5.1 Learning Family Relations

We introduce a new dataset for research in SRL: the original dataset was created in 1992 by Denis R. Reid, including 3010 individuals and 1422 families of European royalty. We further parsed the genealogical data to extract six pairs of inter-related family relations: {*uncle & aunt, sister & brother, daughter & son, father & mother, husband & wife, niece & nephew*}. (Learning such pairs of relations has proven to be quite difficult for ProPPR structure-learning systems in past work [Wang *et al.*, 2014a].) We use a temporal split to separate the train and test subsets. The training set includes 21,430 facts, while the test set contains 8,899 facts. In our KB completion experiment, we randomly delete 50% of the background facts for both the training and test sets respectively, and we use soft PI to complete the missing facts, and evaluate the effectiveness of our approach using Mean Average Precision (MAP), and Area Under the precision-recall Curve (AUC-PR). Throughout the experiments, the regularization coefficient μ for the L_2 penalty was set to 0.00001, and μ for the L_1 penalty was set to 0.00002. We repeat each experiment 3 times and report the averaged score and the standard deviation of the results. We also report the number of non-zero rules after learning, as well as the inference time on the test set. The McNemar test [McNemar, 1947] is used to test the statistical significance of various models.

Table 2 shows the MAP results for KB completion on the royal family dataset, using the non-iterated and iterated structural gradient variants [Wang *et al.*, 2014a] respectively. We see that hard PI performs poorly, due to the error cascades.[4] Traditional element-wise non-sparse and sparse methods, namely, the Ridge and Lasso estimators, did help improving the test performance. The pair-wise graph Laplacian regularization also improves the performance. This is probably because by forcing similar logic clauses to learn similar weights, we are implicitly learning similarities among various clauses, and use them to find informative clauses that lead to better predictive results. We observe that for soft PI, the proposed sparse overlapping group Lasso method do have strong gain on this task: it outperforms all the competitive baselines by a large margin. In general, we also see the advantage of soft PI for inference with probabilistic logic programs— they tend to lead to better predictive performances than hard PI or no PI solutions. Our results also align with the findings in a prior study on empirical loss minimization with sparsity-inducing regularization terms [Duchi and Singer, 2009]: both papers suggest that the compact parameter space with fewer non-zero weights may lead to better predictive results.

	KB seed	
	Google	*baseball*
top 1k entities		
#train/test queries	100	100
#DB facts	853	890
top 10k entities		
#train/test queries	1000	1000
#DB facts	10630	11972
top 100k entities		
#train/test queries	5000	5000
#DB facts	12902	9746

Table 3: Summary of the KBs used in experiments on completing subsets of NELL's KB. Note that we also use a temporal split to create the train/test split for this experiment.

5.2 Completing the NELL KB

Finally, as a larger-scale and more realistic task, we explore learning inference rules for the NELL knowledge base. The NELL (Never Ending Language Learning) research project is an effort to develop a never-ending learning system that operates 24 hours per day, for years, to continuously improve its ability to read (extract structured facts from) the web [Carlson *et al.*, 2010]. NELL is given as input an ontology that defines hundreds of categories (e.g., person, beverage, athlete,

[4] Hard PI here is the best of several PI techniques we experimented with. Details are omitted due to space.

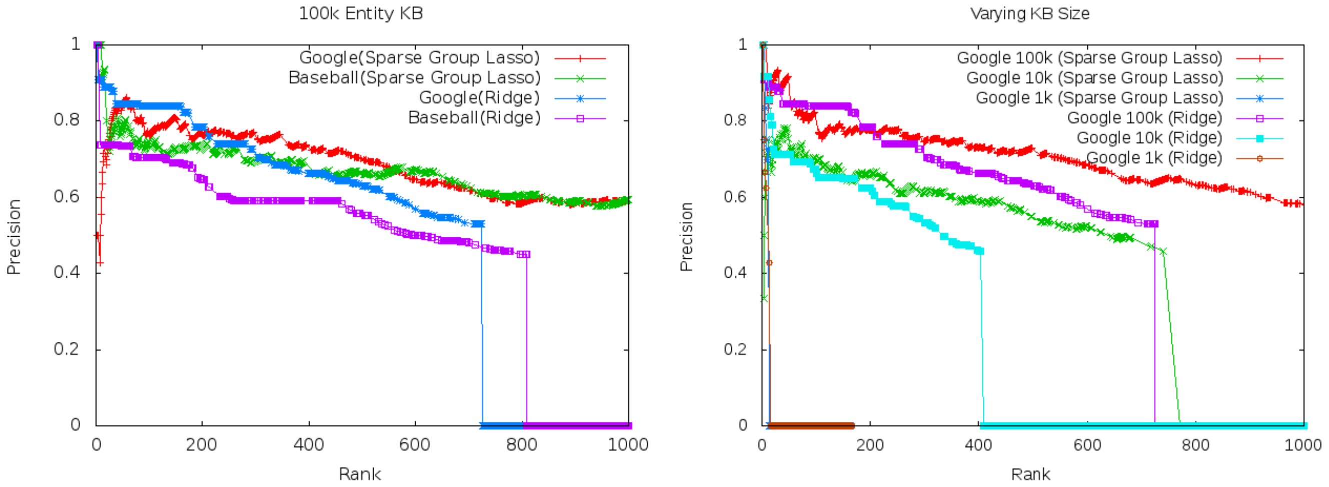

Figure 1: Performance on completing subsets of the NELL KB. Left, interpolated precision vs rank for two KBs with 100k entities; right; comparison on three KBs of different sizes based on the seed "Google".

sport) and two-place typed relations among these categories (e.g., *athletePlaysSport(Athlete, Sport)*), which it must learn to extract from the web. NELL is also provided a set of 10 to 20 positive seed examples of each such category and relation, along with a downloaded collection of 500 million web pages from the ClueWeb2009 corpus (Callan and Hoy, 2009) as unlabeled data, and access to 100,000 queries each day to Google's search engine. NELL uses a multi-strategy semi-supervised multi-view learning method to iteratively grow the set of extracted "beliefs".

For experimental purposes, we construct a number of varying-sized versions of the KB using the following procedure. First, we construct a "knowledge graph", where the nodes are entities and the edges are the binary predicates from NELL. Then, we pick a seed entity s, and find the M entities that are ranked highest using a simple untyped random walk with restart over the full knowledge graph from seed s. Finally, we project the KB to just these M entities: i.e., we select all entities in this set, and all unary and binary relationships from the original KB that concern only these M entities. Here the seed entities are "Google" and "Baseball". We use the same datasets from Wang *et al.*[2014a], and compare with their ridge method. The summary of the dataset is shown in Table 3.

Inference on NELL's learned KB is challenging for two reasons. First, the learned KB is not only incomplete, but also noisy, since it is extracted imperfectly from the web. For example, a football team might be wrongly recognized as two separate entities, one with connections to its team members, and the other with a connection to its home stadium. Second, the inference problems are large.

The performance of soft PI via sparse group Lasso on these tasks is shown in Figure 1. Here we sort the list of answers by their PPR scores in the descending order, and the rank corresponds to the position in the list. Even though the data is noisy, we see that the overlapping sparse group Lasso learns large and useful theories—theories such that the high-confidence predictions do indeed correspond, in most cases, with facts actually in the NELL knowledge base. Comparing to the Ridge estimator, our proposed method is better at modeling the long-tail distribution of facts on all of the "Google" and "Baseball" subsets in the NELL KB. For conciseness, we do not show other baselines in the figures, but the element-wise and structured regularization models' results are consistent with the family dataset.

6 Conclusions

In this work, we investigate an alternative approach to hard predicate invention. Instead of explicitly inventing new predicates, our approach relies on structured regularization techniques to learning similar clauses that lead to the discovery of informative clauses and better predictive performances. In particular, we focus on a new, scalable logic called ProPPR. More specifically, we use an iterated structural gradient approach and CHAMP-style compression to induce the overlapping groups, and solve a group Lasso problem. To reduce the overhead in regularization, we propose a Lazy proximal update algorithm.

We also introduce a new royal family dataset[5] for research in statistical relational learning: it contains more than 30K of facts about royal families in Europe. On this dataset, we show that our proposed soft PI model improves over various hard PI and no PI baselines by a large margin. In addition to this, we demonstrate the scalability of our approach by performing sparse group Lasso experiments on the NELL dataset. By comparing to the non-sparse L_2 regularization method, it is shown that our proposed sparse group Lasso method has better performances on modeling the long tail distribution of the NELL KB.

Acknowledgment

This work was sponsored in part by DARPA grant FA87501220342 to CMU and a Google Research Award.

[5] http://www.cs.cmu.edu/˜yww/data/family_data.zip

References

[Andersen *et al.*, 2008] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local partitioning for directed graphs using pagerank. *Internet Mathematics*, 5(1):3–22, 2008.

[Belkin *et al.*, 2006] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*, 2006.

[Carlson *et al.*, 2010] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[Carpenter, 2008] Bob Carpenter. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. *Alias-i, Inc., Tech. Rep*, pages 1–20, 2008.

[Cropper and Muggleton, 2014] Andrew Cropper and Stephen H Muggleton. Can predicate invention in meta-interpretive learning compensate for incomplete background knowledge? *Proc. of ILP*, 2014.

[Csalogny *et al.*, 2005] Kroly Csalogny, Dniel Fogaras, Balzs Rcz, and Tams Sarls. Towards scaling fully personalized PageRank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358, 2005.

[Cussens, 2001] James Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3):245–271, 2001.

[Duchi and Singer, 2009] John Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009.

[Forman, 2003] George Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003.

[Friedman *et al.*, 2010] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A note on the group lasso and a sparse group lasso. *arXiv preprint arXiv:1001.0736*, 2010.

[Glorot and Bengio, 2010] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[Hinton, 1986] Geoffrey E Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, pages 1–12. Amherst, MA, 1986.

[Kemp *et al.*, 2006] Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, 2006.

[Khosravi *et al.*, 2010] Hassan Khosravi, Oliver Schulte, Tong Man, Xiaoyuan Xu, and Bahareh Bina. Structure learning for Markov logic networks with many descriptive attributes. In *AAAI*, pages 487–493, 2010.

[Khot *et al.*, 2011] Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude W. Shavlik. Learning Markov logic networks via functional gradient boosting. In *ICDM*, 2011.

[Kijsirikul *et al.*, 1992] Boonserm Kijsirikul, Masayuki Numao, and Masamichi Shimura. Discrimination-based constructive induction of logic programs. In *AAAI*, 1992.

[Kok and Domingos, 2007] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *Proc. of ICML*, 2007.

[Le Cessie and Van Houwelingen, 1992] Saskia Le Cessie and JC Van Houwelingen. Ridge estimators in logistic regression. *Applied statistics*, pages 191–201, 1992.

[McNemar, 1947] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[Muggleton and Buntine, 1992] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the fifth international conference on machine learning*, pages 339–352, 1992.

[Muggleton *et al.*, 2014] Stephen H Muggleton, Dianhuan Lin, Jianzhong Chen, and Alireza Tamaddoni-Nezhad. Metabayes: Bayesian meta-interpretative learning using higher-order stochastic refinement. In *Inductive Logic Programming*, pages 1–17. Springer, 2014.

[Muggleton, 1996] Stephen Muggleton. Stochastic logic programs. *Advances in inductive logic programming*, 32:254–264, 1996.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[Nishino *et al.*, 2014] Masaaki Nishino, Akihiro Yamamoto, and Masaaki Nagata. A sparse parameter learning method for probabilistic logic programs. In *StarAI*, 2014.

[Olshausen and Field, 1997] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

[Page *et al.*, 1998] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. In *Technical Report, Computer Science department, Stanford University*, 1998.

[Tibshirani, 1996] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[Van Daele *et al.*, 2014] Dries Van Daele, Angelika Kimmig, and Luc De Raedt. Pagerank, proppr, and stochastic logic programs. 2014.

[Wang *et al.*, 2013] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *Proc. of CIKM*, 2013.

[Wang *et al.*, 2014a] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Structure learning via parameter learning. *Proc. of CIKM*, 2014.

[Wang *et al.*, 2014b] William Yang Wang, Kathryn Mazaitis, Ni Lao, Tom Mitchell, and William W Cohen. Efficient inference and learning in a large knowledge base: Reasoning with extracted information using a locally groundable first-order probabilistic logic. *arXiv:1404.3301*, 2014.

[Wogulis and Langley, 1989] James Wogulis and Pat Langley. Improving efficiency by learning intermediate concepts. In *IJCAI*, pages 657–662. Citeseer, 1989.

[Wright *et al.*, 2010] John Wright, Yi Ma, Julien Mairal, Guillermo Sapiro, Thomas S Huang, and Shuicheng Yan. Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE*, 98(6):1031–1044, 2010.

[Yuan and Lin, 2006] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.