

Relational Learning via Propositional Algorithms: An Information Extraction Case Study

Dan Roth (danr@uiuc.edu) and Wen-tau Yih (yih@uiuc.edu)

*Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801-2987, USA*

April 26, 2002

Abstract. This paper develops a new paradigm for relational learning which allows for the representation and learning of relational information using propositional means.

A relational language is developed, along with a generation function that produces features in this language, that represent the relational structures of a given domain and is suitable for use by propositional learning algorithms. We use the language and generation functions within a learning framework that is shown to learn efficiently and accurately concept representations in terms of the relational features. The paradigm is designed to support learning in domains that are relational but where the amount of data and size of representation learned are very large. It 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 scalable and more flexible and allows the use of any propositional algorithm, including probabilistic algorithms, within it.

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

Keywords: Relational Learning, Inductive Logic Programming, Information Extraction, SNoW

1. Introduction

Relational learning is the problem of learning structured concept definitions from structured examples. In relational learning a first-order language is used to describe the problem domain and examples are presented to the learner in terms of objects and object relations rather than by simple propositions.

A variety of AI problems such as natural language understanding related tasks, visual interpretation and planning, are often described by a collection of objects along with some relations that hold among them. The fundamental problem there is to learn definitions for some relations or concepts of interest in terms of properties of the given objects and relations among them. 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 interest in studying relational representations and learning.

Inductive Logic Programming (ILP) is an active subfield of machine learning that addresses relational learning and is the 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 (Kietz and Dzeroski, 1994; Cohen, 1995; Cohen and Page, 1995). While there are several successful heuristics for learning ILP programs (Muggleton and De Raedt, 1994; Cussens, 1997; Quinlan, 1990), there are many practical difficulties with the inflexibility, brittleness and inefficiency of “generic” ILP systems. In most cases, researchers had to develop their own *problem-specific* ILP systems (Mooney, 1997), but did not always escape from 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 all 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 concept descriptions as outcome.

The learning approach presented here focuses, as do other learning approaches, including ILP, on producing a definition (a classifier) for a single predicate, that is a function of other domain predicates, those predicates that serve as a representation of the domain. In our approach, these domain predicates (“features”) are elements in a language that we define, a subset of FOL; formulae in this language can be efficiently evaluated, given a domain element, to either True or False, and thus the learning algorithm can view these as propositions. In this sense, our approach is related to a class of propositionalization methods that have been suggested in ILP (Kramer et al., 2001). It differs from existing methods both in its representation power and in the way it interacts with the learning algorithm, to allow, essentially, unlimited induction of relational structures, in a parameterized way. In our approach features are generated up front, before the learning stage, in a data-

driven way, allowing us to dictate the level of complexity of our intermediate representation before any search for a learned function occurs. In particular, domain knowledge and the relational structure of the domain are incorporated in this way into features. Thus, expressive features that could not possibly be generated during the search itself are allowed to influence our final learned function in a significant way.

After describing the general learning paradigm in Sec. 2, Sec. 3 describes the main technical contribution of this paper – a relational language for features and a mechanism, Relational Generation Functions (RGF) that allows for a data driven generation of features. We then discuss some of the key difference between standard relational approach and the one proposed here Sec. 4.

This paper evaluates the new paradigm in the domain of Information Extraction (IE). Information extraction is the NLP task of extracting relevant items, typically functional phrases – such as the *author* of a document, a *speaker* in a seminar, an *address* of a restaurant or the *current occupation* of a job candidate – 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. In Sec. 5 we describe the IE system we have developed 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 within it. Experimental comparisons (Sec. 6) show that our approach outperforms several ILP-based systems tried on this task, sometimes significantly, while being orders of magnitude more efficient.

2. The Learning Framework

Our learning framework attempts to learn classifiers for relational learning problems by utilizing propositional learning algorithms. The key to our propositionalization technique is the combination of a data driven feature extraction stage and a learning stage. The first makes use of relational generation functions which define “types” of features to be extracted, and maps a relational representation of the data along with a specific target property or relation of interest (a “label”) to a propositional representation with respect to this target. The latter utilizes a feature-efficient propositional learning algorithm to learn a function representing the target relation or property.

In our framework, as in ILP, observations in the domain are assumed to be represented as a collection of predicates that hold in elements in the domain. From this, the task then becomes to produce a classifier that can predict,

given a particular element, whether a given predicate holds in it. For example, we may wish to predict that for some domain element X (e.g., a phrase in a sentence), the predicate `location( $X$ )` – indicating that the phrase X represents a location – holds, or that for some domain elements X, Y , the predicate `authorof( $X, Y$ )` – indicating that X is the author of Y , holds.

To accomplish this task using propositional learning algorithms, we must generate examples in the form of lists of propositions (features) that represent information relevant to the predicate to be learned. Propositions of this form may either be fully ground as in the predicate `location(washington)` or individually quantified as in the predicate $\exists X \text{author}(\text{Roth}, X) \wedge \text{book}(X)$.

Each list of propositions, generated with respect to some element(s) serves as a positive example for a predicate (a relation, resp.) that holds in it, and can also serve as a negative example for the existence of “competing” predicates or relations. These examples are used to train a classifier for each relation type, a classifier that defines a mapping from this features space to $\{\text{True}, \text{False}\}$ for the predicate of interest. Our first major task then becomes to re-represent the data in a manner conducive to producing features for which we can learn a good discriminating function. Later we will show how by defining a focus of attention region “near” the predicate to be learned, we can differentiate the range of features produced based on the current target. In order to facilitate this re-representation, we develop the notion of a relational generation function; it allows us to define the “type” of features we would like to generate and map the properties of the domain and its relational structure, relative to a specific element, to a propositional example in a data driven way. This language extends ideas presented first in (Cumby and Roth, 2000; Khardon et al., 1999).

As we will see, the feature extraction method we present operates with the so-called closed-world assumption, generating only the features judged to be active in the example relative relation generating function. All other features are judged to be inactive, or false. As it may be inefficient or impossible to list all possible features that could be active for a particular interpretation, this is very significant from a computational complexity perspective. Thus our learning algorithm should be able to accept examples of variable length. In addition, given a small set of “types” of features, our method generates a large number of features, each time with respect to different “focus” elements; thus, our learning algorithm should be able to learn well in the presence of a possibly large number of irrelevant features.

Although not required by the framework presented here, for the learning component, we use the SNoW learning system. SNoW¹ is a mutliclass propositional classifier suitable for learning in high dimension, especially when the input examples are sparse; it handles variable length examples and uses a

¹ Available at <http://L2R.cs.uiuc.edu/~cogcomp>.

network of linear functions to learn the target concept. Learning a hypothesis as a linear threshold function makes our hypotheses more expressive, as well as easier to learn, relative to conjunctive “rule” representations. SNoW employs a feature-efficient learning algorithm, Winnow (Littlestone, 1988), and learns well in the presence of many irrelevant features. It has been shown to be successful in large scale NLP and IE problems (e.g., (Roth, 1998)).

3. 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 a limited scope of quantifiers (Def. 3.1), and (2) defining general formulae 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, in learning definitions in terms of relational expressions. This approach extends previous related constructions from ILP (Lavrac et al., 1991) and (Khardon et al., 1999; Cumby and Roth, 2000), but technically is more related to the latter.

In the remainder of this section we present the main constructs of the language by first defining primitive formulae; then we describe the way we represent a domain, which allows us to move on and define the collection of all formulae in $\mathcal{R}$. We omit many of the standard FOL definitions and concentrate 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 3.1. *A primitive formula is defined inductively: (1) A term is either a variable or a constant. (2) Let p be a k -ary predicate, $t_1, \dots, t_k$ terms. Then $p(t_1, \dots, t_k)$ is an atomic formula. (3) Let F be an atomic formula, z 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 $(\neg F)$, $(F \wedge G)$, $(F \vee G)$.*

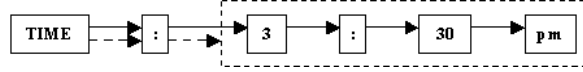


Figure 1. A fragment of an article

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-free 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 3.2 (Domain). *The language $\mathcal{R}$ is defined over a domain $\mathcal{D}$ which consists of a structured element $D = \langle \mathcal{V}, \mathcal{G} \rangle$; $\mathcal{V}$ 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).*

Constants and predicate symbols vary for different domain. In particular, for each constant in $\mathcal{R}$, there is an assignment of an element in V . For each k -ary predicate in $\mathcal{R}$, there is an assignment of a mapping from V^k to $\{0,1\}$ ($\{\text{true}, \text{false}\}$).

EXAMPLE 3.1. *The fragment in Fig. 1 forms a domain $\mathcal{D} = \langle \mathcal{V}, \mathcal{G} \rangle$, 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 general and represent, say, a parse tree over a sentence.*

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. For that reason 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*. Respectively, predicates of type A will take as their first argument an element in $\mathcal{O}$ and as their second argument an element in $\mathcal{A}$. In this way they describe properties of elements in $\mathcal{O}$. For example, $p(o, a)$, which may also be written as $p(o) = a$, is interpreted as “ $p(o, a)$ 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 the o_1, o_2 are nodes of 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 A 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. As usual, the “world” in which we interpret the abovementioned constructs could be a single sentence, a document, etc.

DEFINITION 3.3. *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. Notice that if F has the form $\exists p (\forall p, \text{resp.})$, for some k -ary predicate, then its truth value is *true* (1) iff there exists (for all, resp.) $d_1, \dots, d_k \in x$ such that $p(d_1, \dots, d_k)$ has truth value *true*. 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, the following properties trivially hold. It will be clear that the way we extend the language (Sec. 3.1) maintains this properties.

PROPOSITION 3.1. *Let F be a formula in $\mathcal{R}$, x 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 3.2 (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 $f(F_1, \dots, F_n)$ 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 .*

3.1. RELATION GENERATION FUNCTIONS

The definition of the general formulae allowed in $\mathcal{R}$ will be operational and will use the structures in $\mathcal{G}$. In order to do this we first introduce the mechanism used to generate more expressive formulae in a way that respects the structures in the domain, and thus, restricts the formulae generated.

DEFINITION 3.4. *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 , $F : X \rightarrow \{0, 1\}$.*

EXAMPLE 3.2. *Let instance x be the fragment illustrated in Ex. 3.1. Some active relations in x are $\text{word}(\text{TIME})$, $\text{word}(\text{pm})$, and $\text{number}(30)$.*

Given an instance, we would like to know what the relations (formulae) that are *active* in it are. 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 problem domains such as NLP, where inactive relations vastly outnumber active relations. Therefore, only active relations are noticed and recorded. As will be clear later, this notion will also allow us to significantly extend the language of formulae by exploiting properties of the domain.

DEFINITION 3.5. *Let $\mathcal{X}$ be an enumerable collection of relations on X . A relation generation function (RGF) is a mapping $G : X \rightarrow 2^{\mathcal{X}}$ 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 formulae) is generated. An RGF can be thought of as having its own range $\mathcal{X}$ of relations.

EXAMPLE 3.3. *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 $\text{TIME} : 3 : 30 \text{ pm}$, only the active relations of this kind: $\text{number}(3)$ and $\text{number}(30)$ and, potentially, $\text{number}(x)$ (see below) – 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 (1) basic RGFs, called *sensors* and (2) 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 3.6. *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 3.4. *Following are some sensors that are commonly used in NLP.*

- *The word sensor over word elements, which outputs active relations $\text{word}(\text{TIME})$, $\text{word}(:)$, $\text{word}(3)$, $\text{word}(30)$, and $\text{word}(\text{pm})$ from “ $\text{TIME} : 3 : 30 \text{ pm}$ ”.*

- The vowel sensor over word elements, which outputs $\text{vowel}(\text{word})$ or $\text{vowel}(x)$ when the word it operates on begins with a vowel.
- The length sensor over phrase elements, which outputs active relations $\text{len}(4)$ 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 pre-learned 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 3.7. *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 by 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 3.1. 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 and thereby generate more general formulae that can have more interactions between variables but still allow 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 3.8. *Let $s_1, s_2, \dots, s_k$ be RGFs for $\mathcal{R}$. $\text{colloc}_g(s_1, s_2, \dots, s_k)$ 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 $v_1, v_2, \dots, v_k$ a chain in g . The formulae generated by $\text{colloc}_g(s_1, s_2, \dots, s_k)$ are those generated by $s_1[v_1] \& s_2[v_2] \& \dots \& s_k[v_k]$, where (1) by $s_j[v_j]$ we mean here the RGF s_j focused to $\{v_j\}$ (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 than one formulae.*

EXAMPLE 3.5. 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 $colloc(\text{word}, \text{word})$ extracts the bigrams $\text{word}(\text{Dr}) - \text{word}(\text{John})$ and $\text{word}(\text{John}) - \text{word}(\text{Smith})$.

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 $scolloc_g(s_1, s_2, \dots, s_k)$ as follows: Let $v_1, v_2, \dots, v_n$ be a chain in g . For each subset $v_{i_1}, v_{i_2}, \dots, v_{i_k}$ of elements in v , such that $i_j < i_l$ when $j < l$, all the formulas: $s_1[v_{i_1}] \& s_2[v_{i_2}] \& \dots \& s_k[v_{i_k}]$, are generated.

Notice that while primitive formulae in $\mathcal{R}$ have a single predicate in their scope, the structural properties provide 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. 3.1, 3.2, with the only additional cost being that of finding a chain in the graph.

4. Comparison to ILP Methods

Propositional learning on relational features provides a different paradigm for relational learning. In principle, ILP methods could allow induction over relational structures and unbounded data structures, and their expressivity cannot be matched by propositional methods. However, 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 inefficiency of “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.

4.1. 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 by the RGFs. Some of these relational features 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.

4.2. 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.

4.3. 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. 3, 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.

4.4. 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 much 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 we show in Sec. 6), the rest of this discussion focuses in this direction.

4.5. EXPRESSIVITY (II)

Several issues can be mentioned in the specific context of using linear threshold functions as concept representations over the relational features extracted using RGFs. 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 -DNE, 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 4.1. *Assume that in several seminar announcements, fragments that represent **speaker** have the pattern:*

*... **Speaker** : Dr FName LName line-feed ...*

*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.

5. Case Study – Information Extraction

Information Extraction (IE) is a natural language processing (NLP) task that processes text and attempts to extract items with specific semantic or functional meaning from unrestricted and unstructured text. For example, in the domain of terrorism the information extraction system might extract names of perpetrators, locations of attacks, times of attacks etc. (DARPA, 1995).

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, 2000) 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.

This section first introduces the specific IE tasks studied in this paper and then explains how our relational paradigm is applied to generate examples of relational features from documents, followed by the description of the system architecture.

5.1. 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, our experiments make use of the following two data sets.

5.1.1. *Seminar Announcements*

The first data set consists of 485 CMU seminar announcements². The goal here is to extract four types of fragments from each announcements – those describing the start time (*stime*) and end time (*etime*) of the seminar, its location (*location*) and the seminar’s speaker (*speaker*). Note that 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.

Given an article, our system picks at most one fragment for each slot. In the evaluation, if the chosen fragment represents the slot, we consider it a correct prediction. Otherwise, it is a wrong prediction (including the case that the article doesn’t contain the slot at all) (Freitag, 2000).

5.1.2. *Computer-related Job Postings*

The second data set is a set of 300 computer-related job postings from UT Austin³. In this case, 17 types of fragments that describe the job position are

² The data set was originally collected from newsgroups and annotated by Dayne Freitag; it is available at <http://www.isi.edu/~muslea/RISE/>.

³ The data set was originally collected from newsgroups and annotated by Mary Califf; it is available at <http://www.isi.edu/~muslea/RISE/>.

... the talk given by: Dr FName LName *line-feed* Topic : A novel ...

Figure 2. Three regions for feature extraction

extracted from each article. Some of the fields, such as `title`, `company`, `salary`, take single values (each could be a phrase). When a document contains several fragments for a single-valued slot, these fragments will have the same value. Therefore, given a job posting, our system picks at most one fragment for each of these slots. Other fields such as `language` (programming language skills required) or `platform` (platforms used in this computer related job) may take multiple values. For these fields, our system may pick several different fragments for the same slot.

5.2. EXTRACTING RELATIONAL FEATURES

The basic strategy of our IE solution is to learn a classifier that discriminates a specific desired fragment. To model this problem as a supervised learning problem we first generate examples for each type of fragment. This is done by:

1. Identifying candidate fragments in the document. Suppose a document consists of tokens $t_1, t_2, \dots, t_n$, indexed by their positions in the document. Any string of consecutive tokens $t_i, t_{i+1}, \dots, t_j$, where $1 \leq i \leq j \leq n$, constitute a candidate fragment. In practice, we limit the size of these fragments to a constant l (i.e. $j - i \leq l$). Note that fragments may overlap, and only a small number of them are meaningful fragments.
2. For each candidate fragment, re-represent it as an example which consists of all active features extracted for this fragment using the predefined RGFs.

Let $f = (t_i, t_{i+1}, \dots, t_j)$ be a fragment. Our RGFs are defined to extract features from three regions: left window $(t_{i-w}, \dots, t_{i-1})$, target fragment $(t_i, \dots, t_j)$, and right window $(t_{j+1}, \dots, t_{j+w})$, where w is the window size. Figure 2 demonstrates one example, in which the three framed boxes represent left window, target region, and right window ($w = 4$) respectively.

The domain formed by these three regions contains two types of elements – word elements (e.g., $t_{i-w}, \dots, t_{j+w}$) 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. Examples of RGFs used in the experiments are shown in Section 5.4.

5.3. APPLYING PROPOSITIONAL LEARNING ALGORITHMS

Given RGFs and an instance as describe above, an example is a list of the formulae that are active in this instance. Once examples are generated, we can apply any propositional learning algorithms to learn on them. The algorithms we tested are the SNoW learning architecture and the naive Bayes ().

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 in a variety of tasks in natural language and visual processing (Golding and Roth, 1999; Roth et al., 2002). 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.

One potential computational difficulty of any propositionalization approach is that it generates a huge number of features, and most of them occur very few times in examples. Eliminating some infrequent features speeds up the learning process, but somewhat sacrifices the performance (Golding and Roth, 1999). To test its effect, we conducted a series of experiments that eliminate different portion of features according to the appearing frequency.

Another potential computational difficulty comes from the essence of IE tasks. Since any fragment from a document can be of potential interest as the field for some specific slot, a large number of examples need to be generated in order to represent all possible fragments in a document. Among them, negative examples (irrelevant fragments) vastly outnumber positive examples (fragments that represent legitimate slots). In the seminar announcements data set, for example, the average length of an article is 200 words. If we limit the maximum length of fragments to 14 words, then each article would generate about 3300 candidate fragments, out of which about 10 fragments (i.e. 0.3%) represent legitimate slots. To handle this, we consider several methods to eliminate many of the negative training examples⁴. This reduces the learning time, but may have adverse impact on performance. To test this we compare several methods, including randomly sub-sampling the negative examples and using only positive examples. Our main approach to address this issue, is describe next.

⁴ Note that since we consider the *recognition* of fragments of interest and their *classification* at the same time, an approach that only uses positive examples, that is, only fragments of interest, will significantly degrades recognition performance.

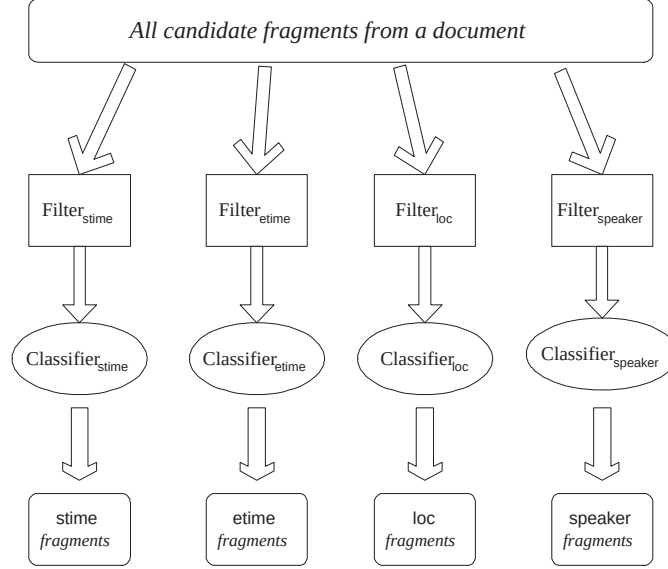


Figure 3. Two-stage architecture for seminar announcements data

5.4. TWO-STAGE ARCHITECTURE

In order to efficiently eliminate negative examples without a degrading accuracy we suggest to address the IE task using two learning stages. The same classifier, SNoW, is used in both stages, but in a slightly different way.

SNoW is typically used as a *predictor*, when it uses a winner-take-all mechanism over the activation value of the target classes. Here we suggest to rely directly on the activation value it outputs, computed using a sigmoid function over the linear sum. The normalized activation value increases with the confidence in the prediction and can thus be used as a density function. The success of our two-stage architecture relies heavily on the robustness of this measure. The two stages are:

1. Filtering: reducing the amount of candidate fragments to a small number, and
2. Classification: identifying the correct fragment from the remaining fragments and classifying it into one of the target classes.

Figure 3 illustrates how this architecture is applied to the seminar announcements data. All the candidate fragments are sent to four filters, one for each slot type. Four corresponding binary classifiers are then used respectively to determine if the remaining fragments they seen in their input belong to the appropriate slots.

Intuitively, this architecture increases the expressivity of our classification system. Moreover, given the relatively small number of positive examples and the existence of simply learned, robust learner that eliminates most of the negative examples, a large number of irrelevant features is also eliminated, an important issue given the small data set.

5.4.1. *Filtering*

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: the fragment does not contain an active feature that should be active on positive examples.
2. General Classifier: the fragment's confidence value is below the threshold.

For criterion 1, it turns out that there exists some features that are (almost) always active on positive examples. For example, in our experiments, the length of fragments satisfies this. $len(fragment) < 7$ is always satisfied by `stime` fragments in seminar announcements. Also, the `title` fragment in job postings must contain a word that is a noun.

For criterion 2, implemented using SNoW, relying on its robust confidence estimation, the problem becomes finding the right threshold. The activation values on SNoW, converted via a sigmoid function are just used to measure how likely a fragment is to represent a specific slot. The minimum activation values of the positive examples in training data are used as thresholds (for the different types of slots). Examples with low activation values are filtered out.

The two stages also differ in the RGFs used. Only the following crude RGFs are using in 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.

The way these RGFs are used to extract features that represent a fragment is demonstrated in Figure 4.

5.4.2. *Classifying*

Fragments that survive the filtering stage are then classified using a second SNoW classifier, for each slot. First, an additional collection of RGFs is

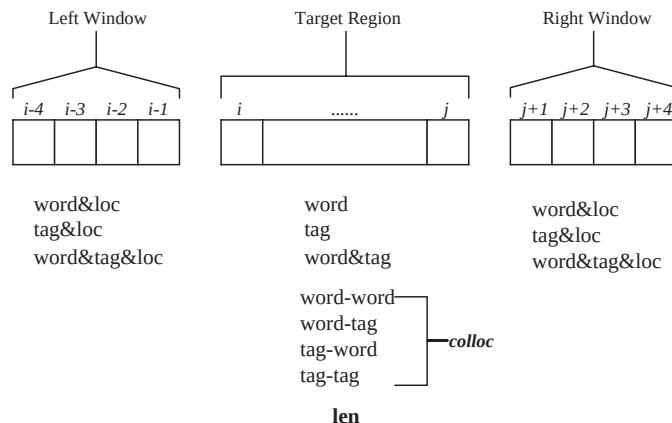


Figure 4. Graphical Illustration of a set of RGFs

applied to enhance the representation of the candidate fragments, and thus allow for more accurate classification. As before, in training, the remaining fragments (which are annotated) 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. The RGFs used by the classifiers in this stage are described in the appendix.

A fragment may be considered by several classifiers; it is classified as type t if the t th classifier decides so. For single-value slots, at most one fragment of type t is chosen in each article based on the activation value of the corresponding classifier.

6. Experimental Results

Our paradigm has been tested on two data sets, *seminar announcements* and *computer-related job postings*, which were used previously to test several ILP-based IE systems (e.g. RAPIER (Califf and Mooney, 1999), SRV (Freitag, 2000), and WHISK (Soderland, 1999)). We compare our overall results to theirs in sections 6.1 and 6.2. We also report the results on experiments on the effects of pruning features and examples in section 6.3.

6.1. SEMINAR ANNOUNCEMENTS

Our experiments use the same data, methodology and evaluation metrics used by several ILP-based IE systems in previous works. The data (485 documents) is randomly split into two sets of equal size, one for training and the other for testing. The reported results are an average of five runs. As usual, the

Table I. Results for seminar announcements task

	stime			etime			loc			speaker		
System	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
2-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
2-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
1-SNoW-IE	97.9	89.7	93.0	89.2	91.7	90.3	93.3	53.9	68.2	82.2	63.9	71.9
1-NB-IE	97.6	97.6	97.6	79.7	92.2	85.5	40.4	34.7	37.4	16.5	13.6	14.9
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

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, 2-SNoW-IE (2-stage architecture SNoW Learning), are shown in the first row of table I. As a comparison, we also present the results of our other two systems, 1-SNoW-IE and 1-NB-IE, in the second and third rows. 1-SNoW-IE learns the classifier based only on examples generated for the first stage. This provides some ideas of how much the overall performance benefits from the two-stage architecture. In addition to the SNoW learner, we have also experimented with a second propositional algorithm, the naive Bayes (1-NB-IE) algorithm. NB was used on exactly the same set of features (same examples) that were generated using our relational paradigm for 1-SNoW-IE, and in exactly the same way.

Along with the results are 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 do not differ only in the algorithms but also in the way they represent the domain and generate examples, that is, their features. The words in the documents are used by all systems. Part-of-speech tags are used both in RAPIER-WT and our systems; SRV uses other predicates that capture POS information to some extent. A version of RAPIER also uses 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.

Overall, 2-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 types of relational features as do the ILP-based systems,

Table II. Filtering efficiency on seminar announcement data

slot	% examples that survive filtering				% positive examples lost	
	training		testing		training	testing
stime	1.96%	(13682/698136)	1.94%	(13468/696669)	0.16%	2.43%
etime	0.68%	(4731/698136)	0.69%	(4766/696669)	0.00%	2.50%
location	2.92%	(20345/698136)	2.87%	(20011/696669)	0.78%	9.88%
speaker	6.50%	(45240/698136)	6.03%	(42106/696669)	0.70%	8.97%

only that instead of a collection of conjunctive rules over these, it is represented as a linear function. Examples of the dominant features represented in our final hypotheses appear in the appendix.

The results above also exhibit the enhancement in performance due to the two state architecture of the SNoW-IE system. As a side note, the ability to use this architecture provide another indication of the flexibility of the approach and the advantage of learning with propositional means. Table II gives some insight into the second classification stage by showing the average performance of the filtering stage. The first two columns show the ratio of examples that survive the filtering stage in training and testing respectively. Generally, the number of examples is reduced by 93.5% to 99.3%. The relatively smaller number of remaining fragments, makes it computationally possible to generate complex relational features in the second stage. The third and fourth columns show the fraction of positive examples that are filtered out in the training and testing sets. These fragments do not even reach the second stage. Note, however, that an article may contain more than one fragment that represents a given slot; it is therefore sometimes still possible for the classifier to pick the correct slot. That is, the reduction in the performance is lower than the loss due to the filter.

Although the results of 1-NB-IE are not as good as those of 1-SNoW-IE – the quality of the classifier is certainly an important issue – it is credible; the experiments with a second propositional algorithm exhibit the fact that our relational paradigm is a general one. As indicated in (Craven and Slattery, 2000; 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 favorably with ILP methods.

6.2. COMPUTER-RELATED JOB POSTINGS

For computer-related job posting data, we used the same data and methodology as with Rapier (Califf and Mooney, 1999). The result reflect 10-fold validation on 300 newsgroup documents. The results show that 2-SNoW outperforms Rapier in most of the slots, and the overall result is better. We note

Table III. Results for computer-related job postings task

	2-SNoW			1-SNoW			1-NB			Rapier		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
id	99.7	99.7	99.7	96.9	96.0	96.4	93.2	93.0	93.1	98.0	97.0	97.5
title	62.0	45.9	52.7	63.6	34.8	45.0	39.4	39.2	39.3	67.0	29.0	40.5
salary	89.3	61.6	72.9	71.1	39.2	50.5	10.6	86.6	18.9	89.2	54.2	67.4
company	89.7	65.1	75.4	59.5	66.9	62.9	56.1	67.1	61.1	76.0	64.8	70.0
recruiter	89.4	81.5	85.3	75.6	74.7	75.2	47.5	53.1	50.1	87.7	56.0	68.4
state	91.7	91.8	91.7	85.7	95.7	90.4	87.2	91.8	89.5	93.5	87.1	90.2
city	90.1	87.9	89.0	80.2	91.1	85.3	76.7	87.3	81.6	97.4	84.3	90.4
country	95.6	95.4	95.5	93.5	92.5	93.0	74.4	93.8	83.0	92.2	94.2	93.2
language	83.5	81.6	82.5	55.0	74.8	63.4	29.9	77.7	43.2	95.3	71.6	81.8
platform	74.4	73.8	74.1	68.2	57.6	62.5	43.3	71.7	54.0	92.2	59.7	72.5
application	84.7	47.5	60.9	51.8	45.0	48.2	43.9	63.6	52.0	87.5	57.4	69.3
area	63.5	43.4	51.6	39.3	36.5	37.9	23.2	51.5	32.0	66.6	31.1	42.4
req_yrs_exp	90.2	78.4	83.9	78.2	69.1	73.4	32.2	47.1	38.2	80.7	57.5	67.2
des_yrs_exp	75.3	83.1	79.0	56.2	68.1	61.6	30.0	56.1	39.1	94.6	81.4	87.5
req_degree	86.3	80.9	83.5	65.2	74.0	69.3	67.3	75.6	71.2	88.0	75.9	81.5
des_degree	81.0	48.8	60.9	26.2	20.2	22.8	34.1	65.5	44.9	86.7	61.9	72.2
post_date	99.0	99.3	99.2	99.0	99.3	99.0	99.7	99.3	99.3	99.3	99.7	99.5
total	83.4	72.4	77.6	65.3	66.3	65.8	40.2	71.7	51.5	89.4	64.8	75.1

that this data set is very sparse, and for several of the slots the number of examples is very small. This results in fairly high variance in the accuracy of slots like `des_yrs_exp` and `des_deg` deeming the results not statistically significant.

6.3. PRUNING FEATURES AND NEGATIVE EXAMPLES

As mentioned above, our approach deals with large scale learning both in terms of the number of examples and the number of features. First, the modeling of the IE task as a supervised learning task results in the generation of a large number of negative examples. Second, our feature generation approach yields a very large number of potential features. The following two experiments study the effects of pruning examples and features on the performance of our system.

Figure 5 presents the effect of pruning features. While learning, SNoW builds a histogram that represents the number of feature that occur k times, for all values of k . Pruning features is done in a relative fashion. The horizontal axis in Fig. 5 represents the percentage of the tail of this histogram that was

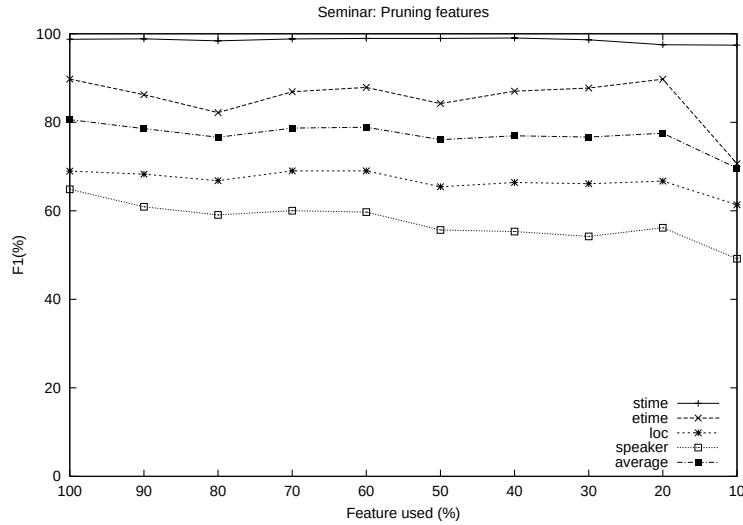


Figure 5. Seminar: 10-fold cross validation: relative feature elimination (-e) in SNoW

eliminated. That is, all features that occurred less than j times are eliminated, as long as the total number of features eliminated does not exceed $x\%$ of the features. Training continues for one more round over all the examples after features are eliminated (see details in (Carlson et al., 2001)).

The result show that removing features this way (as opposed to pruning based on weights, for example (Carlson et al., 2001)), does not degrade the results significantly, for all slots, as long as 20% of the frequent features are maintained.

The performance on the *speaker* is the only exception, since it declines gradually as features are eliminated. This may explains why our system performs so well on this difficult slot. For this complex slot, many features that represent “exceptions” needs to be taken into account to guarantee good prediction. Unlike traditional ILP learners, learning via a linear threshold function is able to make predictions taking into account a large number of relational features. This can be contrasted with slots like *stime* and *etime* that are easy and on which the performance does no suffer at all from pruning a large number of the features.

Figure 6 shows the effect of eliminating negative examples from the training set. As discussed earlier, in IE tasks, the negative examples significantly outnumber the positive examples. This has significant computational consequences. This experiment address the question of whether all the negative examples are needed.

Several “sophisticated” methods can be used to address this problem, attempting to eliminate only negative examples that are “easy”, and thus may

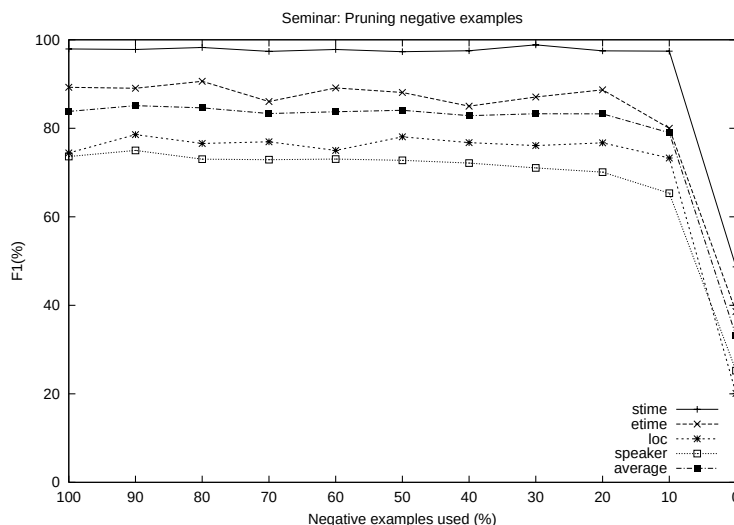


Figure 6. Seminar: 10-fold cross validation: negative examples elimination (-N) in SNoW

not contribute to learning. In this experiment we have considered a very simple minded approach – we randomly eliminate $x\%$ of the negative examples, from each slot separately.

The results are very significant and show that this strategy is very useful – there is no adverse effect of the results as long as sufficiently many negative examples, about 20% of them, are retained. The main lesson is that the performance degrades significantly only when (almost) only positive examples are used. The reason is that in this case, the learning algorithm has no way to identify irrelevant fragments.

7. 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 or concepts of interest 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 thoroughly recently. Our paradigm was used to develop a new approach for learning in the context of information extraction, and we have shown that it outperforms existing methods, while being several orders of magnitude more efficient.

We believe that this work opens up several directions for further work both in the direction of relational learning and knowledge representation, and as an approach for further research toward practical and efficient solutions to NLP and IE problems.

Appendix

A. RGFs added in the second stage classifiers

The RGFs added for seminar announcement data include:

1. For `stime` and `etime`:
`conjunct[word&loc(-1)[l_window], word&loc[r_window]],`
`conjunct[word&loc(-1)[l_window], tag&loc[r_window]]`
2. For `location` and `speaker`:
`conjunct[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 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.

The RGFs added for job posting data include:

```
phAllCap[targ], phAllNotNum[targ], phAllWord[targ]
conjunct[word&loc(-2)[l_window], tag&loc(1)[r_window]]
conjunct[tag&loc(-2)[l_window], tag&loc(1)[r_window]]
conjunct[word&loc(-1)[l_window], word&loc[r_window]]
conjunct[word&loc(-1)[l_window], tag&loc[r_window]]
phAllCap[targ], phAllNotNum[targ], phAllWord[targ]
conjunct[word&loc(-2)[l_window]; coloc(word,word,word)[targ]; tag&loc(1)[r_window]]
conjunct[word&loc(-2)[l_window]; coloc(word,word)[targ]; tag&loc(1)[r_window]]
conjunct[word&loc(-2)[l_window]; word[targ]; tag&loc(1)[r_window]]
```

Among these RGFs, `phAllCap[targ]`, `phAllNotNum[targ]` and `phAllWord[targ]` check if the words in the target region are all capitalized, without numbers, or without numbers and symbols respectively. `coloc(word,word,word)[targ]` extracts all the trigrams in the target region, and `coloc(word,word)[targ]` extracts the bigrams.

Table IV. Some dominant features for slot **recruiter** in computer-related job posting data

Relational Features	Predicate Logic Translation
w[Ltd]-w[.]	$\exists x, y \quad Tr(x) \wedge Tr(y) \wedge Before(x, y) \wedge Word(x, "Ltd") \wedge Word(y, ".")$
w[*LCS]&t[*NNP]	$\exists x \quad Tr(x) \wedge LocInTr(x, 1) \wedge Word(x, "LCS") \wedge Tag(x, "NNP")$
w[*_Staffing]&t[*_NNP]	$\exists x \quad Tr(x) \wedge LocInTr(x, 2) \wedge Word(x, "Staffing") \wedge Tag(x, "NNP")$
w[LCS]&t[NNP]	$\exists x \quad Tr(x) \wedge Word(x, "LCS") \wedge Tag(x, "NNP")$
w[Global]-w[Staffing]	$\exists x, y \quad Tr(x) \wedge Tr(y) \wedge Before(x, y) \wedge Word(x, "Global") \wedge Word(y, "Staffing")$
w[*_Austin]	$\exists x \quad Tr(x) \wedge LocInTr(x, 2) \wedge Word(x, "Austin")$
w[:*]-t[*_NNPS]	$\exists x, y \quad BeforeTr(x, 1) \wedge Word(x, ":") \wedge Tr(y) \wedge LocInTr(y, 2) \wedge Tag(y, "NNPS")$
w[at*]-w[*.]	$\exists x, y \quad BeforeTr(x, 1) \wedge Word(x, "at") \wedge Tr(y) \wedge LocInTr(y, 1) \wedge Word(y, ".")$
w[:*]-w[*_Services]	$\exists x, y \quad BeforeTr(x, 1) \wedge Word(x, ":") \wedge Tr(y) \wedge LocInTr(y, 2) \wedge Word(y, "Services")$
phLen[4]	$TrLen(4)$

B. Dominant features

The following two table list some of the dominant features used by our predictors. The left column describe the features in the FEX language we use, and the right column describes them using standard FOL predicates.

The intended meaning of the predicates used is as follows:

- Tr : in the target region
- $Before$: one word is right before the other
- $Word$: spelling of the word
- Tag : part-of-speech tag of the word (The exception is “BOL”, which means the word is in the beginning of the line.)
- $LocInTr$: the location in the target region
- $BeforeTr$: the location before the target region
- $BeforeTr$: the location after the target region
- $TrLen$: length of the target region

These tables by no means represent the learned hypothesis, which makes use of a very large number of features. The total number of features used in the first learning stage was 421300 for the seminar task and 240574 for the jobs tasks. The total number of features in the second stage varies from slot to slot. For the speaker slot the number was 245081; for the recruiter slot it was 143104.

Table V. Some dominant features for slot **speaker** in seminar announcement data

Relational Features	Predicate Logic Translation
w[*]-t[NNP]-t[*-LRB-]	$\exists x, y, z \quad BeforeTr(x, 1) \wedge Word(x, ".") \wedge Tr(y) \wedge Tag(y, "NNP") \wedge$ $AfterTr(z, 1) \wedge Tag(z, "-LRB-")$
t[NNP]-t[NNP]	$\exists x, y \quad Tr(x) \wedge Tr(y) \wedge Before(x, y) \wedge Tag(x, "NNP") \wedge Tag(y, "NNP")$
t[BOL]-t[NNP]	$\exists x, y \quad Tr(x) \wedge Tr(y) \wedge Before(x, y) \wedge Tag(x, "BOL") \wedge Tag(y, "NNP")$
w[*]-t[NNP]-t[*VBZ]	$\exists x, y, z \quad BeforeTr(x, 1) \wedge Word(x, ".") \wedge Tr(y) \wedge Tag(y, "NNP") \wedge$ $AfterTr(z, 1) \wedge Tag(z, "VBZ")$
w[*Talk]&t[*NN]	$\exists x \quad Tr(x) \wedge LocInTr(x, 1) \wedge Word(x, "Talk") \wedge Tag(x, "NN")$
w[*Professor]&t[*BOL]	$\exists x \quad Tr(x) \wedge LocInTr(x, 1) \wedge Word(x, "Professor") \wedge Tag(x, "BOL")$
w[*]-t[NNP]-t[NNP]-t[*IN]	$\exists w, x, y, z \quad BeforeTr(w, 1) \wedge Word(w, ".") \wedge Tr(x) \wedge$ $Tag(x, "NNP") \wedge Tr(y) \wedge Tag(y, "NNP") \wedge$ $Before(x, y) \wedge AfterTr(z, 1) \wedge Tag(z, "IN")$
w[:*]-t[NNP]-t[*NN]	$\exists x, y, z \quad BeforeTr(x, 1) \wedge Word(x, ":") \wedge Tr(y) \wedge Tag(y, "NNP") \wedge$ $AfterTr(z, 1) \wedge Tag(z, "NN")$
w[:*]-t[NNP]-t[NNP]-t[*VBG]	$\exists w, x, y, z \quad BeforeTr(w, 1) \wedge Word(w, ":") \wedge Tr(x) \wedge$ $Tag(x, "NNP") \wedge Tr(y) \wedge Tag(y, "NNP") \wedge$ $Before(x, y) \wedge AfterTr(z, 1) \wedge Tag(z, "VBG")$
w[appointment_*]-t[NNP]-t[*VBP]	$\exists x, y, z \quad BeforeTr(x, 2) \wedge Word(x, "appointment") \wedge Tr(y) \wedge$ $Tag(y, "NNP") \wedge AfterTr(z, 1) \wedge Tag(z, "VBP")$

References

- Califf, M. and R. Mooney: 1999, 'Relational Learning of Pattern-Match Rules for Information Extraction'. In: *National Conference on Artificial Intelligence*.
- Carleson, A., C. Cumby, J. Rosen, and D. Roth: 1999, 'The SNoW Learning Architecture'. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department.
- Carlson, A. J., J. Rosen, and D. Roth: 2001, 'Scaling Up Context Sensitive Text Correction'. In: *Proceedings of the National Conference on Innovative Applications of Artificial Intelligence*. pp. 45–50.
- Cohen, W.: 1995, 'PAC-learning recursive logic programs: Negative result'. *Journal of Artificial Intelligence Research* 2, 541–573.
- Cohen, W. and D. Page: 1995, 'Polynomial Learnability and Inductive Logic Programming: Methods and Results'. *New Generation Computing* pp. 369–409.
- Craven, M. and S. Slattery: 2000, 'Relational Learning with Statistical Predicate Invention: Better Models for Hypertext.'. *Machine Learning*. To appear.
- Cumby, C. and D. Roth: 2000, 'Relational Representations that facilitate learning'. In: *Proc. of the International Conference on the Principles of Knowledge Representation and Reasoning*. pp. 425–434.
- Cussens, J.: 1997, 'Part-of-Speech Tagging Using Progol'. In: *International Workshop on Inductive Logic Programming*. Prague, Czech Republic, pp. 93–108, Springer. LNAI 1297.
- DARPA: 1995, *Proceedings of the 6th Message Understanding Conference*. Morgan Kaufman.

- Freitag, D.: 2000, 'Machine Learning for Information Extraction in Informal Domains'. *Machine Learning* **39**(2/3), 169–202.
- Golding, A. R. and D. Roth: 1999, 'A Winnow based approach to Context-Sensitive Spelling Correction'. *Machine Learning* **34**(1-3), 107–130. Special Issue on Machine Learning and Natural Language.
- Kharon, R., D. Roth, and L. G. Valiant: 1999, 'Relational Learning for NLP using Linear Threshold Elements'. In: *Proc. of the International Joint Conference of Artificial Intelligence*. pp. 911–917.
- Kietz, J. and S. Dzeroski: 1994, 'Inductive Logic Programming and Learnability'. *SIGART Bulletin* **5**(1), 22–32.
- Kramer, S., N. Lavrac, and P. Flach: 2001, 'Propositionalization Approaches to Relational Data Mining'. In: S. Dzeroski and N. Lavrac (eds.): *Relational Data Mining*. Springer-Verlag, pp. 262–291.
- Lavrac, N., S. Dzeroski, and M. Grobelnik: 1991, 'Learning Nonrecursive Definitions of Relations with LINUS'. In: *Proceedings of the European Working Session on Learning : Machine Learning (EWSL-91)*, Vol. 482 of *LNAI*. Porto, Portugal, pp. 265–281, Springer Verlag.
- Littlestone, N.: 1988, 'Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm'. *Machine Learning* **2**, 285–318.
- Lloyd, J. W.: 1987, *Foundations of Logic Programming*. Springer-verlag.
- Mooney, R. J.: 1997, 'Inductive logic programming for natural language processing'. In: *Proceedings of the 6th International Workshop on Inductive Logic Programming (ILP-96)*, Vol. 1314 of *LNAI*. Berlin, pp. 3–24, Springer.
- Muggleton, S. and L. De Raedt: 1994, 'Inductive Logic Programming: Theory and Methods'. *Journal of Logic Programming* **20**, 629–679.
- Quinlan, J. R.: 1990, 'Learning Logical Definitions from Relations'. *Machine Learning* **5**, 239–266.
- Riloff, E.: 1993, 'Automatically Constructing a Dictionary for Information Extraction Tasks'. In: *National Conference on Artificial Intelligence*. Menlo Park, CA, USA, pp. 811–816, AAAI Press.
- Roth, D.: 1998, 'Learning to Resolve Natural Language Ambiguities: A Unified Approach'. In: *National Conference on Artificial Intelligence*. pp. 806–813.
- Roth, D., M.-H. Yang, and N. Ahuja: 2002, 'Learning to Recognize Objects'. *Neural Computation* **14**(5), 1071–1104.
- Soderland, S.: 1999, 'Learning Information Extraction Rules for Semi-Structured and Free Text'. *Machine Learning* **34**(1-3), 233–272.
- Soderland, S. and W. Lehnert: 1994, 'Wrap-Up: a trainable discourse module for information extraction'. *Journal of Artificial Intelligence Research* **2**, 131–158.

