Propositionalization of Relational Learning: An Information Extraction Case Study[a]

Dan Roth DANR@UIUC.EDU

Department of Computer Science

University of Illinois at Urbana-Champaign

3322 Siebel Center

201 N. Goodwin Avenue

Urbana, IL 61801-2302, USA

Phone: +1-217-244-7068

Fax: +1-217-265-6591

Wen-tau Yih[b] SCOTTYIH@MICROSOFT.COM

Microsoft Research

1 Microsoft Way

Redmond, WA 68052-6399, USA

Phone: +1-425-706-0579

Fax: +1-425-936-7329

a. An earlier version of this paper has appeared in Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01).

b. Most of the work was done when the author was at the University of Illinois at Urbana-Champaign.

Abstract

This paper develops a new propositionalization approach for relational learning which allows for efficient representation and learning of relational information using propositional means. We develop a relational representation language, along with a relation generation function that produces features in this language in a data driven way; together, these allow efficient representation of the relational structure of a given domain in a way that 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 framework 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 Inductive Logic Programming (ILP) approaches and existing propositionalization methods and, as a result, it enjoys several advantages over it. In particular, our approach is both scalable and flexible, which allows the use of efficient propositional algorithms, including probabilistic algorithms, within it. The proposed approach is evaluated on several relational problems, focusing on an important and relation-intensive task – information extraction – and shown to outperform existing methods based on relational learning.

Key words: Information Extraction, Machine Learning, Feature Extraction, Inductive Logic Programming, Propositionalization

1. Introduction

Relational learning is the problem of learning structured concept definitions from structured examples. The data in 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 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 identifying noun phrases in a sentence, detecting faces in an image, and 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.

A natural way to approach these tasks might be an Inductive Logic Programming (ILP) approach. ILP is a subfield of machine learning that addresses relational learning. In principle, these methods allow induction over relational structures and unbounded data structures. However, studies suggest that unless the rule representation is restricted, the learning problem is intractable (Kietz & Dzeroski, 1994; Cohen, 1995; Cohen & Page, 1995). Thus, different heuristics are used in ILP methods (Muggleton & De Raedt, 1994; Cussens, 1997; Quinlan, 1990) to control the search, and learn the concept efficiently.

Propositionalization (Lavrac, Dzeroski, & Grobelnik, 1991; Lavrac & Dzeroski, 1994; Flach & Lachiche, 1999; Lavrac & Flach, 2001; Kramer, Lavrac, & Flach, 2001), a framework that transforms original relational representations to propositional features, is a different way to conduct relational learning. In this approach, relational information is first captured and stored as propositions, according to some predefined declarative bias. Propositional learning algorithms are then applied to learn using these *new* features. Although propositionalization has some inherent limitations, such as a claimed inability to learn recurrent definitions, it enjoys several advantages over traditional ILP methods. In particular, it allows the use of general purpose propositional algorithms that have been well studied, including probabilistic algorithms, but nevertheless learns relational representations.

In this paper, we propose a relational learning framework that is best viewed as a specific propositionalization method. At the center of our framework 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 (especially those that can handle large feature space when each example has relatively few active features), but results in relational concept descriptions as outcome.

Our framework decouples the feature extraction stage from the learning stage thus providing a generic and flexible language for defining declarative bias for propositionalization. A design with careful consideration of efficiency allows feature types defined by our language to be extracted rapidly. Finally, although not required by the framework, the multiclass propositional learning algorithm we use in conjunction with our propositionalization method is SNoW[1]. SNoW handles variable length examples and uses a 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.

We apply our learning framework to an important Natural Language Processing task – Information Extraction (IE). IE is a text processing task that 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. from news articles (DARPA, 1995). Several relational learning based methods have been proposed for this shallow text processing problem (e.g., (Califf & Mooney, 1999; Freitag & McCallum, 2000; Craven & Slattery, 2001)), making this an interesting task to evaluate our learning framework on.

The next subsection provides a high-level view of our learning framework; a more detailed description of the information extraction problem is given in Sec. 4.

1. Available at http://l2r.cs.uiuc.edu/~cogcomp.

ID	Predicate logic representation	Truth value
p1	*Father*($\texttt{John}, \texttt{Tom}$)	1
p2	*Mother*($\texttt{Tom}, \texttt{Mary}$)	0
p3	*Father*($\texttt{John}, \texttt{Tom}$) $\wedge$ *Mother*($\texttt{Mary}, \texttt{Tom}$)	1
p4	($\exists x$ *Father*($\texttt{John}, \texttt{x}$)) $\vee$ ($\exists y$ *Mother*($\texttt{y}, \texttt{Tom}$))	1
...	...	...

Table 1: An example of propositionalization in the kinship domain.

1.1 The Learning Framework

Our learning framework attempts to learn classifiers for relational learning problems by utilizing propositional learning algorithms. The key to our technique is the decoupling of a data driven feature extraction stage for propositionalization and a feature efficient learning stage.

Propositionalization is a transformation that maps a relational representation to features in propositional form (Kramer et al., 2001). These features capture the relational information and are usually stored as attributes in a vector, which forms an example provided to propositional learners. Consider, for example, the classical ILP problem of identifying kinship relations (Hinton, 1986; Quinlan, 1990). Suppose `Tom` is the only child of `John` and `Mary`. Numerous relational features may be extracted as shown in Table 1.

Our propositionalization approach makes use of relation generation functions (RGF) which define "types" of features to be extracted. Along with a specific target property of interest – a "label" (e.g., an attribute of the input or a relation that holds in the input), an RGF maps a relational representation of the data to a propositional representation with respect to this target.

In our framework, as in ILP, observations in the domain are assumed to be represented as a collection of predicates that hold in elements of the domain. Given this assumption, the task then becomes that of producing a classifier that predicts whether a given predicate (target property) holds for some particular observation. For example, we may wish to predict for some domain element X (e.g., a phrase in a sentence), if the predicate $location(X)$, which indicates that the phrase X represents a location, holds. Similarly, we may want to decide if the predicate $author\_of(X,Y)$, which indicates that X is the author of Y, holds, given some domain elements X, Y.

To accomplish this task using propositional learning algorithms, we must generate examples in the form of lists of propositions (features) representing information that might be relevant to the predicate to be learned. Propositions of this form may either be fully ground as in the predicate $location(US)$ or predicates that are individually quantified as in the expression $(\exists X father(John, X)) \wedge (\exists Y mother(Y, Tom))$.

Each list of propositions, generated to represent the input observation with respect to the target element, serves as a positive example for a predicate that holds in the target, and can also serve as a negative example for the existence of "competing" predicates (i.e., different classes). These examples are used to train a classifier that defines a mapping from instances in this feature space to one of the possible values the target predicate can take.

In order to facilitate efficient feature generation that captures the relational information among the domain elements, we define a language that extends ideas presented first in (Cumby & Roth, 2000; Khardon, Roth, & Valiant, 1999). To apply this language, the relational data is first re-represented as a graph, where each node is an element, and the edges define relations between two elements. When learning whether a predicate holds among particular elements of interest, features are generated from a *focus* of the attention region "near" the predicate to be learned. Therefore, features produced depend on, and can be differentiated by, the target with respect to which they are produced. Efficient feature extraction is achieved by restricting the syntax of the propositions. In particular, the *scope* of a quantifier is always a single predicate. More expressive propositions are allowed by using the graphical structure of the domain representation.

We develop the notion of *relation generation functions*, which allows us to define the "type" of features we would like to generate in order to abstract the properties of the domain and its relational structure, relative to a specific element, and represent them as a propositional example in a data driven way.

The feature extraction method we present operates with the so-called closed-world assumption: it generates only the features judged to be active in the example by the relation generating function; other features are judged to be inactive (i.e., false) and are not generated. Since 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. As a result, 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; therefore, our learning algorithm should be able to learn well in the presence of a large number of irrelevant features.

SY: Please see if this paragraph is appropriate here and also check whether it's too rough.

Our graphical representation may be contrasted to SUBDUE (Cook & Holder, 1994), which is a system that aims to discover common substructures in structural data. SUBDUE searches for same or similar subgraphs as the output. Therefore, its whole process can be treated as a feature selection process that operates directly on the original structural data representation, where in our propositionalization method the features are generated through the declarative language directly, and the selection process can be done separately if needed.

Our framework is similar to (Bournaud, Courtine, & Zucker, 2003), but the features generated are constrained by a combination of the language bias and the graphical structure. In particular, our framework differs from existing propositionalization methods in two major respects. First, we define a language that allows users to parameterize preferences when constructing relational features. The language is designed with serious consideration of efficiency. Specifically, only local binding of quantifiers is allowed, in order to ensure rapid feature extraction. With the help of operators over the graphical structure, however, it is still possible to express complex features. These features are not limited to conjunctions but rather constitute a much richer, yet restricted set of FOL expressions. Second, the mapping from domains to graphs is very flexible, and allows multi-labeled nodes and edges. For instance, although in the information extraction problem addressed in this paper, the graph is basically a linked list, this framework has been applied to other problems such as the kinship problem, problems in computational chemistry and other natural language problems, where domain elements are described as general graphs (Cumby & Roth, 2003a). For more detailed survey and comparison on different propositionalization approaches, see (Kramer et al., 2001) and (Krogel, Rawles, Zelezny, Flach, Lavrac, & Wrobel, 2003).

The fundamental difference between the traditional ILP approach, including most earlier propositionalization schemes, and the propositionalization approach proposed here, is their different behaviors with respect to generating “features”. Previously, “features” were generated as part of the search procedure in an attempt to find good bindings. Ideally, a generic ILP algorithm may determine good features itself without much guidance from users and output the learned concept in FOL. In a propositionalization approach like ours, the process of feature extraction is decoupled from the learning stage. “Types” of features are proposed by the system designer, via the RGFs mechanism; then, features are generated based on these in a data driven way, once data is observed. The level of abstraction is also determined when designing the RGFs: some of these relational features are grounded and some have free variables in them. With the help of RGFs defined in our language, users can easily define the “types” of relational features that might be important to learning. Learning is done only after the features are extracted, namely when the data is re-represented via a feature-based representation. When propositional learning algorithms like Perceptron or naive Bayes are used, appropriate weights are learned for the features “in parallel”, which makes the learning process more efficient.

Our learning stage uses a feature-efficient propositional learning algorithm, SNoW, to learn a function that represents the target property in terms of the extracted features. SNoW is a multiclass propositional classifier suitable for learning in a high dimensional feature space, especially when the input examples are sparse. The SNoW architecture is a network of threshold gates. Nodes in the first layer of the network are allocated to input features in a data-driven way, given the variable length examples. Target classes are represented by nodes in the second layer. Links from the first to the second layer have weights; each target node is thus defined as a (linear) function of the lower level nodes.

Learning in SNoW proceeds in an on-line fashion. Every example is treated independently by each target subnetwork. Every labeled example is treated as positive to the target node corresponding to its label, and as negative to all others, in a one-vs-all manner. Each example is used once to refine the weights and then discarded. At prediction time, given an input example which activates a subset of the input nodes, the information propagates through all the subnetworks; the target node that produces the highest activation value determines the prediction (i.e., a winner-take-all policy[2]).

2. SNoW actually supports a more general multi-class paradigm (Har-Peled, Roth, & Zimak, 2002) that we do not use here.

SNoW employs several linear update rules including Perceptron, naive Bayes, and Winnow. We use here a variation of the Winnow (Littlestone, 1988) update rule to learn the features' weights. Winnow is a mistake driven on-line algorithm that is similar to Perceptron (Rosenblatt, 1962), but updates its weights in a multiplicative way. The key advantage it has is that the number of examples required to learn the target function grows linearly with the number of relevant features, but only logarithmically with the total number of features. SNoW has been shown to be successful in large scale NLP and IE problems (e.g., (Roth, 1998; Roth & Yih, 2001)). More discussion of SNoW can be found in (Carlson, Cumby, Rosen, & Roth, 1999; Roth, 1998; Carlson, Rosen, & Roth, 2001).

1.2 Structure of the Paper

The rest of the paper is organized as follows. Sec. 3 describes the main technical contribution of this paper – a relational language for extracting features and a mechanism that allows for a data driven generation of features. This paper evaluates the proposed framework in the domain of Information Extraction. In Sec. 4 we describe the IE system we have developed based on our approach. In addition to SNoW, we also show that standard learning algorithms, like naive Bayes, can also work within it. Experimental comparisons (Sec. 5) show that our approach outperforms several ILP-based systems tried on this task, sometimes significantly, while the time of training the system is much shorter. Finally, Sec. 7 concludes this paper and discusses some issues compared to other approaches.

2. Related Work

2.1 Propositionalization

Propositionalization is a transformation that maps a relational representation to features in propositional form (Kramer et al., 2001). These features capture the relational information and are usually stored as attributes in a vector, which forms an example provided to propositional learners.

Consider, for example, the classical ILP problem of identifying kinship relations (Hinton, 1986; Quinlan, 1990). Suppose `Tom` is the only child of `John` and `Mary`. Numerous relational features may be extracted as shown in Table 2.

ID	Predicate logic representation	Truth value
p1	$\textit{Father}(\texttt{John}, \texttt{Tom})$	1
p2	$\textit{Mother}(\texttt{Tom}, \texttt{Mary})$	0
p3	$\textit{Father}(\texttt{John}, \texttt{Tom}) \wedge \textit{Mother}(\texttt{Mary}, \texttt{Tom})$	1
p4	$(\exists x\ \textit{Father}(\texttt{John}, \texttt{x})) \vee \exists y\ \textit{Mother}(\texttt{y}, \texttt{Tom})$	1
...	...	...

Table 2: An example of propositionalization in the kinship domain

Although generating a full propositional representation from a relational learning setting is possible when the problem is in the simplest first-order form, where there is only one relation, the transformation process itself is exponential in the number of parameters of the original learning problem, such as the number of relations and the maximum number of rules in a hypothesis (De Raedt, 1998). Therefore, without restrictions, the process of propositionalization is intractable. Restrictions may be employed through the language bias in a logical representation, or through the parameters in a graphical representation of the relational data. Below we briefly describe several propositionalization approaches.

LINUS, DINUS and SINUS LINUS (Lavrac et al., 1991; Lavrac & Dzeroski, 1994) is the first published system that employs propositionalization. The language used by LINUS comprises constrained function-free non-recursive Horn clauses with negation. The clauses are constrained in the sense that all variables in the body also appear in the head. After propositionalization, each literal in the body is defined as a feature. For example, in the clause "`son(X,Y) :- male(X), parent(Y,X)`", `male(X)` and `parent(Y,X)` are treated as features.

DINUS (Lavrac & Dzeroski, 1994) is the successor of LINUS. The main difference is that it relaxes the language by allowing determinate non-constrained clauses. That is, if a literal in the clause has a variable that does not appear in preceding literals, this variable can only have one possible binding. An example of such a clause is "`grandfather(X,Y) :- father(Z,Y), father(X,Z)`"

The latest extension of the LINUS system, SINUS[3] (Krogel et al., 2003) takes flattened Prolog clauses (similar to those used in 1BC (Flach & Lachiche, 1999)) as its language. It generates

3. http://www.cs.bris.ac.uk/home/rawles/sinus/

features by examining literals in a clause from left to right. For each new literal, SINUS tries to apply various predicates given the current bindings of the variables. Users can restrict features by limiting the maximum number of literals, variables, etc.

WARMR WARMR (Dehaspe & Toivonen, 1999) is a system that detects Datalog queries that succeed frequently. The language used in Datalog is similar to Prolog, but without function symbols. A Datalog query is in the form "`?-` $A_1, \ldots, A_n$", which is a conjunction of logical atoms $A_1, \ldots, A_n$, and can be transformed to a relational feature. WARMR provides several options for restraining the features generated. The basic mechanism is called *mode constraints*, which can require the variables in an atom to be bound before the atom is called (i.e., input variables), or bound by the atom (i.e., output variables). Through type declarations, variable name sharing can be constrained. In addition, the choice of atoms picked as features can depend on whether other atoms appear or not.

RSD RSD is a system designed for relational subgroup discovery (Zelezny & Lavrac, 2006; Lavrac, Zelezny, & Flach, 2003; Krogel et al., 2003). Its input language is very similar to those used by the systems Aleph (Srinivasan & King, 1996) or Progol (Muggleton, 1995). Features generated are constrained by defining variable typing, moding, setting a *recall* parameter etc. For instance, a structural predicate declaration in the East-West trains domain (Michie, Muggleton, Page, & Srinivasan, 1994) can be defined as "`:-modeb(1,hasCar(+train, -car))`", where the recall number 1 indicates that a feature can address at most one car of a given train. The `+` and `-` signs assign input and output variables, respectively. RSD first generates an exhaustive set of features that satisfy the mode and setting declarations. A feature is a clause that cannot be decomposed into a conjunction of two features, and does not contain constants. Given this type of feature, users can further specify a type of variable that should be substituted with a constant by using the special property predicate `instantiate`. New features with constants will then be constructed. For example, a constant-free feature with the `instantiate` predicate "`f(A) :- hasCar(A,B), hasLoad(B,C), shape(C,D), instantiate(D)`" will examine the constants that variable `D` can be bound, and may generate features like "`f1(A) :- hasCar(A,B),hasLoad(B,C),shape(C,rectangle)`".

1BC 1BC (Flach & Lachiche, 1999) is a first-order Bayesian classifier that operates directly on the relational data. Unlike the aforementioned approaches, propositionalization is done implicitly and internally. 1BC uses an ISP (Individual, Structural predicate, and Property) representation in flattened Prolog. It provides a well-defined notation for *individuals* (e.g., a molecule in mutagenicity prediction, or a phrase in a sentence) and distinguishes two kinds of predicates: *properties* and *structural predicates*. Informally, a property is a predicate that describes the attributes of an individual, and a structural predicate is a binary predicate that denotes the relation between objects of two types. In addition, a language bias similar to the *mode constraints* in WARMR is also provided. Take the East-West trains problem as example: the individual is the train; structural predicates are `train2car` and `car2load`, which describe the car associated with the train, and the load of a car respectively; properties include predicates like `shape` (of a car), or `eastbound` (the direction of a train).

1BC is implemented in the context of `Tertius` (Flach & Lachiche, 2001), which is a first-order descriptive learner. `Tertius` searches for interesting conjunctions of literals as features. The search is restricted to a maximum number of literals and of variables, and limited by the language bias. Moreover, only *elementary* features are returned. A feature is elementary if it has only one property. This is important because combining both elementary and non-elementary features clearly violates the conditional independence assumption, which may degrade the performance of the naive Bayes learner.

Graphical representation Different from the above mentioned approaches, which represent relational data in logical languages, (Bournaud et al., 2003) employed propositionalization on a graphical representation of the relational data. This graphical representation is a subset of conceptual graphs (Sowa, 1984; Chein & Mugnier, 1992). In this formalism, a description of a relational data domain is a labeled directed graph with two types of vertices: *concept vertices* and *relation vertices*. The concept vertices are similar to the objects in a logical representation, and the relation vertices denote the relations between concepts. For example, a sub-graph $concept_s \rightarrow relation_r \rightarrow concept_t$ indicates that concepts s and t have the relation r.

The propositionalization process is restricted by parameterizing *abstract relations*. Given two concepts C_s and C_t in a graph, an abstract relation R_a is denoted by the path between C_s and C_t. The level of R_a is defined by the number of relation vertices the path has. Depending on the directions of edges in the path of an abstract relation, there are three types of canonical subgraphs: *sequence*, *star*, and *hole*. Every sub-graph of these three types along with different levels (number of edges) are then extracted as relational features. Note that this approach only allows binary relations, and the features generated are variable-free.

Similar to this approach, (Geibel & Wysotzki, 1996) propose a method that also generates features from a graphical representation. The relational features are derived from fixed-length paths in the neighborhood of a node in the graph. Objects represented by distant nodes are not of interest. This approach also has some relations to (Cumby & Roth, 2003a), which uses a *concept graph* representation, and discusses the equivalence to a logical (description logic based) representation.

Database oriented approaches A relational database is in fact a representation of relational data. It can be modeled as a graph, where tables are the nodes and foreign key relationships are the directed edges. Following the foreign key attribute, a dependent table points to the independent table that has the corresponding primary key. Propositionalization is usually employed by using aggregation operators, such as *average*, *minimum*, *maximum*, *count*, and *sum*. As a result, these features are often numerical instead of boolean. Examples of this type of approaches include RELAGGS (Krogel & Wrobel, 2001), and works in (Neville, Jensen, Friedland, & Hay, 2003; Perlich & Provost, 2003).

In addition to the language bias or parameters used in the above propositionalization approaches, feature selection or elimination is often applied as well. While most methods select features after they are generated (Kramer & Frank, 2000; Kramer & De Raedt, 2001), some methods do it during the constructing process (Alphonse & Rouveirol, 2000). For a more detailed survey and comparison on different propositionalization approaches, interested readers can refer to (Kramer et al., 2001) and (Krogel et al., 2003).

Our propositionalization framework is based on a graphical representation which is similar to (Bournaud et al., 2003), but the features generated are constrained by a combination of the language bias and the graphical structure. In particular, our framework differs from existing methods in two major respects. First, we define a language that allows users to parameterize preferences when constructing relational features. The language is designed with serious consideration of efficiency. Specifically, only local binding of quantifiers is allowed, in order to ensure rapid feature extraction. With the help of operators over the graphical structure, however, it is still possible to express complex features. These features are not limited to conjunctions but rather constitute a much richer, yet restricted set of FOL expressions. Second, the mapping from domains to graphs is very flexible, and allows multi-labeled nodes and edges. For instance, this framework has been applied to problems such as the kinship problem, problems in computational chemistry and other natural language problems, where domain elements are described as general graphs (Cumby & Roth, 2003a).

2.2 Information Extraction

SY: copied directly from Section 3 and need to be revised.

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 amount 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 & Lehnert, 1994), it has been argued quite convincingly (Califf & Mooney, 1999; Craven & Slattery, 2001) that relational methods are appropriate in learning how to directly extract the desired items from documents. Indeed, previous works (Califf & Mooney, 1999; Freitag, 2000; Goadrich, Oliphant, & Shavlik, 2006) have demonstrated the success of ILP methods in this domain.

Therefore, it is an interesting task to which our proposed relational learning framework can be applied.

We note that in addition to these systems, which view IE as a relational problem, several works have applied *word* based classifiers for the purpose of identifying phrases; most of these systems, though, still use fairly sophisticated relational features, without making this stage in the process explicit. Examples include (Freitag & McCallum, 2000), who apply hidden Markov models to predict each slot and (Chieu & Ng, 2002), who use Maximum Entropy classifier to predict whether a word is inside or outside a slot, and then combine these predictions to extract the target phrases.

3. Propositional Relational Representations

In this section we present a knowledge representation language that has two components: (1) a graphical representation of relational data, and (2) a subset of first order logic (FOL). A brief summary of how to apply this language for propositionalization on two relational learning tasks is also provided.

The language allows the encoding of first order representations and relational structures as propositions and thus supports the use of 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 & Roth, 2000), but technically is more related to the latter.

3.1 Graphical Representation of Relational Data

The relational data of interest constitutes the *domain* of our language. A domain consists of elements and relations among these elements. Predicates in the domain either describe the relation between an element and its attributes, or the relation between two elements.

Definition 3.1 (Domain) *A domain $\mathcal{D} = \langle \mathcal{V}, \mathcal{E} \rangle$ consists of a set of typed elements, $\mathcal{V}$, and a set of binary relations $\mathcal{E}$ between elements in $\mathcal{V}$. An element is associated with some attributes. When two elements have different sets of attributes, we say that the two elements belong to different types.*

Elements in a domain $\mathcal{D}$ represent objects in the world. For example, in NLP applications SY: please check these might be words, phrases, sentences or documents. Different from the traditional definition of "domain", which consists of only elements, we specifically denote the binary relations $\mathcal{E}$ in the graphical representation of the data. As we will see, this design helps to evaluate the truth value of a predicate efficiently. Predicates used for $\mathcal{D}$ can be categorized into two types. One describes properties of these elements – the spelling of a word, the part-of-speech tag of a word, a phrase type, etc. The other describes relations between (sets of) objects. For example, word w_1 is *before* w_2, w_3 is the *first* word of phrase ph_1, etc. As usual, the "world" in which we interpret the aforementioned construct could be a single sentence, a document, etc.

Definition 3.2 *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.*

Each instance can be mapped to a directed graph. In particular, each node represents an element in the domain, and each link (directed edge) denotes the relation that holds between the two connected elements. This relational data representation is close to the conceptual graph formalism (Sowa, 1984; Chein & Mugnier, 1992; Bournaud et al., 2003). Compared to (Bournaud et al., 2003), the relations between elements are represented as links in the *instance*, while Bournaud et al. define *relation vertices* to store the relation information. In contrast to the ISP representation (Flach & Lachiche, 1999), the *individual* is the same as the *instance* here. *Properties* are predicates that extract attributes of an *element*. *Structural predicates* are similar to the relations represented as links, although the elements may belong to the same type in our case.

We provide the following two examples to illustrate this graphical representation. One is the classical kinship example (Quinlan, 1990), and the other is a natural representation of text fragments for many NLP problems.

Example 3.1 *The instance of a family domain in Fig. 1 shows the kinship of several people, which are represented by the nodes. In addition, each node is associated with a set of attributes, like* name, gender*, and* age*. Each link in the graph shows the relation between two people. For example, n_4 is the* father *of n_3, who is also a* son *of n_2. In this domain, all objects belong to the same type.*

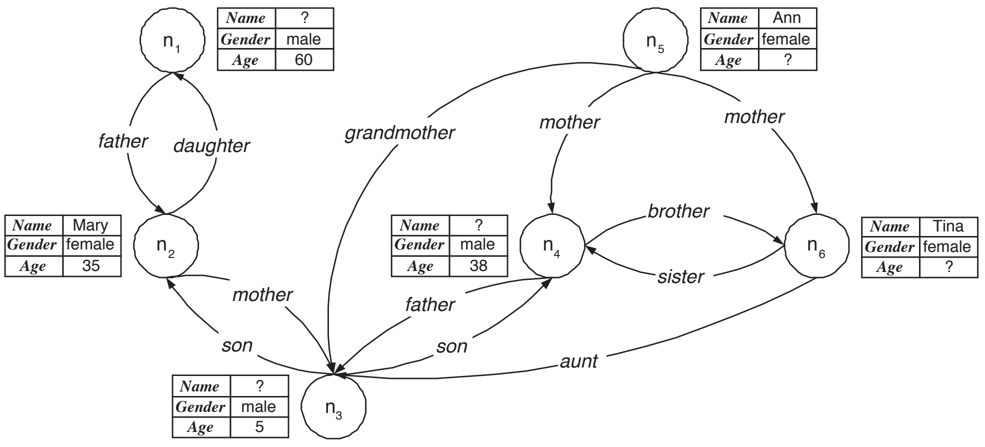

Figure 1: An instance of the kinship domain: each node (element) represents a person in this family. The associated attributes are *Name*, *Gender*, and *Age*. Some attribute values of an element may be unknown. The relation of two persons is denoted by the link connecting them.

Example 3.2 *Fig. 2 shows a representation of the text fragment "TIME : 3 : 30 pm – 6" as an instance in the text domain. In this domain, there are two types of objects: e.g.,* $w_1, w_2, \cdots, w_8$ *are words, and* ph_1 *is a phrase. The attributes of each word include its* spelling *and* part-of-speech*, while the attributes of a phrase are its* length *and* label*. The positional relations* before *and* after *are used to connect two consecutive words. Relations* first *and* last *indicate the first word and last word of a phrase respectively. Note that there can be different ways to represent text. For example, when the parsing information is available, the instance that describes a sentence may be a tree.*

3.2 Relational Language

We define the relational language $\mathcal{R}$ as 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 those that can be evaluated very efficiently on given instances (Def. 3.2). This is done by (1) defining primitive formulae with a limited scope of quantifiers (Def. 3.3), and (2) defining general formulae inductively, in terms of primitive formulae, in a re-

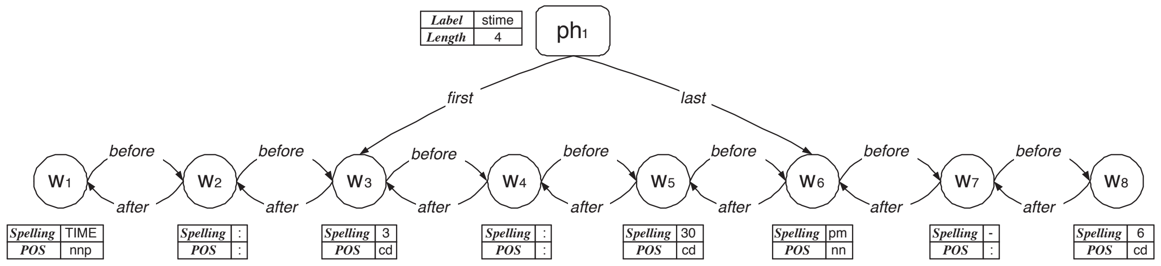

Figure 2: An instance in the text domain: there are two types of elements – *words* and *phrases*. Words $w_1, \cdots, w_8$ are connected by their positional relations, *before* and *after*. Each word has attributes *spelling* and *part-of-speech*. Phrase ph_1 has links pointing to its first and last word, and is associated with its label and length.

stricted way that depends on the relational structures in the domain (Sec. 3.3). The emphasis is on locality with respect to the relational structures that are represented as graphs.

We omit many of the standard FOL definitions and concentrate on the unique characteristics of $\mathcal{R}$ (see, e.g., (Lloyd, 1987) for details). The vocabulary in $\mathcal{R}$ consists of constants, variables, predicate symbols, quantifiers, and connectives. Constants and predicate symbols vary for different domains. In particular, for each constant in $\mathcal{R}$, there is an assignment of an element in $\mathcal{V}$. For each k-ary predicate in $\mathcal{R}$, there is an assignment of a mapping from $\mathcal{V}^k$ to $\{0,1\}$ ($\{$true, false$\}$). We present the main constructs of the language by first defining primitive formulae.

Definition 3.3 *A primitive formula is defined inductively:*

1. *A term is either a variable or a constant.*
2. *Let p be a k-ary predicate with terms $t_1, \cdots, t_k$. Then $p(t_1, \cdots, t_k)$ is an atomic formula.*
3. *Let F be an atomic formula, z a free variable in F. 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 $(\neg F)$, $(F \wedge G)$, $(F \vee G)$.*

SY: check it

For simplicity, an atomic formula is also called an atom. There are several differences between the primitive formula here and the definition of formula in FOL. First, as stated earlier, our relational language is function-free; therefore, a term can only be either a variable or a constant (rule 1). Second, quantified atomic formulae are still atomic formulae (rule 3). This design guarantees that we can evaluate quantification fairly efficiently. Notice that for primitive formulae in $\mathcal{R}$, the *scope* of a quantifier is always the unique predicate that occurs within 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 the same as usual.

SY: check the new example

Example 3.3 $p(x, y)$, $(\forall z(p(z, a)))$, $(\exists z(\forall x(p(x, z))))$ *are all atomic formulae.* $((\exists z(\forall x(p(x, z))))\wedge (\neg(\forall z(p(z, a)))))$ *is a primitive formula, while* $(\exists z((\forall x(p(x, z))) \vee (\exists y(q(y, z)))))$ *is not.*

Definition 3.4 (Formula) *Let* $f : \{0, 1\}^n \rightarrow \{0, 1\}$ *be a Boolean function of* n *variables, and let* $F_1, F_2, \ldots F_n$ *be primitive formulae in* $\mathcal{R}$*. A* clause *is a formula of the form* $f(F_1, F_2, \ldots, F_n)$*.*

This can be used, in particular, to define disjunctive, conjunctive and implication clauses.

SY: emphasize that "formula" is really a feature function

Notice the term formula in our language is somewhat different from the definition in FOL. The role of formula is in fact a binary feature function as usually referred in the machine learning community. Given an instance x, a formula F in $\mathcal{R}$ outputs a unique truth value, *the value of* F *on* x. It is defined inductively using the truth values of the predicates in F and the semantics of the connectives. Notice that if F has the form $\exists d_1, \ldots d_k p(d_1, \ldots d_k)$, for some k-ary predicate p, then its truth value is *true* if and only if there exists $d_1, \ldots d_k \in x$, $p(d_1, \ldots d_k)$ has truth value *true*. Similarly, if F has the form $\forall d_1, \ldots d_k p(d_1, \ldots d_k)$, for some k-ary predicate p, then its truth value is *true* if and only if for all $d_1, \ldots d_k \in x$ such that $p(d_1, \ldots d_k)$ 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. 3.3) maintains these 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.*

3.3 Relation Generation Functions

So far we have defined only primitive formulae and minimized the interaction between variables. In order to generate more expressive formulae that respect the graphical structure in a given instance, we define *relation generation functions* in this section, which allows the formulae generated to be

SY: check this

evaluated efficiently. The role of relation generation functions can be thought of as the feature *type*,

new paragraph

which generates various active features (i.e., formulaes that have the true value) given an instance.

SY: I try to explain what we mean by “active”

Definition 3.5 *A formula in $\mathcal{R}$ is used as a feature function that maps an instance x to its truth value in x. The feature is* active *if it has truth value* true *in x. In other words, x is a model for the formula. We denote by X the set of all instances, the* instance space. *A formula $F \in \mathcal{R}$ is thus a* relational feature *over X, $F : X \rightarrow \{0,1\}$.*

Example 3.4 *Let instance x be the text fragment illustrated in Ex. 3.2. Some active formulae in x are* word*(TIME),* word*(pm), and* number*(30). On the contrary, infinitely more formulae, such as* word*(am), are not active.*

Given an instance, we would like to know what formulae (relational features) are *active* in it. In addition, this should be done without the need to write down explicitly all possible formulae in the domain. This is important over infinite domains or in problem domains such as NLP, where inactive formulae vastly outnumber active formulae. Therefore, only active formulae 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 3.6 *Let $\mathcal{X}$ be an enumerable collection of formulae 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 χ 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 "types" of formulae, or to parameterize over a large space of formulae. Only when an instance x is present, a concrete formula (or a collection of formulae) is generated. An RGF can be thought of as having its own range $\mathcal{X}$ of relational features.

Example 3.5 *It is impossible to list all formulae that use the* number *predicate in advance. However, an RGF can specify formulae of this kind. Given the instance "*`TIME : 3 :  30 pm`*", only the active relations of this kind:* number(3) *and* number(30) *and, potentially,* $\exists x$ number(x) *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 defines 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. The use of sensors 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 3.7 *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 3.6 *Following are some sensors that are commonly used in NLP.*

- *The* word *sensor over word elements, which outputs the active relations* word(TIME), word(:), word(3), word(30), *and* word(pm) *from "TIME : 3 : 30 pm".*

- *The* vowel *sensor over word elements, which outputs* vowel(word) *or* $\exists x$ vowel(x) *when the word it operates on begins with a vowel.*

- *The* length *sensor over phrase elements, which outputs the active relation* length(4) *from "3 : 30 pm".*

- *The* is-a *sensor, which outputs the semantic class of a word.*

- *The* tag *sensor, which outputs the part-of-speech tag of a word*

The word *and* length *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.*

One of the key cited advantages of ILP methods is the ability to incorporate background knowledge. In our framework, 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.

Several mechanisms can be used in the relational calculus to restrain the scope of RGFs' operations. We define here the *focus* mechanism, which specifies a subset of elements in an instance on which an RGF can be applied.

Definition 3.8 *Let E be a set of elements in a given instance x. An RGF r is* focused *on E if it generates only formulae in its range that are active in x due to elements in E. The focused RGF is denoted by $r[E]$.*

SY: check this supplemental explanation

The focus mechanism can be treated as a sub-domain of interpretation. In practice, it is quite only that we want to restrict the formulae being evaluated on only a subgraph of the data, and therefore generate features only from this subset of the data. There are several ways to define a focus set. It can be specified explicitly or described indirectly by using the structure information (i.e., the links) in the instance. For example, when the problem is to predict the label of each element in a text fragment (e.g., part-of-speech tagging), focus may be defined relative to the target element.

When the goal is to predict some property of the whole instance (e.g., distinguish the mutagenicity of a compound), the focus can simply be the whole instance.

The relational calculus allows one to inductively generate new RGFs by applying connectives and quantifiers over existing RGFs. Using the standard connectives one can define RGFs that output formulae of the type defined in Def 3.3 (see (Cumby & Roth, 2000) for details).

We now augment the relational calculus by adding structural operations, which exploit the structural (relational) properties of a domain as expressed by the links. RGFs defined by these structural operations can generate more general formulae that have more interactions between variables but still allow for efficient evaluation and subsumption, due to the graph structure.

We define two structural collocation operators that make use of the chain structure in a graph as follows.

Definition 3.9 (collocation) *Let $s_1, s_2, \ldots s_k$ be RGFs for $\mathcal{R}$, g a chain-structured subgraph in a given domain $\mathcal{D} = (\mathcal{V}, \mathcal{E})$. $colloc_g(s_1, s_2, \ldots s_k)$ is a restricted conjunctive operator that is evaluated on a chain of length k in g. Specifically, let $v_1, v_2, \ldots v_k \in \mathcal{V}$ be 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 here the RGF s_j is focused to $\{v_j\}$, and*

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. This is needed since each RGF in the conjunction may produce more than one formula.*

The labels of links can be chosen to be part of the generated features if the user thinks the information could facilitate learning.

Example 3.7 *When applied with respect to the graph g which represents the linear structure of a sentence, $colloc_g$ generates formulae that corresponds to n-grams. E.g., given the fragment "Dr John Smith", RGF* colloc(word, word) *extracts the bigrams* `word(Dr)-word(John)` *and* `word(John)-word(Smith)`. *When the labels on the links are shown, the features become* `word(Dr)-before-word(John)` *and* `word(John)-before-word(Smith)`. *If the linguistic structure is given instead, features like* `word(John)-SubjectOf-word(builds)` *may be generated. See (Even-Zohar & Roth, 2000) for more examples.*

Similarly to $colloc_g$, one can define a *sparse* collocation as follows:

Definition 3.10 (sparse collocation) *Let $s_1, s_2, \ldots s_k$ be RGFs for $\mathcal{R}$, g a chain structured subgraph in a given domain $\mathcal{D} = (\mathcal{V}, \mathcal{E})$. $scolloc_g(s_1, s_2, \ldots s_k)$ is a restricted conjunctive operator that is evaluated on a chain of length n in g. Specifically, let $v_1, v_2, \ldots v_n \in \mathcal{V}$ be a chain in g. For each subset $v_{i_1}, v_{i_2}, \ldots v_{i_k}$, where $i_j < i_l$ when $j < l$, all the formulae: $s_1[v_{i_1}]\&s_2[v_{i_2}]\& \ldots \&s_k[v_{i_k}]$, are generated.*

Example 3.8 *Given the fragment "Dr John Smith", the features generated by RGF scolloc(word,word) are* `word(Dr)-word(John)`, `word(Dr)-word(Smith)`, *and* `word(John)-word(Smith)`.

Notice that while primitive formulae in $\mathcal{R}$ have a single predicate in the scope of each quantifier, 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 instance. It also 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.

3.4 Application Examples

Our propositionalization method has been applied to several classical ILP tasks (Cumby & Roth, 2003a), where SNoW is used as the propositional learner. In order to provide a more complete exposition of the framework, we summarize the experiments of *learning kinship relations* and *predicting mutagenicity* here.

3.4.1 Learning Kinship Relations

The problem of learning the definitions of kinship relations has been a typical task for ILP approaches since late 80's (Hinton, 1986; Quinlan, 1990). The benchmark consists of 112 positive relations over two separate families, and the goal is to learn the relation of any two people.

The data set is first mapped to the graphical representation presented in Sec. 3.1, where a person is represented by a node, and the relation between two persons is depicted by a link. Given two nodes S and T, the target instance is the sub-graph of these two nodes and all the links and nodes on the paths from S to T. Relational features are generated along the paths between them. In particular, `colloc` of size 2 and 3 along each path are used. Since the only attribute, *name*, of each element is not important to learning the definition of a relation, only the labels of the links are kept as features. For example, a relational feature might be –`brother`–`mother`–, or –`husband`–`sister`–`father`–. Note that if important attributes of nodes (e.g., *gender*) are present, they can also be extracted by sensors as features.

The experimental result shows the feasibility and efficiency of our framework. After two cycles of training on 101 examples, the accuracy converges to 99.36%. As a comparison, FOIL (Quinlan, 1990) takes 500 sweeps and FORTE (Richards & Mooney, 1992) trains on over 300 examples to achieve the same level of accuracy.

3.4.2 Mutagenesis

Mutagenicity prediction has become a standard benchmark problem for propositionalization methods in ILP (Srinivasan, Muggleton, King, & Sternberg, 1996). In this task, the goal is to distinguish compounds with positive log mutagenicity (labeled as "active") from those having zero or negative log mutagenicity (labeled as "inactive"). For each compound, the internal structure is represented as a set of Prolog facts in tuples. Specifically, `bond(compound,atom1,atom2,bond-type)` describes the bonds connecting atoms, and `atm(compound,atom,element,atom-type, charge)` denotes the elements of an atom. Global information about a compound, such as its mutagenicity, logP, and lumo values, is also provided.

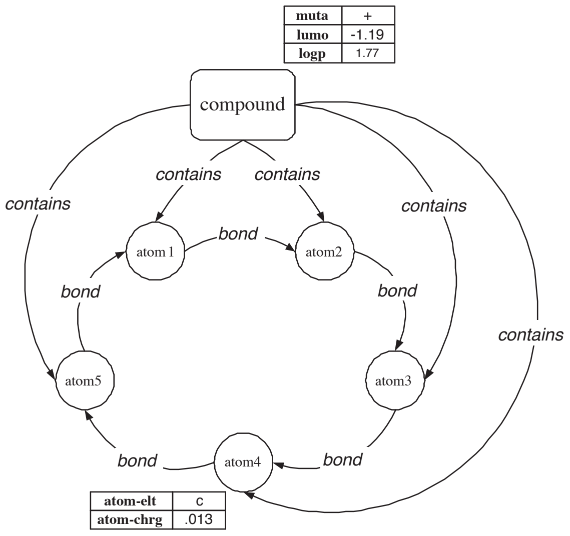

Figure 3: The graphical representation of a mutagenesis domain (taken and revised from (Cumby & Roth, 2003b)): each domain has one *compound* element that has relations *contains* pointing to each *atom*; *atoms* have *bond* relations among them; each element has its own attributes.

To apply our framework, each compound is mapped to a graph, where each node represents an atom. The attributes of an atom include the atom types, element types, and partial charges. Edges between atoms corresponds to the bonds, labeled with bond-types. In addition, there is one special node that represents the whole compound with the global information stored as attributes. Fig. 3 shows an example of this graphical representation.

The features generated for each example include the global information of the compound, and `colloc` of size 9 on paths between two atoms. The relational features consist of the attributes of the nodes extracted by predefined sensors and the bond-types labeled on the edges as well. Notice that the `colloc` operator used to match the paths are somewhat specialized for matching the chain subgraphs. However, one can, in principle, invent new operators that produce features that match any subgraph; the key issue here is computational since determining whether a feature is active in a given domain instance will become a sub-graph isomorphism problem. Interested readers can refer to (Cumby & Roth, 2003b) for more discussion.

SY: To address comment by Reviewer #2

The system based on our framework achieves an accuracy of 88.6%, which is comparable to the results achieved in (Srinivasan et al., 1996) using the Progol (Muggleton, 1992) system, but somewhat below the best results on this data set (Sebag & Rouveirol, 1997; Zelezny & Lavrac, 2006).

4. 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 amount 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 & Lehnert, 1994), it has been argued quite convincingly (Califf & Mooney, 1999; Craven & Slattery, 2001) that relational methods are appropriate in learning how to directly extract the desired items from documents. Indeed, previous works (Califf & Mooney, 1999; Freitag, 2000; Goadrich et al., 2006) have demonstrated the success of ILP methods in this domain. Therefore, it is an interesting task to which our proposed relational learning framework can be applied.

We note that in addition to these systems, which view IE as a relational problem, several works have applied *word* based classifiers for the purpose of identifying phrases; most of these systems, though, still use fairly sophisticated relational features, without making this stage in the process explicit. Examples include (Freitag & McCallum, 2000), who apply hidden Markov models to predict each slot and (Chieu & Ng, 2002), who use Maximum Entropy classifier to predict whether a word is inside or outside a slot, and then combine these predictions to extract the target phrases.

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

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

4.1.1 Seminar Announcements

The first data set consists of 485 CMU seminar announcements[4]. The goal here is to extract four types of fragments from each announcement – 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., `speaker`, more than once.

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

4.1.2 Computer-related Job Postings

The second data set is a set of 300 computer-related job postings from UT Austin[5]. In this case, 17 types of fragments that describe the job position are 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.

4. The data set was originally collected from newsgroups and annotated by Dayne Freitag; it is available at (RISE, 1998).
5. The data set was originally collected from newsgroups and annotated by Mary Califf; it is available at (RISE, 1998).

$\cdots$ the $\boxed{\text{talk given by:}}$ $\boxed{\text{Dr FName LName}}$ $\boxed{\textit{line-feed}\text{ Topic : A}}$ novel $\cdots$

Figure 4: Three regions for feature extraction

4.2 Extracting Relational Features

The basic strategy of our IE solution is to learn a classifier that labels text fragments. 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, $t_2, \cdots, t_n$, indexed by their positions in the document. Any string of consecutive tokens t_i, $t_{i+1}, \cdots, t_j$, 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 predefined RGFs.

Let $f = (t_i, t_{i+1}, \cdots, t_j)$ be a fragment. 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. Figure 4 demonstrates one example, in which the three framed boxes represent left window, target region, and right window ($w = 4$) respectively.

For each fragment, an instance is formed by these three regions. There are two types of elements – word elements (e.g., $t_{i-w}, \cdots, t_{j+w}$) and one phrase element (the target region) inside the instance. The graphical representation is the same as shown in Fig. 2. The RGFs are focused either on a specific word element or the phrase element and define relational features relative to these. Examples of RGFs used in the experiments are shown in Section 4.4.1 and Figure 6.

4.3 Applying Propositional Learning Algorithms

Given RGFs and an instance as described above, an example is a list of the formulae that are active in this instance. Once examples have been generated, we can apply any propositional learning algorithm to learn on them. In addition to the Winnow variation within the SNoW learning architecture, we also tested the naive Bayes algorithm (also implemented within the SNoW architecture).

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 may somewhat sacrifice the performance (Golding & Roth, 1999; Carlson et al., 2001). To test its effect, we conducted a series of experiments that eliminate different portions of features according to the occurrence frequency.

Another potential computational difficulty comes from the essence of IE tasks. Since any document fragment may be of potential interest as the field for some slot, a large number of examples needs 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 examples[6]. This reduces the learning time, but may have an adverse impact on performance. To test this we compare several methods, including randomly selecting a subset of the negative examples and using only positive examples. Our main approach to address this issue is described next.

4.4 Two-stage Architecture

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

SNoW is typically used as a *predictor*, when it uses a winner-takes-all mechanism over the activation values of the target classes. Here we suggest to rely directly on the activation value it

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

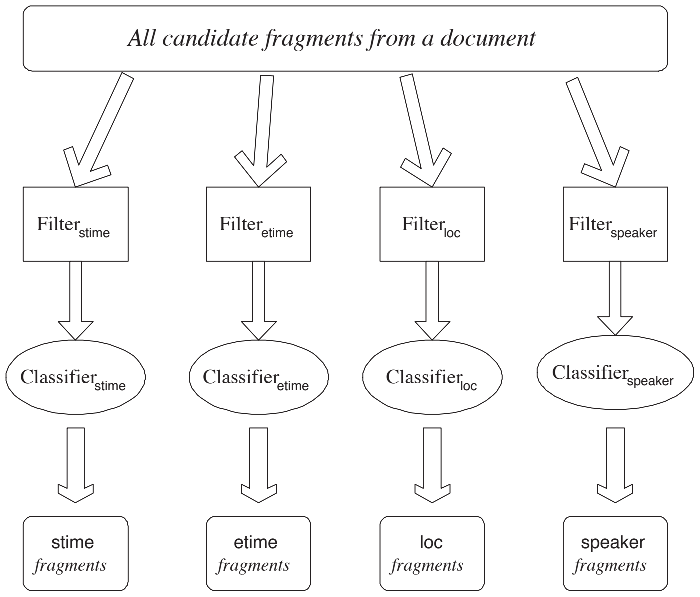

Figure 5: Two-stage architecture for seminar announcements data

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 number of candidate fragments to a small number by removing fragments that are very likely to be irrelevant, and
2. Classification: identifying the correct fragment from the remaining fragments and classifying it into one of the target classes.

Figure 5 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 see 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 the 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.

4.4.1 Filtering

This stage attempts to filter out most of the negative examples while eliminating as few positive examples as possible. 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. This information can be derived from simple statistics from the training data or specified by human experts.

2. General Classifier: the fragment's confidence value is below the threshold.

For criterion 1, it turns out that there exist some features that are (almost) always active on positive examples. For example, in our experiments, the *length of fragments* satisfies this criterion. $len(fragment) < 7$ always holds in `stime` fragments in seminar announcements. Similarly, 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 value of SNoW, converted via a sigmoid function, is used to measure how likely a fragment is to represent a specific slot. Thresholds for the different types of slots are set to be slightly higher than the minimum activation values of the positive examples in the training data. Examples with low activation values are filtered out.

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

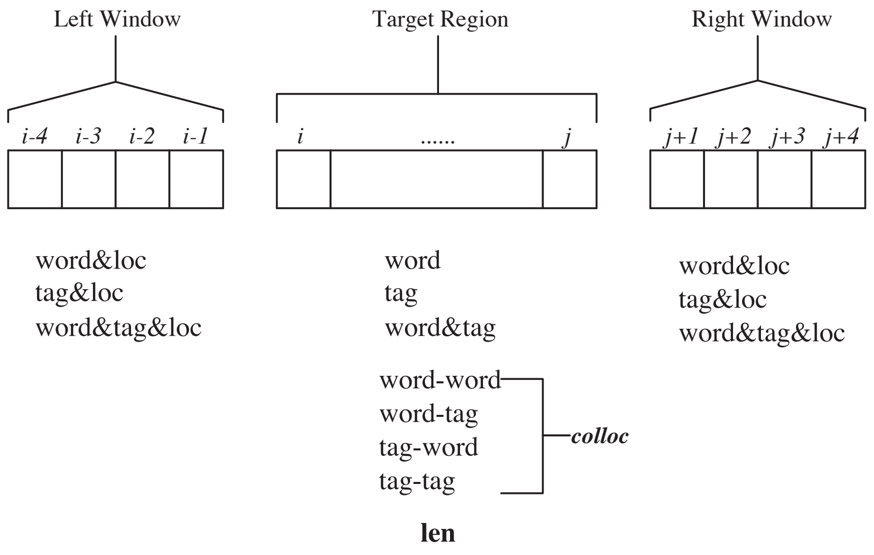

Figure 6: Graphical Illustration of a set of RGFs

The use of these RGFs to extract features that represent a fragment is demonstrated in Figure 6.

4.4.2 Classification

Fragments that pass the filtering stage are then classified using a second SNoW classifier for each slot. First, an additional collection of RGFs is applied to enhance the representation of the candidate fragments, and thus allow for more accurate classification. In training, the remaining fragments (which are annotated) are used as positive or negative examples to train the classifiers as in the previous stage. 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.

A good way to view our two-stage architecture is as a multi-class classifier that runs the sequential model (Even-Zohar & Roth, 2001). The first stage (i.e., filtering) outputs a smaller set of classes that need to be considered. Given a candidate phrase, the second stage (i.e., classification) just predicts it as one of the classes in the subset.

5. Experimental Results

Our framework 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, 1998; Califf & Mooney, 1999, 2003), SRV (Freitag, 2000), and WHISK (Soderland, 1999)). We compare our overall results to theirs in sections 5.1. We also report the results of experiments on the effects of pruning features and examples in section 5.2. Interested readers can find the description of the feature extraction tool we developed as well as the detailed settings of our experiments on the Web [7].

5.1 Comparison to Other Systems

Our experiments use the same data, test methodology and evaluation metrics used by several ILP-based IE systems in previous works. As usual, the performance is quantified in terms of *precision* (the percentage of correct predictions), *recall* (the percentage of *slots* that are identified), and $F_{\beta=1}$ (the harmonic mean of precision and recall). We estimate the 95% confidence intervals of the $F_{\beta=1}$ values using bootstrap re-sampling (Noreen, 1989). To do this, we repeatedly sample the documents for each test set, with replacement, to generate new test data. The distribution of $F_{\beta=1}$ in this data is then used as the distribution of the performance of each learned system.

In addition to the results of our main system, 2-SNoW-IE (2-stage architecture SNoW Learning), we also present the results of our other two systems, 1-SNoW-IE and 1-NB-IE, as a comparison. 1-SNoW-IE learns the classifier based only on examples generated for the first stage. This provides some idea on 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 exam-

7. http://l2r.cs.uiuc.edu/∼cogcomp/ie-fex.html

System	2-SNoW-IE			1-SNoW-IE			1-NB-IE		
	Prec	Rec	$F_{\beta=1}$	Prec	Rec	$F_{\beta=1}$	Prec	Rec	$F_{\beta=1}$
stime	99.8	99.8	99.8 ± 0.2	98.2	97.4	97.8 ± 0.8	97.9	97.9	97.9 ± 0.8
etime	98.4	95.4	96.9 ± 1.2	92.3	91.2	91.7 ± 2.1	81.1	91.8	86.1 ± 2.4
location	89.9	63.9	74.7 ± 2.3	83.7	58.9	62.9 ± 2.6	40.3	34.7	37.3 ± 2.8
speaker	82.4	63.0	71.4 ± 2.6	72.6	61.1	66.4 ± 2.9	17.4	14.4	15.7 ± 2.4

System	RAPIER-WT			RAPIER			SRV			WHISK		
	Prec	Rec	$F_{\beta=1}$	Prec	Rec	$F_{\beta=1}$	Prec	Rec	$F_{\beta=1}$	Prec	Rec	$F_{\beta=1}$
stime	96.5	95.3	95.9	93.9	92.9	93.4	98.6	98.4	98.5	86.2	100.0	92.6
etime	94.9	94.4	94.6	95.8	94.6	95.2	67.3	92.6	77.9	85.0	87.2	86.1
location	91.0	61.5	73.4	91.0	60.5	72.7	74.5	70.1	72.2	83.6	55.4	66.6
speaker	79.0	40.0	53.1	80.9	39.4	53.0	54.4	58.4	56.3	52.6	11.1	18.3

Table 3: Results for seminar announcements task

ples) that were generated using our propositionalization framework for 1-SNoW-IE, and in exactly the same way. We discuss the experiments in more details in the following subsections.

5.1.1 Seminar Announcements

In the experiment of *seminar announcements*, the data (485 documents) is randomly split into two sets of equal size, one for training and the other for testing. The reported results (Table 3) are an average of five runs. 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 cross 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 (i.e., 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. The full 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 performs very competitively and its F-measure scores of all four slots are higher than those of the existing rule-based IE systems [8]. To clarify, we note that the output representation of our system makes use of similar types of relational features as do the ILP-based

SY: the new version. The old version is "Overall, 2-SNoW-IE outperforms the existing rule-based IE systems on all the four slots."

8. Whether the differences are statistically significant is unclear. Since the data splits of other systems were not published, it is not possible to run a statistical significance test such as the paired-t test.

	% examples that are retained after filtering				% positive examples lost	
slot	training		testing		training	testing
stime	6.47%	(45664/706270)	6.33%	(43593/688535)	0.41%	1.27%
etime	2.04%	(14414/706270)	2.06%	(14194/688535)	0.00%	1.57%
location	9.28%	(65555/706270)	9.17%	(63117/688535)	0.87%	6.31%
speaker	12.68%	(89550/706270)	12.18%	(83860/688535)	0.68%	11.78%

Table 4: Filtering efficiency on seminar announcement data

systems, 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 are in the appendix.

The above results also exhibit the enhancement in performance due to the two-stage architecture of the SNoW-IE system, compared to the one-stage architecture using either SNoW or naive Bayes. As a side note, the ability to use this architecture provides another indication of the flexibility of the approach and the advantage of learning with propositional means. Table 4 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 are retained after filtering in training and testing respectively. Generally, the number of examples is reduced by 87.32% to 97.96%. 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 due to the quality of the classifier, the experiments with a second propositional algorithm exhibit the fact that our relational learning framework is a general one. As indicated in (Craven & Slattery, 2001; Freitag, 2000) a simple-minded use of this algorithm is not competitive for this task. However, when used on top of a framework that is able to exploit the relational nature of the data, it compares favorably with ILP methods in some cases.

	2-SNoW-IE			1-SNoW-IE			1-NB-IE			Rapier		
	Prec	Rec	$F_{\beta=1}$	Prec	Rec	$F_{\beta=1}$	Prec	Rec	$F_{\beta=1}$	Prec	Rec	$F_{\beta=1}$
id	99.0	98.3	98.7 ± 1.3	98.3	96.3	97.3 ± 1.7	93.6	92.3	93.0 ± 3.0	98.0	97.0	97.5
title	65.3	40.0	49.2 ± 6.3	65.3	33.2	44.0 ± 6.8	36.8	35.3	36.1 ± 5.9	67.0	29.0	40.5
salary	84.1	54.2	65.9 ± 9.9	83.3	32.7	47.0 ± 9.9	23.4	70.1	35.1 ± 4.1	89.2	54.2	67.4
company	86.4	64.8	74.0 ± 8.6	73.4	65.9	69.5 ± 8.9	56.4	60.2	58.2 ± 9.7	76.0	64.8	70.0
recruiter	83.9	78.3	81.0 ± 5.1	71.8	73.5	72.6 ± 6.2	50.6	52.4	51.5 ± 7.4	87.7	56.0	68.4
state	93.7	95.0	94.3 ± 2.8	93.7	95.0	94.3 ± 2.8	92.6	91.3	91.9 ± 3.1	93.5	87.1	90.2
city	96.1	92.3	94.6 ± 2.1	94.2	93.2	93.7 ± 2.3	93.7	89.7	91.7 ± 3.0	97.4	84.3	90.4
country	97.0	94.0	95.5 ± 2.8	97.0	93.4	95.1 ± 3.1	93.9	91.2	92.5 ± 3.6	92.2	94.2	93.2
language	88.2	76.3	81.8 ± 3.0	83.2	61.0	70.4 ± 3.9	42.0	63.1	50.4 ± 3.1	95.3	71.6	81.8
platform	77.6	68.1	72.5 ± 4.2	65.8	50.9	57.4 ± 4.8	46.7	62.6	53.5 ± 4.0	92.2	59.7	72.5
application	78.4	54.6	64.4 ± 5.3	50.9	46.5	48.6 ± 5.3	54.4	61.2	57.6 ± 5.0	87.5	57.4	69.3
area	56.9	34.6	43.0 ± 4.5	46.1	32.5	38.1 ± 4.3	24.7	33.5	28.4 ± 3.1	66.6	31.1	42.4
req_yrs_exp	87.3	81.6	84.4 ± 5.1	84.0	79.6	81.8 ± 5.8	43.4	54.0	48.1 ± 7.2	80.7	57.5	67.2
des_yrs_exp	81.8	83.7	82.8 ± 9.0	89.3	58.1	70.4 ± 12.6	33.8	60.5	43.3 ± 11.4	94.6	81.4	87.5
req_degree	98.3	70.7	82.3 ± 7.7	83.8	81.7	82.7 ± 6.3	71.9	78.1	74.9 ± 7.2	88.0	75.9	81.5
des_degree	100.0	38.1	55.2 ± 23.5	42.9	28.6	34.3 ± 21.5	31.7	61.9	41.9 ± 13.8	86.7	61.9	72.2
post_date	99.0	99.3	99.2 ± 0.9	99.0	99.3	99.2 ± 0.9	99.0	99.3	99.2 ± 0.9	99.3	99.7	99.5
total	86.6	72.0	77.6 ± 2.2	77.8	66.0	70.4 ± 2.3	58.1	68.0	61.6 ± 2.0	89.4	64.8	75.1

Table 5: Results for computer-related job postings task

5.1.2 Computer-related Job Postings

In the experiment of *computer-related job postings*, we used the same data and methodology as with Rapier. In particular, we report the results using 10-fold cross validation on 300 newsgroup documents. Table 5 shows the results of our three systems, and the Rapier (Califf, 1998) system that uses words, part-of-speech tags, and semantic classes. Unlike the experiments on *seminar announcements*, semantic classes do help Rapier to achieve better performance in this case. Nevertheless, this information is not used in any of our information extraction systems.

SY: A revised paragraph. The old version can be found in the latex file, exp.tex.

As shown in Table 5, 2-SNoW-IE performs comparably to Rapier and has higher F-measure scores on slots like `title`, `recruiter`, `state`, `city`, and `req_yrs_exp`, and has similar scores on others. As a result, the average $F_{\beta=1}$ is slightly higher despite not using the semantic class sensor, although it is unclear whether the differences are statistically significant due to different data splits. Similar to the trend we observe in the *seminar announcements* experiment, 1-SNoW-IE and 1-NB-IE are not as good as 2-SNoW-IE experiment. While they still achieve an equal or higher level of performance on some slots, they perform worse on others, and therefore the overall $F_{\beta=1}$ are somewhat lower.

Since Rapier is biased to generate more reliable rules, we notice that the precisions of these slots tend to outnumber those in our systems. However, one advantage of using propositional algorithms like Winnow-based SNoW or naive Bayes is that the trade-off of recall and precision can be easily tuned by changing the value of a threshold parameter. In applications where precision is important, our system can be required to only output slots with high activation values, which indicate the confidence of the predictions.

5.1.3 Training Time

One key advantage of our framework is that the training time needed to construct a system is much shorter than ILP-based systems. As reported in (Califf, 1998), running on an SGI-Origin-200, Rapier takes 8 hours to train on 240 seminar announcements when only words are used. Adding part-of-speech tags and semantic classes increases the training time to 15 hours in average. Similarly, to train on 270 job postings, the words only version and the full version of Rapier spend 26 and 42 hours respectively.

On the contrary, 1-SNoW-IE takes around 50 minutes both to extract features and train on 240 seminar announcements, and 110 minutes on 270 job postings, running on a Pentium-III 866MHz machine. Depending on the filtering efficiency and the number of slots in the template, 2-SNoW-IE takes a bit longer, but still finishes the process within an average of 2 hours for *seminar announcements* and 4 hours for *job postings* experiments.

As the training time of a system is seldom reported in the literature, it is fairly hard to conduct an extensive study of this issue. Even if the information is available, the use of different hardware still do not allow exact comparisons. However, we believe the numbers reported here at least provide some feeling for the significant computational difference.

5.2 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.

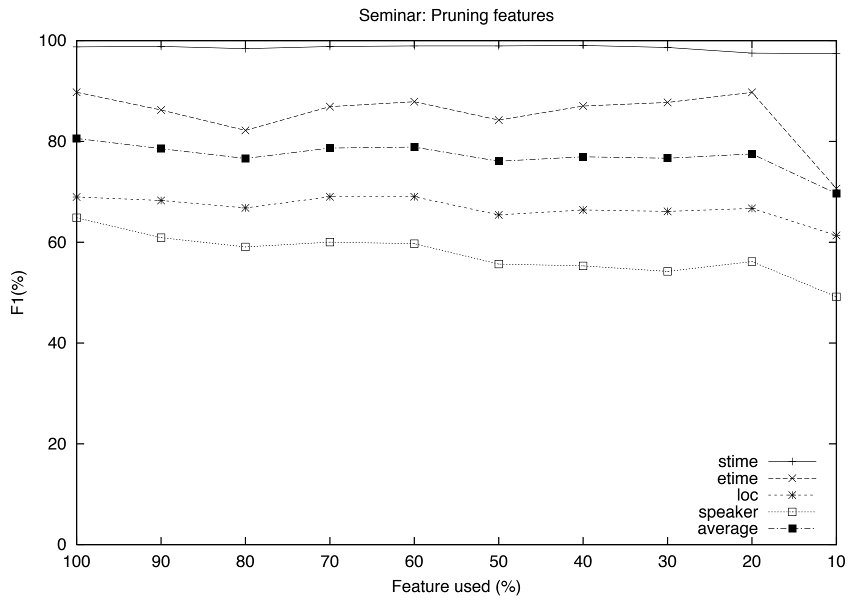

Figure 7: Eliminate infrequent features in *seminar* experiments (10-fold cross validation)

Figure 7 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. 7 represents the percentage of the tail of this histogram that was 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 shows that removing features this way (as opposed to pruning based on weights, for example) does not degrade the results, as long as 20% of the frequent features are maintained. This agrees with results reported in other domains, e.g, in (Carlson et al., 2001).

The performance on the *speaker* slot is the only exception, since it declines gradually as features are eliminated. This may explain why our system performs so well on this difficult slot. For this complex slot, many features that represent "exceptions" need to be taken into account to guarantee a good prediction. Unlike traditional ILP learners, learning via a linear threshold function is able to

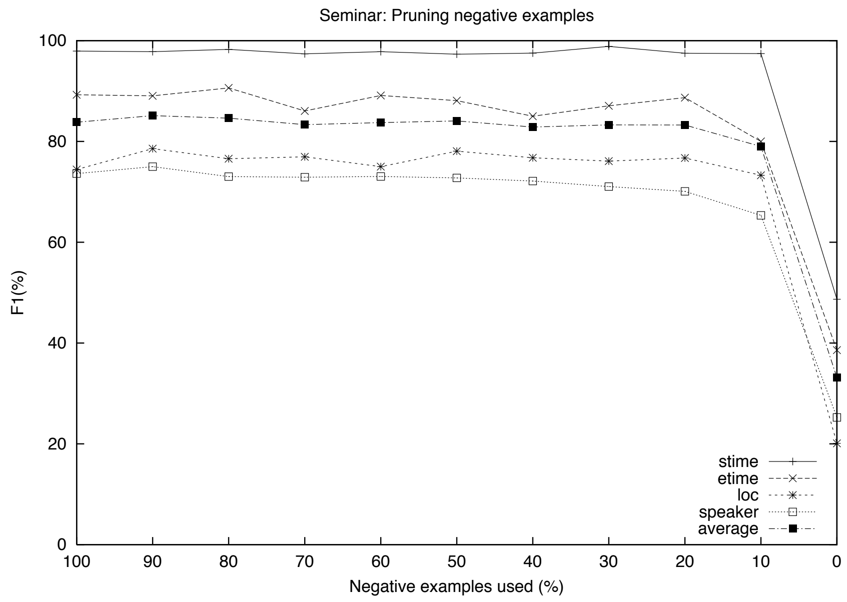

Figure 8: Eliminate negative examples randomly in *seminar* experiments (10-fold cross validation)

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 not suffer at all from pruning a large number of the features.

Figure 8 shows the effect of eliminating negative examples from the training set. As discussed earlier, in IE tasks, the negative examples vastly outnumber the positive examples, and this has significant computational consequences. This experiment addresses the question of whether all the negative examples are needed for training.

Several "sophisticated" methods (e.g., the filters in our 2-stage architecture) can be used to address this problem. These methods attempt to eliminate only negative examples that are "easy", and thus may not contribute to learning. In this experiment we have considered a very simple minded approach – we randomly eliminate $x\%$ of the negative examples for each slot separately.

The results clearly show that this strategy is very useful – there is no adverse effect on the results as long as sufficiently many negative examples, about 20% of them, are retained. As mentioned in Sec. 4.3, the highly skewed class distribution may be the explanation. The main lesson here is that the performance degrades significantly only when (almost) only positive examples are used. The reason is that the learning algorithm has no way to identify irrelevant fragments. Note that the 2-stage architecture still gives the best F-measure compared to this random sampling strategy. In addition, while random sampling of negative examples can help to cut the size of training data, the number of negative testing examples is still unaffected. This is an important issue when complicated RGFs are used and the time spent on feature extraction is significant.

6. Discussion

We discuss some subtle issues in the comparison between our method and other propositionalization methods or traditional ILP approaches in this section.

One key difference between the traditional ILP approach and a propositionalization method such as 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 quantified variables in them. The learning algorithm will look at all of them "in parallel" and find the best representation. This is because feature generation and learning are decoupled into two stages explicitly. Search control methods used by ILP methods are thus analogous to the expressivity or bias we give to our RGFs. The order of "visiting" these features is different.

Another difference we would like to discuss is the expressivity of learning or representing complex concepts using our approach. A nice property of ILP is the better comprehensibility of the FOL rules it learns, which is somewhat preserved in earlier propositionalization approaches, such as when learning algorithms like decision trees are used. In our case, when using linear threshold functions as the hypothesis space, we may lose some of it (but see below) for the benefit of gaining expressivity – we represent "generalized rules", a linear threshold function, rather than the special case of conjunctive rules. A linear threshold function such as $\mathbf{1}(w_1x_1 + w_2x_2 + w_nx_n \geq \theta)$, where $\mathbf{1}(\cdot)$ is the indicator function, can represent concepts like *conjunctions*, *disjunctions* or *at least m*

out of n, etc. For example, $y = x_1 \vee x_3 \vee x_5$ can be represented by $y = \mathbf{1}(x_1 + x_3 + x_5 \geq 1)$, and $y =$ at least 2 of $\{x_1, x_3, x_5\}$ can be represented as $y = \mathbf{1}(x_1 + x_3 + x_5 \geq 2)$.

When learned over an expressive feature space like the one generated by our methods, a feature-efficient algorithm that learns a linear threshold function represents, in fact, a low degree DNF or CNF over the original feature space. 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 pose 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. For example, when an ILP algorithm such as FOIL is used, the length of each Horn clause is usually not long. In our framework, 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:

SY: reworded this sentence using FOIL as a specific example.

Example 6.1 *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:*

`speaker`(target) $\leftarrow$ before_targ(2, "Speaker") $\wedge$ contains(target,"Dr") $\wedge$ after_targ(1, *line-feed*)

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.

7. Conclusions

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 propositionalization approach for relational learning, which allows for the representation and learning of relational information using propositional means. We first define a language based on restricted FOL to facilitate relational feature generation. Conceptually, users can easily describe the types of feature functions they want to apply on the target data and active features are then extracted in a data-driven manner. In this language, domain knowledge can be encoded in sensors and various declarative biases are specified in the relation generation functions constructed using sensors and connective operators.

In addition to giving examples of using this language to represent some relational domains such as the kinship and mutagenesis problems, we focus on the task of information extraction. Relational features are first generated based on some straightforward RGFs. The classifiers are trained using SNoW and naive Bayes on top of these features. Among the learning frameworks we tested, the two-stage architecture maintains a good trade-off of the computational complexity and classification accuracy, which achieves fairly competitive results compared to existing ILP-based or -inspired methods and is more efficient.

Finally, we would like to emphasize that the method presented here is flexible and allows the use of propositional algorithms within it, including probabilistic approaches. Using the probability estimation or other real-valued output that indicates the prediction confidence, one can easily adjust the desired accuracy trade-off such as precision vs. recall, which may not be easily achieved using some traditional ILP approaches.

For future work, the most interesting direction is to investigate how the advantages of our approach can be used to move it more toward a "real" relational learning method. In order to reduce the burden of defining the "types" of features, we plan to study automatic ways of abstracting features and learning the RGFs needed for a given problem.

Acknowledgments

We thank RISE (RISE, 1998) for the data used in the experiments, and the anonymous reviewers for their valuable comments. This research is supported by NSF grants ITR-IIS-0085836, ITR-IIS-0085980 and IIS-9984168 and an ONR MURI Award.

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,-1], word&loc[1,4]],

   conjunct[word&loc[-1,-1], tag&loc[1,4]]

2. For `location` and `speaker`:

   conjunct[word&loc[-2,-1], tag[0,0], tag&loc[1,1]]

In the above RGFs, numbers in the brackets represent the range of the windows. For example, [-1,-1] indicates the element immediately before the target region, [0,0] means the target region, and [1,4] shows the first to the fourth elements immediately after the target region. Therefore, the first set of RGFs is a sparse structural conjunction of the word immediately before 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:

1. For `title`:

   phAllCap[0,0], phAllNotNum[0,0], phAllWord[0,0]

   conjunct[word&loc[-2,-1], tag&loc[1,1]]

   conjunct[tag&loc[-2,-1], tag&loc[1,1]]

2. For `recruiter`, `req_yrs_exp`:

   conjunct[word&[-1,-1], word&loc[1,1]]

   conjunct[word&[-1,-1], tag&loc[1,1]]

3. For `salary`,`state`, `city`, `language`, `platform`, `application`, `des_yrs_exp`, `req_degree`, `des_degree`, `area`:

   phAllCap[0,0], phAllNotNum[0,0], phAllWord[0,0]

   conjunct[word&loc[-2,-1]; colloc(word,word,word)[0,0]; tag&loc[1,1]]

   conjunct[word&loc[-2,-1]; colloc(word,word)[0,0]; tag&loc[1,1]]

   conjunct[word&loc[-2,-1]; word[0,0]; tag&loc[1,1]]

Among these RGFs, phAllCap[0,0], phAllNotNum[0,0] and phAllWord[0,0] check if the words in the target region are all capitalized, without numbers, or all alphabetical (without numbers and symbols), respectively. colloc(word,word,word)[0,0] extracts all the trigrams in the target region, and colloc(word,word)[0,0] extracts the bigrams.

SY: Clarification for the comment by Reviewer 2.

RGFs provide the means for adjusting declarative bias easily as the target domain changes, such as the RGFs here for the information extraction problems. However, designing RGFs should not be treated as creating very specialized features. Instead, most RGFs, such as the ones in the above example, are just simple combinations of basic sensors like the words, POS tags or numbers.

Appendix B. Dominant features

Tables 6 and 7 list some of the dominant features used by our predictors. The left column describes the features in the feature extraction language we use, and the right column describes them using standard FOL predicates. Notice that the global quantifiers are used only to describe the chain structure in the original graphical representation, and can be evaluated efficiently as stated in Sec. 3.3. When the chain structural information is present, only local quantifiers within each focused region are allowed.

The intended meaning of the predicates used is as follows:

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

Table 6: Some dominant features for slot `recruiter` in computer-related job posting data

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

Table 7: Some dominant features for slot `speaker` in seminar announcement data

- 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 at 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 421,300 for the seminar task and 240,574 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 245,081; for the recruiter slot it was 143,104.

References

Alphonse, E., & Rouveirol, C. (2000). Lazy propositionalisation for relation learning. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pp. 256–260.

Bournaud, I., Courtine, M., & Zucker, J.-D. (2003). Propositionalization for clustering: Symbolic relational descriptions. In Matwin, S., & Sammut, C. (Eds.), *The 12th International Conference on Inductive Logic Programming (ILP-02)*, pp. 1–16. Springer-Verlag. LNAI 2583.

Califf, M. (1998). *Relational Learning Techniques for Natural Language Information Extraction*. Ph.D. thesis, The University of Texas at Austin.

Califf, M., & Mooney, R. (1999). Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 328–334.

Califf, M., & Mooney, R. (2003). Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, *2003*(4), 177–210.

Carlson, A., Cumby, C., Rosen, J., & Roth, D. (1999). The snow learning architecture. Tech. rep. UIUCDCSR-99-2101, Department of Computer Science, University of Illinois at Urbana-Champaign.

Carlson, A. J., Rosen, J., & Roth, D. (2001). Scaling up context sensitive text correction. In *Proceedings of the National Conference on Innovative Applications of Artificial Intelligence*, pp. 45–50.

Chein, M., & Mugnier, M.-L. (1992). Conceptual graphs: Fundamental notions. *Revue d'Intelligence Artificielle*, *6*(4), 365–406.

Chieu, H., & Ng, H. (2002). A maximum entropy approach to information extraction from semi-structure and free text. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002)*, pp. 786–791.

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

Cohen, W., & Page, D. (1995). Polynomial learnability and inductive logic programming: Methods and results. *New Generation Computing*, *13*(3&4), 369–409.

Cook, D. J., & Holder, L. B. (1994). Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, *1*, 231–255.

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

Cumby, C., & Roth, D. (2000). Relational representations that facilitate learning. In *Proc. of the International Conference on the Principles of Knowledge Representation and Reasoning*, pp. 425–434.

Cumby, C. M., & Roth, D. (2003a). Learning with feature description logics. In Matwin, S., & Sammut, C. (Eds.), *The 12th International Conference on Inductive Logic Programming (ILP-02)*, pp. 32–47. Springer-Verlag. LNAI 2583.

Cumby, C. M., & Roth, D. (2003b). On kernel methods for relational learning. In Fawcett, T., & Mishra, N. (Eds.), *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003)*, pp. 107–114, Washington, DC, USA. AAAI Press.

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

DARPA (1995). *Proceedings of the 6th Message Understanding Conference*. Morgan Kaufman.

De Raedt, L. (1998). Attribute-value learning versus inductive logic programming: The missing links. In *The Eighth International Conference on Inductive Logic Programming (ILP-98)*.

Dehaspe, L., & Toivonen, H. (1999). Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, *3*, 7–36.

Even-Zohar, Y., & Roth, D. (2000). A classification approach to word prediction. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics (NAACL-00)*, pp. 124–131.

Even-Zohar, Y., & Roth, D. (2001). A sequential model for multi-class classification. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP-01)*, pp. 10–19.

Flach, P., & Lachiche, N. (1999). 1BC: A first-order Bayesian classifier. In Džeroski, S., & Flach, P. (Eds.), *The 9th International Conference on Inductive Logic Programming (ILP-99)*, Vol. 1634 of *LNAI*, pp. 92–103. Springer-Verlag.

Flach, P., & Lachiche, N. (2001). Confirmation-guided discovery of first-order rules with tertius. *Machine Learning*, *42*(1/2), 61–95.

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

Freitag, D., & McCallum, A. (2000). Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI 2000)*, pp. 584–589.

Geibel, P., & Wysotzki, F. (1996). Relational learning with decision trees. In *Proceedings of the 12th European Conference on Artificial Intelligence*, pp. 428–432.

Goadrich, M., Oliphant, L., & Shavlik, J. (2006). Gleaner: Creating ensembles of first-order clauses to improve recall-precision curves. *Machine Learning*, *64*(1-3), 231–261.

Golding, A. R., & Roth, D. (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.

Har-Peled, S., Roth, D., & Zimak, D. (2002). Constraint classification: A new approach to multiclass classification and ranking. In *Neural Information Processing Systems*.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, Mass.

Khardon, R., Roth, D., & Valiant, L. G. (1999). Relational learning for NLP using linear threshold elements. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 911–917.

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

Kramer, S., & De Raedt, L. (2001). Feature construction with version spaces for biochemical applications. In *Proceedings of the 18th International Conference on Machine Learning (ICML-01)*.

Kramer, S., & Frank, E. (2000). Bottom-up propositionalization. In *Work-In-Progress Track at the 10th International Conference on Inductive Logic Programming*, pp. 156–162.

Kramer, S., Lavrac, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In Dzeroski, S., & Lavrac, N. (Eds.), *Relational Data Mining*, pp. 262–291. Springer-Verlag.

Krogel, M.-A., Rawles, S., Zelezny, F., Flach, P., Lavrac, N., & Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In Horvth, T., & Yamamoto, A. (Eds.), *The 13th International Conference on Inductive Logic Programming (ILP-03)*, pp. 197–214. Springer-Verlag. LNAI 2835.

Krogel, M.-A., & Wrobel, S. (2001). Transformation-based learning using multirelational aggregation. In Rouveirol, C., & Sebag, M. (Eds.), *The 11th International Conference on Inductive Logic Programming (ILP-01)*, pp. 142–155. Springer-Verlag. LNAI 2157.

Lavrac, N., Dzeroski, S., & Grobelnik, M. (1991). Learning nonrecursive definitions of relations with LINUS. In *Proceedings of the European Working Session on Learning : Machine Learning (EWSL-91)*, pp. 265–281. Springer-Verlag. LNAI 482.

Lavrac, N., Zelezny, F., & Flach, P. (2003). RSD: Relational subgroup discovery through first-order feature construction. In Matwin, S., & Sammut, C. (Eds.), *The 12th International Conference on Inductive Logic Programming (ILP-02)*, pp. 149–165. Springer-Verlag. LNAI 2583.

Lavrac, N., & Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York.

Lavrac, N., & Flach, P. A. (2001). An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, *2*(4), 458–494.

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 Progamming*. Springer-Verlag.

Michie, D., Muggleton, S., Page, D., & Srinivasan, A. (1994). To the international computing community: A new East-West challenge. Tech. rep., Oxford University Computing laboratory, Oxford,UK.

Mooney, R. (1997). Inductive logic programming for natural language processing. In *Proceedings of the 6th International Workshop on Inductive Logic Programming (ILP-96)*, pp. 3–24. Springer-Verlag. LNAI 1314.

Muggleton, S. (1992). Inductive logic programming. In Muggleton, S. (Ed.), *Inductive Logic Programming*, pp. 1–27. Academic Press.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*(3–4), 245–286.

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

Neville, J., Jensen, D., Friedland, L., & Hay, M. (2003). Learning relational probability trees. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-03)*, pp. 625–630.

Noreen, E. W. (1989). *Computer Intensive Methods for Testing Hypotheses*. John Wiley & Sons, Inc.

Perlich, C., & Provost, F. (2003). Aggregation-based feature invention and relational concept classes. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-03)*, pp. 167–176.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

Richards, B., & Mooney, R. (1992). Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 50–55.

Riloff, E. (1993). Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 811–816.

RISE (1998). A repository of online information sources used in information extraction tasks. http://www.isi.edu/info-agents/RISE/index.html. University of Southern California, Information Sciences Institute.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan, New York.

Roth, D. (1998). Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 806–813.

Roth, D., & Yih, W. (2001). Relational learning via propositional algorithms: An information extraction case study. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pp. 1257–1263.

Sebag, M., & Rouveirol, C. (1997). Tractable induction and classification in FOL. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 888–892.

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

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

Sowa, J. (1984). *Conceptual structures in mind and machines*. Addison-Wesley.

Srinivasan, A., & King, R. (1996). Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. In Muggleton, S. (Ed.), *The 6th International Conference on Inductive Logic Programming (ILP-96)*, pp. 89–104. Springer-Verlag. LNAI 1314.

Srinivasan, A., Muggleton, S., King, R., & Sternberg, M. (1996). Theories for mutagenicity: a study of first-order and feature based induction. *Artificial Inteligence*, *85*(1-2), 277–299.

Zelezny, F., & Lavrac, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, *62*(1-2), 33–63.