Relational Learning via Propositional Algorithms: An Information Extraction Case Study*

Dan Roth **Wen-tau Yih**

Department of Computer Science
University of Illinois at Urbana-Champaign
{danr, yih}@uiuc.edu

Abstract

This paper develops a new paradigm for relational learning which allows for the representation and learning of relational information using propositional means. This paradigm suggests different tradeoffs than those in the traditional approach to this problem – the ILP approach – and as a result it enjoys several significant advantages over it. In particular, the new paradigm is more flexible and allows the use of any propositional algorithm, including probabilistic algorithms, within it.

We evaluate the new approach on an important and relation-intensive task - Information Extraction - and show that it outperforms existing methods while being orders of magnitude more efficient.

1 Introduction

Relational learning is the problem of learning structured concept definitions from structured examples. Relational representations use a first-order model to describe the problem domain and examples are given to the learner in terms of objects and object relations rather than by simple propositions.

In a variety of AI problems such as natural language understanding related tasks, visual interpretation and planning, given a collection of objects along with some relations that hold among them, the fundamental problem is to learn definitions for some relations or concepts of interest in terms of the given relations. Examples include the problem of identifying noun phrases in a sentence in terms of the information in the sentence, detecting faces in an image or defining a policy that maps states and goals to actions in a planning situation. In many of these cases it is natural to represent and learn concepts relationally; propositional representations might be too large, could lose much of the inherent domain structure and consequently might not generalize well. In recent years, this realization has renewed the interest in studying relational representations and learning.

Inductive Logic Programming (ILP) is an active subfield of machine learning that addresses relational learning and is a natural approach to apply to these tasks. While, in principle, ILP methods could allow induction over relational structures and unbounded data structures, theoretical and practical considerations render the use of unrestricted ILP methods impossible. Studies in ILP suggest that unless the rule representation is severely restricted the learning problem is intractable. While there are several successful heuristics for learning ILP programs, there are many practical difficulties with the inflexibility, brittleness and inefficient "generic" ILP systems. In most cases, researchers had to develop their own, problem specific, ILP systems [Mooney, 1997] but have not always escaped problems such as search control and inefficiency - especially in large scale domains like NLP.

This paper develops a different paradigm for relational learning that allows the use of general purpose and efficient propositional algorithms, but nevertheless learns relational representations. Our paradigm takes a fresh look at some of the restrictions ILP systems must make in order to work in practice and suggests alternatives to these; as a result, it enjoys several significant advantages over traditional approaches. In particular, the new paradigm is more flexible and allows the use of any propositional algorithm, including probabilistic algorithms, within it. It maintains the advantages of ILP approaches while allowing more efficient learning, improved expressivity and robustness.

At the center of our paradigm is a knowledge representation language that allows one to efficiently represent and evaluate rich relational structures using propositional representations. This allows us to learn using propositional algorithms but results in relational concepts descriptions as outcome.

This paper evaluates the new paradigm in the domain of Information Extraction (IE). This is the NLP task of extracting specific types or relevant items from unrestricted text. Relational learning methods are especially appealing for learning in this domain since both the target concepts and the information in the domain (within and across documents) are often relational. We develop an IE system based on our paradigm; the learning component of our system makes use of a feature efficient learning algorithm that is especially suitable for the nature of our paradigm. However, we show that standard learning algorithms like naive Bayes also work well with it. Experimental comparisons show that our approach outperforms several ILP-based systems tried on this tasks, sometime significantly, while being orders of magnitude more efficient.

*Research supported by NSF grants IIS-9801638 and IIS-0085836 and an ONR MURI Award.

2 Propositional Relational Representations

In this section we present a knowledge representation language that has two components: (1) a subset of first order logic (FOL) and (2) a collection of structures (graphs) defined over elements in the domain.

The relational language $\mathcal{R}$ is a restricted (function free) first order language for representing knowledge with respect to a domain $\mathcal{D}$. The restrictions on $\mathcal{R}$ are applied by limiting the formulae allowed in the language to a collection of formulae that can be evaluated very efficiently on given instances (interpretations). This is done by (1) defining primitive formulae with limited scope of the quantifiers (Def. 2.1). (2) General formulae are defined inductively in terms of primitive formulae in a restricted way that depends on the relational structures in the domain. The emphasis is on locality with respect to these relational structures that are represented as graphs over the domain elements.

This language allows the encoding of first order representations and relational structures as propositions and thus supports the use of better learning algorithms, including general purpose propositional algorithms and probabilistic algorithms over the elements of the language. This approach extends previous related constructions from ILP [Lavrac *et al.*, 1991; Khardon *et al.*, 1999] but technically is more related to the latter. In the rest of this section we present the main constructs of the language. We omit many of the standard definitions and concentrates on the unique characteristics of $\mathcal{R}$. See, e.g., [Lloyd, 1987] for general details.

The vocabulary Σ consists of constants, variables, predicate symbols, quantifiers, and connectives.

Definition 2.1 *A primitive formula is defined inductively: (1) A term is either a variable or a constant. (2) Let p be a k-ary predicate, $t_1, \cdots, t_k$ terms. Then $p(t_1, \cdots, t_k)$ is an atomic formula. (3) Let F be an atomic formula, and z be a variable. Then $(\forall z F)$ and $(\exists z F)$ are atomic formulae. (4) An atomic formula is a primitive formula. (5) If F and G are primitive formulae, then so are $(\neg F)$, $(F \wedge G)$, $(F \vee G)$.*

Notice that for primitive formulae in $\mathcal{R}$ the *scope* of a quantifier is always the unique predicate that occurs with it in the atomic formula. We call a variable-less atomic formula a *proposition* and a quantified atomic formula, a *quantified proposition* [Khardon *et al.*, 1999]. The informal semantics of the quantifiers and connectives is as usual.

Before we move on to extend the language $\mathcal{R}$ we discuss the domain and the notion of an instance.

Definition 2.2 (Domain) *The language $\mathcal{R}$ is defined over a domain $\mathcal{D}$ which consists of a* structured element *$D = \langle \mathcal{V}, \mathcal{G} \rangle$ where $\mathcal{V}$ is a collection of typed elements and $\mathcal{G}$ is a set of partial orders over $\mathcal{V}$ (specifically, each partial order $g_i \in \mathcal{G}$ is an acyclic graph (V_i, E_i), where $V_i \subseteq \mathcal{V}$ and E_i is a set of edges on V_i). Along with it, for each constant there is an assignment of an element in V and for each k-ary predicate, an assignment of a mapping from V^k to {0,1} ({true, false}).*

Example 2.1 *The fragment in Fig. 1 forms a domain $\mathcal{D} = (\mathcal{V}, \mathcal{G})$, where V consists words* **TIME, :, 3,:, 30, pm** *and the phrase* **3 : 30 pm** *and $\mathcal{G}$ consists of two linked lists (denoted in the solid line and the dashed line). In general g_i could be more complex and represent, say, a parse tree over a sentence.*

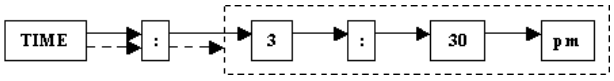

Figure 1: A fragment of an article

The notion of types is used as a way to classify elements in the language according to their properties. In particular, we will think also of predicates as typed in the sense that their domain is typed. We distinguish within the set $\mathcal{V}$ two main types; a set $\mathcal{O} \subset \mathcal{V}$ of *objects* and a set $\mathcal{A} \subset \mathcal{V}$ of *attributes*. Correspondingly, we define types of predicates. Type 1 predicates take as their first argument an element in $\mathcal{O}$ and as their second argument an element in $\mathcal{A}$. Type 1 predicates describe properties of elements in $\mathcal{O}$; thus $p(o, a)$ may also be written as $p(o) = a$, with the semantics that in a given interpretation, $p(o, a)$ holds. All other predicate types will have elements in $\mathcal{O}$ as their arguments. These predicates will be defined via elements in $\mathcal{G}$, and will have $g \in \mathcal{G}$ as their type. Specifically, $p_g(o_1, o_2)$ indicates that o_1, o_2 are nodes in the graph $g \in \mathcal{G}$ and there is an edge in g between them.

Elements in $\mathcal{O}$ represent objects in the world. For example, in NLP applications such as ours these might be words, phrases, sentences or documents. Predicates of type 1 describe properties of these elements – spelling of a word, syntactic tag of a word, a phrase type, etc. The graphs in $\mathcal{G}$ describe relations between (sets of) objects – word w_1 is *before* w_2, w_1 is the *subject* of the verb w_2, etc. (It is possible to generalize $\mathcal{G}$ to a collection of hypoergraphs, but this is not needed in the current application.) As usual, the "world" in which we interpret the aforementioned constructs could be a single sentence, a document, etc.

Definition 2.3 *An* instance *is an interpretation[Lloyd, 1987] which lists a set of domain elements and the truth values of all instantiations of the predicates on them.*

Given an instance x, a formula F in $\mathcal{R}$ is given a unique truth value, *the value of F on x*, defined inductively using the truth values of the predicates in F and the semantics of the connectives. Since for primitive formulae in $\mathcal{R}$ the scope of a quantifier is always the unique predicate that occurs with it in the atomic formula, we have the following properties. It will be clear that the way we extend the language (Sec. 2.1) maintains this properties (proofs omitted).

Proposition 2.1 *Let F be a formula in $\mathcal{R}$, let x be an instance, and let t_p be the time to evaluate the truth value of an atom p in F. Then, the value of F on x can be evaluated in time $\sum_{p \in F} t_p$.*

That is, F is evaluated simply by evaluating each of its atoms (ground or quantified) separately. This holds, similarly, for the following version of subsumption for formulae in $\mathcal{R}$.

Proposition 2.2 (subsumption) *Let x be an instance and let $f : \{0,1\}^n \rightarrow \{0,1\}$ be a Boolean function of n variables that can be evaluated in time t_f. Then the value of the clause $f(F_1, \ldots F_n)$ on x can be evaluated in time $t_f + \sum_F t_F$, where the sum is over all n formulae that are arguments of f.*

2.1 Relation Generation Functions

The definition of the general formulae allowed in $\mathcal{R}$ will be operational and will use the structures in $\mathcal{G}$. To do that we introduce a mechanism that generates more expressive formulae in a way that respects the structures in the domain, thus restricting the formulae generated.

Definition 2.4 *A formula in $\mathcal{R}$ maps an instance x to its truth value in x. It is* active *in x if it has truth value* true *in it. We denote by X the set of all instances – the* instance space. *A formula $F \in \mathcal{R}$ is thus a* relation *over X, $F : X \to \{0, 1\}$.*

Example 2.2 *Let instance x be the fragment illustrated in Ex. 2.1. Some active relations in x are* word*(TIME),* word*(pm), and* number*(30).*

Given an instance, we would like to know what are the relations (formulae) that are *active* in it. We would like to do that, though, without the need to write down explicitly all possible formulae in the domain. This is important, in particular, over infinite domains or in problems domains such as NLP, where inactive relations vastly outnumber active relations.

Definition 2.5 *Let $\mathcal{X}$ be an enumerable collection of relations on X. A relation generation function (RGF) is a mapping $G : X \to 2^{\mathcal{X}}$ that maps $x \in X$ to a set of all elements in $\mathcal{X}$ that satisfy $\chi(x) = 1$. If there is no $\chi \in \mathcal{X}$ for which $\chi(x) = 1$, $G(x) = \phi$.*

RGFs can be thought of as a way to define "kinds" of formulae, or to parameterize over a large space of formulae. Only when an instance x is presented, a concrete formula (or a collection of) is generated. An RGF can be thought of as having its own range $\mathcal{X}$ of relations.

Example 2.3 *It is impossible to list all formulae that use the* number *predicate in advance. However, RGF can specify formulae of this kind and, given the instance* `TIME : 3 : 30 pm`*, only the active relations of this kind:* number*(3) and* number*(30) – are generated.*

In order to define the collection of formulae in $\mathcal{R}$ we define the family of RGFs for $\mathcal{R}$; the output of these define the formulae in $\mathcal{R}$. RGFs are defined inductively using a relational calculus. The alphabet of this calculus consists of (i) basic RGFs, called *sensors* and (ii) a set of connectives. While the connectives are the same for every alphabet the *sensors* vary from domain to domain. A sensor is a way to encode basic information one can extract from an instance. It can also be used as a uniform way to incorporate external knowledge sources that aid in extracting information from an instance.

Definition 2.6 *A* sensor *is a relation generation function that maps an instance x into a set of atomic formulae in $\mathcal{R}$. When evaluated on an instance x a sensor s outputs all atomic formulae in its range which are* active.

Example 2.4 *Following are some sensors that are commonly used in NLP.*

- *The* word *sensor over word elements, which outputs active relations* word*(TIME),* word*(:),* word*(3),* word*(30), and* word*(pm) from "TIME : 3 : 30 pm".*
- *The* length *sensor over phrase elements, which outputs active relations* len*(4) from "3 : 30 pm".*
- *The* is-a *sensor, outputs the semantic class of a word.*
- *The* tag *sensor, outputs the part-of-speech tag of a word*

The word *and* len *sensors derive information directly from the raw data, while the* is-a *sensor uses external information sources such as WordNet and the* tag *sensor uses a prelearned part-of-speech tagger.*

Several mechanisms are used in the relational calculus to define the operations of RGFs. We mention here only the *focus* mechanism which is actually a *binding* mechanism that is used to define quantified formulae in $\mathcal{R}$.

Definition 2.7 *Let E be a set of elements in the domain. An RGF r is* focused *on E if, given an instance x, it generates only formulae in its range that are active in x due to elements in E. The focused RGF is denoted $r[E]$.*

There are several ways to define a focus set. It can be done explicitly or using the structure $\mathcal{G}$. The focus set is equivalent to the free variables in FOL representations.

The relational calculus allows one to inductively generate new RGFs by applying connective and quantifiers over existing RGFs. Using the standard connectives one can define RGFs that output formulae of the type defined in Def 2.1. Details will not be given here. Instead, we describe only one important type of operations within the relational calculus – structural operations. These operations exploit the structural (relational) properties of the domain as expressed in $\mathcal{G}$ in order to define RGFs. Thus, more general formulae, that can have interactions between variables, are generated, while still allowing for efficient evaluation and subsumption, due to the graph structure. For example, the structural collocation operator, *colloc*, with respect to g is defined as follows.

Definition 2.8 *Let $s_1, s_2, \ldots s_k$ be RGFs for $\mathcal{R}$. $colloc_g(s_1, s_2, \ldots s_k)$ is a restricted conjunctive operator that is evaluated on a chain of length k in g. Specifically, let $\mathcal{D} = (\mathcal{V}, \mathcal{G})$ be a domain, with $g \in \mathcal{G}$, and $v_1, v_2, \ldots v_k$ a chain in g. The formulae generated by $colloc_g(s_1, s_2, \ldots s_k)$ are those generated by $s_1[v_1] \& s_2[v_2] \& \ldots \& s_k[v_k]$, where (1) by $s_j[v_j]$ we mean that the RGF s_j is focused on $\{v_j\}$ (2) the $\&$ operator means that formulae in the output of $(s \& r)$ are active formulae of the form $F \wedge G$, where F is in the range of s and G is in the range of r (evaluated on x). This is needed since each RGF in the conjunction may produce more then one formulae.*

Example 2.5 *When applied with respect to the graph g which represents the linear structure of the sentence, $colloc_g$ simply generates formulae that corresponds to ngrams. E.g., given the fragment "Dr John Smith", RGF* colloc(word, word) *extracts the bigrams* `word(Dr)-word(John)` *and* `word(John)-word(Smith)`.

Similarly to $colloc_g$ one can define a *sparse* collocation operator with respect to a chain in g. This is also a restricted conjunctive operator that is evaluated on a chain in g with the following difference. Formulae are generated by $scolloc_g(s_1, s_2, \ldots s_k)$ as follows: Let $v_1, v_2, \ldots v_n$ be a chain in g. For each subset $v_{i_1}, v_{i_2}, \ldots v_{i_k}$ of elements in v, such that $i_j < i_l$ when $j < l$, all the formulas: $s_1[v_{i_1}] \& s_2[v_{i_2}] \& \ldots \& s_k[v_{i_k}]$, are generated.

Notice that while primitive formulae in $\mathcal{R}$ have a single predicate in their scope, the structural properties provides a way to go beyond that but only in a restricted way that is efficiently evaluated. Structural operations allow us to define RGFs that constrain formulae evaluated on different objects without incurring the cost usually associated with enlarging the scope of free variables. This is done by enlarging the scope only as required by the structure of the domain, modeled by $\mathcal{G}$. This allows for efficient evaluation as in Prop. 2.1, 2.2 with the only additional cost being that of finding chain in the graph (details omitted).

3 Comparison to ILP Methods

Propositional learning on relational features provides a different paradigm for relational learning. While, in principle, ILP methods could allow induction over relational structures and unbounded data structures and their expressivity cannot be matched by propositional methods, theoretical and practical considerations render the use of unrestricted ILP methods impossible. Studies in ILP suggest that, unless the rule representation is severely restricted, the learning problem is intractable [Kietz and Dzeroski, 1994; Cohen, 1995; Cohen and Page, 1995]. Several successful heuristics for learning ILP programs [Muggleton and De Raedt, 1994; Cussens, 1997; Quinlan, 1990] have made different algorithmic and representational restrictions in order to facilitate ILP although there are still many practical difficulties with the inflexibility, brittleness and inefficient "generic" ILP systems.

Our approach offers different tradeoffs than those suggested by the common restrictions made by current ILP systems. While we cannot say that our paradigm dominates the traditional ILP approach in general, we believe that in many cases the alternatives it offers provide a better way to address relational learning, especially in large scale domains such as NLP. Below we address some key issues that could help in developing a better understanding to the suitability of each.

Search: The key difference between the traditional ILP approach and ours is the way they structure the search space. In ILP, "features" are generated as part of the search procedure in an attempt to find good bindings. In our case, the "features" tried by an ILP program during its search are generated up front (in a data driven way) by the RGFs. Some of these are grounded and some have free variables in them. The learning algorithm will look at all of them "in parallel" and find the best representation. Search control methods used by ILP methods are thus analogous to the expressivity we give to our RGFs. The order of "visiting" these features is different.

Knowledge: One of the key cited advantages of ILP methods is the ability to incorporate background knowledge. In our paradigm, this is incorporated flexibly using the notion of *sensors*. Sensors allow us to treat information that is readily available in the input, external information or even previously learned concepts in a uniform way.

Expressivity(I): The basic building blocks of the representations we use (our formulae) are the same as those used by ILP representations. As presented in Sec. 2, for a predicate R and elements a, b we are not representing only the ground term $R(a, b)$ but also $R(X, Y), R(X, b)$, etc. This is similar to the work in [Lavrac *et al.*, 1991] only that our structural operations allow us to avoid some of the determinacy problems of that approach.

Learning: The relational features generated by our RGFs provide a uniform domain for different learning algorithms. Applying different algorithms is easy and straightforward. Moreover, it is straightforward to use probabilistic models over this representation and in this way it provides a natural and general way of using relational representations within a probabilistic framework. On the other hand, it turns out that "generic" ILP methods suffer brittleness and inefficiency and in many cases, researchers had to develop their own, problem specific, ILP systems [Mooney, 1997].

In particular, while time complexity is a significant problem for ILP methods, propositional learning is typically a lot more efficient. In our paradigm, due to the fact that our RGFs will generate a very large number of relational features (see "search" above) we adopt a specific learning methodology, following [Khardon *et al.*, 1999]. While this is not necessary (as shown in Sec. 5) we discuss this direction next.

Expressivity (II): Several issues can be mentioned in the context of using linear threshold functions as concept representation over the relational features extracted using RGFs [Khardon *et al.*, 1999]. Advocates of ILP methods suggest that the rich expressive power of FOL provides advantages for knowledge-intensive problems such as NLP [Mooney, 1997]. However, given strong intractability results, practical systems apply many representational restrictions. In particular, the depth of the clauses (the number of predicates in each clause) is severely restricted. Thus, the learned concept is actually a k-DNF, for small k. In our paradigm, the constructs of $colloc$ and $scolloc$ allow us to generate relational features which are conjunctions of predicates and are thus similar to a clause in the output representation of an ILP program. While an ILP program represents a disjunction over these, a linear threshold function over these relational features is more expressive. In this way, it may allow learning smaller programs. The following example illustrates the representational issues:

Example 3.1 *Assume that in several seminar announcements, fragments that represent* `speaker` *have the pattern:* $\cdots$ *Speaker : Dr FName LName* line-feed $\cdots$

An ILP rule for extracting `speaker` *could then be:*
before_targ(2, "Speaker") $\wedge$ *contains(target, "Dr")* $\wedge$ *after_targ(1,* line-feed*)* $\rightarrow$ `speaker`*(target)*
That is, the second word before the target phrase is "Speaker", target phrase contains word "Dr", and the "line-feed" character is right after the target phrase. In our relational feature space all the elements of this rule (and many others) would be features, but the above conjunction is also a feature. Therefore a collection of clauses of this form becomes a disjunction in our feature space and will be learned efficiently using a linear threshold element.

Finally, we mention that for learning linear threshold elements there exist feature efficient algorithms [Littlestone, 1988] that are suitable for learning in NLP-like domains, where the number of potential features is very large, but only a few of them are active in each example, and only a small

fraction of them are relevant to the target concept.

4 Case Study – Information Extraction

Information Extraction (IE) is a natural language processing (NLP) task that processes unrestricted text and attempts to extract specific types of items from the text.

This form of shallow text processing has attracted considerable attention recently with the growing need to intelligently process the huge amounts of information available in the form of text documents. While learning methods have been used earlier to aid in parts of an IE system [Riloff, 1993; Soderland and Lehnert, 1994], it has been argued quite convincingly [Califf and Mooney, 1999; Craven and Slattery, 2001] that relational methods are necessary in order to learn how to directly extract the desired items from documents. The reason is that the target concepts require the representation of relations over the source document and learning those might require induction over structured examples and FOL representations. Indeed, previous works [Califf and Mooney, 1999; Freitag, 2000] have demonstrated the success of ILP methods in this domain. This is therefore an ideal domain to study our proposed relational learning paradigm.

4.1 Problem Description

In this paper, the IE task is defined as locating specific fragments of an article according to predefined slots in a template. Each article is a plain-text document that consists of a sequence of tokens. Specifically, the data used in experiments is a set of 485 seminar announcements from CMU[1]. The goal is to extract four types of fragments from each article[2] – those describing the start time (`stime`) and end time (`etime`) of the seminar, its location (`location`) and the seminar's speaker (`speaker`). Given an article, our system picks at most one fragment for one slot. If this fragment represents the slot, then it is a correct prediction. Otherwise, it is a wrong prediction (including the case that the article doesn't contain the slot at all) [Freitag, 2000].

4.2 Extracting Relational Features

The basic strategy of our IE solution is to learn a classifier that discriminates a specific desired fragment. First, we generate examples for this type of fragment. This is done by:

1. Identifying candidate fragments in the document. (All fragments are candidates; in training, fragments are annotated.) Note: fragments may overlap, and only a small number of them contain desired information.
2. For each candidate fragment, use the defined RGF features to re-represent it as an example which consists of all active features extracted for this fragment.

Let $f = (t_i, t_{i+1}, \cdots, t_j)$ be a fragment, with t_i representing tokens and i, i+1, $\cdots$, j are positions of tokens in the document. Our RGFs are defined to extract features from three regions: left window ($t_{i-w}, \cdots, t_{i-1}$), target fragment($t_i, \cdots, t_j$), and right window ($t_{j+1}, \cdots, t_{j+w}$), where w is the window size.

The domain formed by these three regions contains two types of elements – word elements (e.g., $t_{i-w}, \cdots, t_{j+w}$) and one phrase element (the target region). The RGFs are focused either on a specific word element (one free variable) or on the borders of the phrase element (two free variables) and define relational features relative to these. In the next section we provide examples of RGFs used in experiments.

4.3 Two-stage Architecture

Once examples are generated, the IE task is accomplished by two learning stages. The same classifier, SNoW, is used in both stages, but in a slightly different way.

SNoW [Roth, 1998; Carleson *et al.*, 1999] is a multi-class classifier that is specifically tailored for large scale learning tasks. The SNoW learning architecture learns a sparse network of linear functions, in which the targets (fragment types, in this case) are represented as linear functions over a common feature space. SNoW has already been used successfully for a variety of tasks in natural language and visual processing [Golding and Roth, 1999; Roth *et al.*, 2000]. SNoW is built on a feature efficient learning algorithm, Winnow [Littlestone, 1988] and therefore is an ideal learning approach to complement our paradigm, as discussed before. While SNoW can be used as a classifier and predicts using a winner-take-all mechanism over the activation values of the target classes, here we rely directly on the activation value it outputs, computed using a sigmoid function over the linear sum. The normalized activation value can be shown to be a distribution function and we rely heavily on it's robustness in our two-stage architecture. The two stages are (1) Filtering: reduce the amount of candidates from all possible fragments to a small number, and (2) Classifying: pick the right fragment from the preserved fragments by the learned classifier. Theoretical justification for this architecture will be presented in a companion paper. Intuitively, this architecture increases the expressivity of our classification system. Moreover, eliminating most of the negative examples significantly reduces the number of irrelevant features, an important issue given the small data set.

Filtering: One common property of IE tasks is that negative examples (irrelevant fragments) extremely outnumber positive examples (fragments that represent legitimate slots). In the seminar announcements data set, for example, 0.3% of the fragments represent legitimate slots. This stage attempts to filter out most of the negative examples without eliminating positive examples. It can also be viewed as a classifier designed to achieve high recall, while the classifier in the second stage aims at high precision. The filter consists of two learned classifiers; a fragment is filtered out if it meets one of the following criteria:

1. Single feature classifier: Fragment doesn't contain a feature that should be active in positive examples.
2. General Classifier: Fragment's confidence value is below the threshold.

[1] The data set was originally collected from newsgroups and annotated by Dayne Freitag; it is available on http://www.isi.edu/∼muslea/RISE/.

[2] An article might not contain one of the fields, e.g., etime, or might contain one of them, e.g., the speaker, more than once.

System	stime			etime			loc			speaker		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
SNoW-IE	99.6	99.6	99.6	97.6	95.0	96.3	90.9	64.1	75.2	83.3	66.3	73.8
NB-IE	98.3	98.3	98.3	96.5	92.8	94.6	76.8	62.0	68.6	40.3	32.0	35.7
RAPIER-WT	96.5	95.3	95.9	94.9	94.4	94.6	91.0	61.5	73.4	79.0	40.0	53.1
RAPIER	93.9	92.9	93.4	95.8	94.6	95.2	91.0	60.5	72.7	80.9	39.4	53.0
SRV	98.6	98.4	98.5	67.3	92.6	77.9	74.5	70.1	72.2	54.4	58.4	56.3
WHISK	86.2	100.0	92.6	85.0	87.2	86.1	83.6	55.4	66.6	52.6	11.1	18.3

Table 1: Results for seminar announcements task

For criterion 1, it turns out that there exists some features that are (almost) always active in positive examples. For example, in our experiments, the length of fragments satisfies this: [$len(fragment) \leq 7$] is always satisfied by `stime` fragments. Also, [*fragment contains a word that is a noun*] always holds in `speaker` fragments.

For criterion 2, implemented using SNoW, relying on its robust confidence estimation, the problem becomes finding the right threshold. Minimum activation values of positive examples in training data are used as thresholds (for the different types of slots). Examples with lower activation values are filtered out.

The two stages also differ in the RGFs used. The following, more crude RGFs are used at the filtering stage:

- Target region: *word, tag, word&tag, colloc(word, word), colloc(word, tag), colloc(tag, word), colloc(tag, tag)* on word elements, and *len* on the phrase element.
- Left & Right window: *word&loc*, *tag&loc*, and *word&tag&loc*, where *loc* extracts the position of the word in the window.

Classifying: Fragments that survived the filtering stage are then classified using a second SNoW classifier, for the four slots. First, an additional collection of RGFs is applied to enhance the representation of the candidate fragments, thus allowing for more accurate classification. As before, in training, the remaining fragments are annotated and are used as positive or negative examples to train the classifiers. In testing, the remaining fragments are evaluated on the learned classifiers to determine if they can fill one of the desired slots. In this stage 4 different classifiers are trained, one for each type of fragments. All examples are run through all 4 classifiers. The RGFs added in this stage include:

For `etime` and `stime`:
scolloc[word&loc(-1)[l_window], word&loc[r_window]],
scolloc[word&loc(-1)[l_window], tag&loc[r_window]]

For `location` and `speaker`:
scolloc[word&loc(-2)[l_window], tag[targ], tag&loc(1)[r_window]]

The first set of RGFs is a sparse structural conjunction of the word directly left of the target region, and of words and tags in the right window (with relative positions). The second is a sparse structural conjunction of the last two words in the left window, a tag in the target, and the first tag in the right window.

A decision is made by each of the 4 classifiers. A fragment is classified as type t if the tth classifier decides so. At most one fragment of type t is chosen in each article, based on the activation value of the corresponding classifier.

slot	pass(training)	pass(testing)	loss(testing)
stime	4.75%	4.72%	0.94%
etime	1.07%	1.07%	1.64%
location	15.42%	15.30%	3.28%
speaker	14.89%	14.28%	2.79%

Table 2: Filtering efficiency

5 Experimental Results

Our experiments use the same data, methodology and evaluation metrics used by several ILP-based IE systems in previous works. The systems such as RAPIER [Califf and Mooney, 1999], SRV [Freitag, 2000], and WHISK [Soderland, 1999] have also been tested on this data set. The data (485 documents) is randomly split into two sets of equal sizes, one for training and the other for testing. The reported results are an average of five runs. As usual, the performance is quantified in terms of *Precision (P)* – the percentage of correct predictions – and *Recall (R)* - the percentage of *slots* that are identified. We also report the $F1 = (\frac{PR}{P+R})$. The results of our system, SNoW-IE, are shown in the first row of table 1 along with the results of several other ILP-based IE systems that were tested on this task under the same conditions. An exception is WHISK, for which the results are from a 10-fold validation using only 100 documents randomly selected from the training set. The systems also use somewhat different information sources. The words in the documents are used by all systems. Part-of-speech tags are used both in RAPIER-WT and SNoW-IE; SRV uses other predicates that capture POS information to some extent. A version of RAPIER uses also semantic information; this can be done in our system by adding, say, an is-a sensor but, given their results we did not incorporate this information.

In addition to our SNoW-IE we have also experimented with a second propositional algorithm, the naive Bayes (NB-IE) algorithm. NB was used on exactly the same set of features (same examples) that were generated using our relational paradigm for SNoW, and in exactly the same way. Although the results of NB-IE are not as good as those of SNoW-IE – the quality of the classifier is certainly an important issue – the experiments with a second propositional algorithm exhibit the fact that our relational paradigm is a general one. As indicated in [Craven and Slattery, 2001; Freitag, 2000] a simple minded use of this algorithm is not competitive for this task; but on top of a paradigm that is able to exploit the relational nature of the data it compares favor-

ably with ILP methods. Overall, SNoW-IE outperforms the existing rule-based IE systems on all the four slots.

To clarify, we note that the output representation of our system makes use of similar type of relational features as do the ILP-based systems, only that instead of a collection of conjunctive rules over these, it is represented as a linear function.

It is difficult to isolate the contribution of our two-stage architecture to the quality of the results. We believe, though, that the ease of incorporating this and other learning architectures is an indication to the flexibility of the approach and the advantages of learning with propositional means. Table 2 gives some insight into that, by showing the average performance of the filtering stage. The first two columns show the ratio of training and testing examples that pass the filter. The third column lists the ratio of positive examples in the testing set that are filtered out. These fragments do not even reach the second stage. (Since an article may contain more than one fragment that represents the same slot, it is sometimes still possible for the classifier to pick the correct slot.)

6 Conclusion

The use of relational methods is back in fashion. It became clear that for a variety of AI problems there is a fundamental need to learn and represent relations and concepts in terms of other relations. Information Extraction – the task of extracting relevant items from unrestricted text is one such task.

This paper suggests a new paradigm for relational learning – which allows for the representation and learning of relational information using propositional means. We argue that our paradigm has different tradeoffs than the traditional approach to this problem – the ILP approach – and as a result it enjoys several significant advantages over it. In particular, the suggested paradigm is more flexible and allows the use of any propositional algorithm within it, including probabilistic approaches. As such, it addresses in a natural and general way the problem of using relational representations within a probabilistic framework, an important problem which has been studied a lot recently.

Our paradigm is exemplified on an important task - Information Extraction (IE). Based on our paradigm, we developed a new approach to learning for IE, and have shown that it outperforms existing ILP-based methods. Moreover, it is several orders of magnitude more efficient. We believe that this work opens up several directions for further work – on relational learning and knowledge representation and on practical and efficient solutions to NLP and IE problems.

References

[Califf and Mooney, 1999] M. Califf and R. Mooney. Relational learning of pattern-match rules for information extraction. In *National Conference on Artificial Intelligence*, 1999.

[Carleson *et al.*, 1999] A. Carleson, C. Cumby, J. Rosen, and D. Roth. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC CS Dept., May 1999.

[Cohen and Page, 1995] W. Cohen and D. Page. Polynomial learnability and inductive logic programming: Methods and results. *New Generation Computing*, pages 369–409, 1995.

[Cohen, 1995] W. Cohen. PAC-learning recursive logic programs: Negative result. *Journal of Artificial Intelligence Research*, 2:541–573, 1995.

[Craven and Slattery, 2001] M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43:97–119, 2001.

[Cussens, 1997] J. Cussens. Part-of-speech tagging using progol. In *International Workshop on Inductive Logic Programming*, pages 93–108, Prague, Czech Republic, 1997. Springer. LNAI 1297.

[Freitag, 2000] D. Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(2/3):169–202, 2000.

[Golding and Roth, 1999] A. R. Golding and D. Roth. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999.

[Khardon *et al.*, 1999] R. Khardon, D. Roth, and L. G. Valiant. Relational learning for NLP using linear threshold elements. In *Proc. of the International Joint Conference of Artificial Intelligence*, pages 911–917, 1999.

[Kietz and Dzeroski, 1994] J. Kietz and S. Dzeroski. Inductive logic programming and learnability. *SIGART Bulletin*, 5(1):22–32, 1994.

[Lavrac *et al.*, 1991] N. Lavrac, S. Dzeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In *Machine Learning (EWSL-91)*, volume 482 of *LNAI*, pages 265–281, Porto, Portugal, 1991. Springer Verlag.

[Littlestone, 1988] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[Lloyd, 1987] J. W. Lloyd. *Foundations of Logic Progamming*. Springer-verlag, 1987.

[Mooney, 1997] Raymond J. Mooney. Inductive logic programming for natural language processing. In *(ILP-96)*, volume 1314 of *LNAI*, pages 3–24. Springer, 1997.

[Muggleton and De Raedt, 1994] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 20:629–679, 1994.

[Quinlan, 1990] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Riloff, 1993] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *National Conference on Artificial Intelligence*, pages 811–816, 1993.

[Roth *et al.*, 2000] D. Roth, M-H. Yang, and N. Ahuja. Learning to recognize objects. In *CVPR'00*, 2000.

[Roth, 1998] D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *National Conference on Artificial Intelligence*, pages 806–813, 1998.

[Soderland and Lehnert, 1994] S. Soderland and W. Lehnert. Wrap-up: a trainable discourse module for information extraction. *Journal of Artificial Intelligence Research*, 2:131–158, 1994.

[Soderland, 1999] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.