Learning and Inference for Information Extraction

Wen-tau Yih

CS Report No. UIUCDCS-R-2005-2534

Engr. Report No. UILU-ENG-2005-1726

May 2005

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

LEARNING AND INFERENCE FOR INFORMATION EXTRACTION

BY

WEN-TAU YIH

B.S., National Taiwan University, 1995
M.S., National Taiwan University, 1997

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

Information extraction is a process that extracts limited semantic concepts from text documents and presents them in an organized way. Unlike several other natural language tasks, information extraction has a direct impact on end-user applications. Despite its importance, information extraction is still a difficult task due to the inherent complexity and ambiguity of human languages. Moreover, mutual dependencies between local predictions of the target concepts further increase difficulty of the task. In order to enhance information extraction technologies, we develop general approaches for two aspects – *relational feature generation* and *global inference with classifiers*.

It has been quite convincingly argued that relational learning is suitable in training a complicated natural language system. We propose a relational feature generation approach that facilitates relational learning through propositional learning algorithms. In particular, we develop a relational representation language to produce features in a data driven way. The resulting features capture the relational structures of a given domain, and therefore allow the learning algorithms to effectively learn the relational definitions of target concepts.

Although the learned classifier can be used to directly predict the target concepts, conflicts between the labels of different target variables often occur due to imperfect classifiers. We propose an inference framework to correct mistakes of the local predictions by using the predictions and task-dependent constraints to produce the best global assignment. This inference framework can be modeled by a Bayesian network or integer linear programming.

The proposed learning and inference frameworks have been applied to a variety of information extraction tasks, including entity extraction, entity/relation recognition, and semantic role labeling.

To Ting, Fannie, and my parents

Acknowledgments

Although finishing a PhD is just a small milestone in life, I would not have been able to complete it without the help from many people.

I am very grateful to my wife Ting for her love and support during this journey. One adventure we lived through during this period was the birth of our daughter Fannie, who brought us a joyful and exciting dimension to our life. Thanks to my parents, sister and brother. Their faith in me brought me the strength to overcome the difficulties I encountered.

I deeply thank my advisor, Dan Roth, who not only provided me stimulating suggestions and guidance, but also encouraged me to pursue various goals. During these years working closely with Dan, I have known him as a patient, sympathetic and enthusiastic person. His brilliant insight into problems often opened doors hidden to me. In the meantime, his patience and tolerance allowed me to grow and improve myself from mistakes. Besides being an excellent advisor, Dan was as close as a good friend to me. I am really glad that I had this opportunity to know Dan in my life.

My colleagues and friends from the Department of Computer Science supported me both in research and in life. I am thankful to them for all their help. Especially, I am obliged to Shivani Agarwal, Fabio Aiolli, Kunal Bagga, Rodrigo de Salvo Braz, Andy Carlson, Xavier Carreras, Chad Cumby, Yair Even-Zohar, Gio Kao, Roxana Girju, Xin Li, Jakob Metzler, Paul Morie, Marcia Muñoz, Ramya Nagarajan, Vasin Punyakanok, Nick Rizzolo, Mark Sammons, Kevin Small, Samarth Swarup, Yuancheng Tu, and Dav Zimak.

Particularly I would like to thank Dav and Vasin. Dav helped me profoundly in shaping my writing and presentation. Also because of his humor, the working environment is full of joy everyday. As a coworker, I am always amazed by Vasin's insightful observations and clever strategies

in solving problems. I enjoy the privilege working with him.

I would also like to thank several researchers for their strict and extensive comments, as well as valuable discussions. Martha Palmer and Szu-ting Yi have helped me to better understand the semantic role labeling task, and also refined the presentation of this work. Xavier Carreras and Lluís Màrquez have exchanged many ideas with us and were generous with help in running experiments.

Finally, I would like to thank my dissertation committee, Dan Roth, Gerald DeJong, Jiawei Han, and ChengXiang Zhai for being willing to serve on my committee. I am very grateful for their comments, questions, and suggestions.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AI Artificial Intelligence.

CNF Conjunctive Normal Form.

DAG Directed Acyclic Graph.

DNF Disjunctive Normal Form.

FOL First-order Logic.

IE Information Extraction.

ILP Inductive Logic Programming or Integer Linear Programming.

LP Linear Programming.

LPR Linear Programming Relaxation.

MPE Most Probable Explanation.

MRF Markov Random Field.

NLP Natural Language Processing.

POS Part-of-speech.

QA Question Answering.

RGF Relation Generation Function.

SNoW Sparse Network of Winnow.

SRL Semantic Role Labeling.

Chapter 1

Introduction

Information extraction is a process that extracts limited semantic concepts from text documents, and presents them in an organized way, such as database tables. While information extraction covers a wide range of text processing problems, the target concepts are often both entities and the relations among the entities. The types of entities may have universal definitions such as *person*, *location*, or *organization*, which appear in different kinds of documents. On the other hand, entities can have definitions depending on the task at hand and apply to only certain collections of documents. For instance, given a collection of seminar announcements, the goal may be to locate the *starting time*, *location*, and *speaker* of each seminar.

Similarly, the number of relation types among entities can vary. It can be a small set, such as whether a location is the *headquarter* of an organization and if the extracted location and speaker refer to the *same* seminar. It can also be a large set, such as in the task of *semantic role labeling* (Kingsbury & Palmer, 2002), where each verb in a sentence represents a relation, and the goal is to identify its argument entities as defined in the templates (i.e., the frame files).

Unlike several other natural language tasks such as part-of-speech tagging or syntactic parsing, information extraction is either the end-user application or the task that has an immediate impact on the end-user application. For a knowledge discovery system (e.g., Craven, DiPasquo, Freitag, McCallum, Mitchell, Nigam, & Slattery, 1998) that aims to *understand* the documents, and thus transfer the Web into Semantic Web, information extraction is the key technology for discovering the concepts of interest hidden in the text collection. For an open-domain question answering

system (Voorhees, 2000), where the goal is often to find specific entities or relations defined in the question, the correctness of information extraction directly determines the system performance.

Despite its importance, information extraction is in general, still a difficult task. As in other natural language processing problems, the main difficulty comes from the inherent complexity and ambiguity of human languages. Early information extraction systems relied on linguistic and domain experts to hand-craft rules for each target concept. The performance of such systems highly depends on the expertise and experience of the knowledge engineers. In addition, it is difficult to extend the systems to new domains since they often do not scale and must be re-engineered with different expert knowledge – a tedious process that requires tremendous man power.

Since late 90s, various machine learning based information extraction systems have been built by training different models using annotated corpora (e.g., Califf & Mooney, 1999; Freitag & McCallum, 2000; Craven & Slattery, 2001). As in many other machine learning based systems, the performance is not only decided by the learning algorithm, but is also remarkably biased by the quality of feature functions. Although feature engineering is not as arduous as crafting rules, the complexity of finding and implementing good features is often overlooked and not reported.

Another aspect of information extraction involves the mutual dependencies that exist among the predictions of various target concepts. For instance, constraints exist among the labels of a relation and its argument entities. Producing a coherent global assignment to the variables representing different target concepts is nontrivial because the number of possible global assignments grows exponentially in the number of variables.

In order to enhance information extraction technologies, we develop general approaches for two aspects – *relational feature generation* to facilitate learning complicated concepts and efficient feature generation, and *global inference with classifiers* to exploit the mutual dependencies in the domain.

Relational feature generation Because of the complexity in natural language, it has been quite convincingly argued that relational learning is suitable in training a natural language system (Mooney,

1997). The natural method for relational learning is *inductive logic programming* (Lavrac & Dzeroski, 1994), which extracts features implicitly during the learning process. Alternatively, we develop an approach for relational learning that allows for the representation and learning of relational information using propositional means. As a result, the features are able to both represent the relational structures in a given domain and and provide a suitable representation for propositional learning algorithms to effectively learn the relational definitions of target concepts.

Inference with classifiers Although the learned classifier can be used to directly predict the target concepts, conflicts between the labels of different target variables often occur due to imperfect classifiers. For example, when a relation is predicted as *headquarter*, neither of its two entity arguments can be *person*. Such knowledge is encoded as constraints that may vary as the problem domain changes.

We propose an inference framework to correct mistakes of the local predictions by using the predictions and task-dependent constraints to produce the best global assignment. We develop two approaches for this problem. The first models the labels of the target concepts as random variables in a Bayesian network (Pearl, 1988). Finding the best assignment is defined as finding the assignment that maximizes the joint probability. The second approach treats this problem as an optimization problem solved using integer linear programming program. This approach allows us to efficiently incorporate domain and task specific constraints at decision time, resulting in significant improvements in the accuracy and the "human-like" quality of final predictions.

1.1 Overview

The rest of this dissertation are organized as follows. Chapter 2 presents some fundamental tools and approaches used in different information extraction tasks, which include the machine learning system SNoW (Carleson, Cumby, Rosen, & Roth, 1999), Bayesian networks (Pearl, 1988), and integer linear programming (Wolsey, 1998).

Chapter 3 describes our framework for relational feature generation. We develop a relational representation language along with a relation generation function to produce features in a data driven way; together, these allow for efficient representation of the relational structure in a given domain, in a way that is suitable for use by propositional learning algorithms. This framework serves as the basic feature generation mechanism, and supports all the learning systems described in the following chapters.

Chapter 4 first applies our learning framework based on relational feature generation to domain specific information extraction tasks. It aims to fill predefined templates with phrases extracted from documents of the same sort. We show results on *seminar announcement* and *job posting* data sets.

Chapter 5 develops a general framework for recognizing relations and entities in sentences, while taking mutual dependencies among them into account. For example, the *kill (KFJ, Oswald)* relation in: "`J. V. Oswald was murdered at JFK after his assassin, R. U. KFJ...`" depends on identifying Oswald and KFJ as *people*, JFK being identified as a *location*, and the *kill* relation between Oswald and KFJ; this, in turn, enforces our belief that Oswald and KJF are *people*.

We develop two inference approaches: the first based on Bayesian network and the second using integer linear programming. We evaluate both approaches in the context of simultaneously learning named entities and relations. Our approaches allow us to efficiently incorporate domain and task specific constraints at decision time, resulting in significant improvements in the accuracy and the "human-like" quality of the inference.

Chapter 6 addresses the problem of *semantic role labeling*, which can be treated as an extended version of relation recognition. In this chapter, we present a general framework for semantic role labeling. It combines a machine learning technique with an inference procedure based on integer linear programming that incorporates linguistic and structural constraints into the decision process. The system is tested on the data provided in the CoNLL-2004 shared task on semantic role labeling and achieves very competitive results. In addition, we experimentally study the necessity

of syntactic parsing for semantic role labeling. As a result, we develop a state-of-the-art semantic role labeling system by combining several systems based on different full parse trees.

Finally, Chapter 7 concludes this dissertation and provides several directions for future research.

Chapter 2

Background

This chapter briefly introduces some fundamental tools or approaches that are used in different tasks. Section 2.1 describes the learning system SNoW, which is applied to all the classification tasks in this dissertation. In order to solve the global inference problem over classifiers, we have developed two different approaches based on Bayesian networks (used in Chapter 5) and integer linear programming (used in Chapter 5 and 6), which are described in Section 2.2 and Section 2.3, respectively.

2.1 Sparse Network of Winnows (SNoW)

SNoW [1](Sparse Network of Winnows) (Roth, 1998; Khardon, Roth, & Valiant, 1999; Carleson, Cumby, Rosen, & Roth, 1999) is a multi-class learning architecture that is specifically tailored for large scale learning tasks. It is suitable for learning in a high dimensional feature space, especially when the input examples are sparse. Its architecture is a two-layer sparse network of linear functions. The nodes in the first layer (feature nodes) represent the input features, and are allocated in a data-driven way, given the variable length examples. The nodes in the second layer (target nodes) represent the target classes, which are linear functions over a common feature space. The weights of the linear functions are stored on the links between the target nodes and feature nodes. The network is sparse in that a link only appears when the corresponding feature is active frequently

[1]available at http://l2r.cs.uiuc.edu/∼cogcomp

enough given the target class in training examples.

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]).

SNoW employs several linear update rules including Perceptron, naive Bayes, and Winnow (a variation of (Littlestone, 1988)). Winnow is a mistake driven on-line algorithm that is similar to Perceptron (Rosenblatt, 1962), but updates its weights in a multiplicative way. Its key advantage is that the number of examples required to learn the target function grows linearly in the number of relevant features, but only logarithmically in the total number of features.

Regularization in SNoW can be achieved in the following two types of modifications in the Winnow and Perceptron update rules. The first one is to learn a "thick hyperplane" in the binary classification problem represented by each subnetwork. This modification is similar to the idea of margin Perceptron (Li, Zaragoza, Herbrich, Shawe-Taylor, & Kandola, 2002), and it is not difficult to show that the mistake bound still depends on the margin of the data in addition to the thickness parameter. The other modification implemented in SNoW is the idea of "voted Perceptron" (Freund & Schapire, 1999). Instead of using the final weight vector as the hypothesis, the averaged vector of all intermediate weights during training is generated as output. Both approaches have shown their effectiveness in practice, and can be applied jointly.

While SNoW is usually used as a classifier and predicts using a winner-take-all mechanism over the activation values of the target classes, the activation values can also help in estimating the posteriors of each class given the features. The raw activation value SNoW outputs is the weighted linear sum of the features. It can be verified that the activation values are monotonic with the

[2]SNoW actually supports a more general multi-class framework (Har-Peled, Roth, & Zimak, 2002).

confidence in the prediction, therefore is a good source of probability estimation.

We use the softmax (Bishop, 1995) function over the raw activation values as the conditional probability estimation. Specifically, suppose the number of classes is n, and the raw activation value of class i is act_i. The posterior estimation for class i is derived by the following equation.

$$p_i = \frac{e^{act_i}}{\sum_{j=1}^{n} e^{act_j}}$$

SNoW has been shown to be successful in a variety of natural language and computer vision tasks (Roth, 1998; Roth & Yih, 2001; Golding & Roth, 1999; Roth, Yang, & Ahuja, 2000; Roth, Yang, & Ahuja, 2002). More discussion of SNoW can be found in (Carleson, Cumby, Rosen, & Roth, 1999; Roth, 1998; Carlson, Rosen, & Roth, 2001) and its user manual (Carlson, Cumby, Rosen, Rizzolo, & Roth, 2004).

2.2 Bayesian Network

Broadly used in the AI community, *Bayesian network* (also known as *belief network*) is a graphical representation of a joint probability distribution (Pearl, 1988). It is a directed acyclic graph (DAG), where the nodes are random variables and a link from node A to node B can be conceptually viewed as A causes B. Each node in a Bayesian network is associated with a conditional probability table (CPT), which defines the conditional probability distribution given its parents. In addition, the variable is conditionally independent of other variables given its parent node.

Suppose a Bayesian network consists of variables $X_1, X_2, \ldots, X_n$. Because of the property of conditional independence, the joint probability can be factorized as

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | Parent(X_i)),$$

where $Parent(X_i)$ are the parent nodes of variable X_i. If $X_1, X_2, \ldots, X_n$ are binary variables, and k is the maximum number of the parent nodes that a node can have, then the space needed

for storing these CPTs is $O(n \cdot 2^k)$, which is much less than the size of storing explicitly the joint probabilities, $O(2^n)$. Therefore, a Bayesian network provides a compact way to represent the joint probabilities.

Given a Bayesian network, there are several interesting types of inference tasks. For example, given the evidence, what is the probability of the most likely cause? Another example is, what is the probability distribution of a specific variable X_i? Among those inference tasks, the most-probable-explanation (MPE) inference problem, which is to find the assignment to all the variables in the Bayesian network that maximizes the joint probability is often the most interesting one.

Since a Bayesian network uniquely defines a joint probability distribution, conceptually we can answer all the inference queries by marginalization. However, given the exponential number of joint variable assignments, the brute-force approach is usually not feasible in practice. For instance, the MPE problem can be solved by Pearl's belief propagation algorithm (Pearl, 1988) in linear time if the network is singly connected (i.e. network without undirected cycles, or loops), but it is intractable (Roth, 1996) in general.

Recently, researchers have achieved great success in solving the problem of decoding messages through a noisy channel with the help of Bayesian networks (MacKay, 1999). The network structure used in the problem is a loopy bipartite DAG. The inference algorithm used is Pearl's belief propagation algorithm (Pearl, 1988), which outputs exact posteriors in linear time if the network is singly connected (i.e., without loops) but does not guarantee to converge for loopy networks. However, researchers have empirically demonstrate that by iterating the belief propagation algorithm several times, the output values often converge to the right posteriors (Murphy, Weiss, & Jordan, 1999).

2.3 Linear Programming Formulation

Linear programming (LP) formulation is a powerful tool for optimization, and has been used in the global inference procedure in several tasks in this dissertation. In this section, we first briefly

introduce the basic idea of linear programming, followed by the description of integer linear programming. Different techniques of solving integer linear programming problems will also be discussed.

2.3.1 Linear Programming

Given a finite number of linear equality and inequality constraints, *linear programming* is the process of searching a solution that satisfies the constraints and minimizes (or maximizes) a linear cost function.

Let $\mathbf{x}, \mathbf{c} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^k, \mathbf{d} \in \mathbb{R}^l$, $\mathbf{A}$ is a $k \times n$ matrix, and $\mathbf{C}$ is a $l \times n$ matrix. A general linear program has the following form.

$$\begin{aligned} \min \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{C}\mathbf{x} \leq \mathbf{d} \\ & x_1 \geq 0, \cdots, x_r \geq 0 \\ & x_{r+1} \neq 0, \cdots, x_n \neq 0 \end{aligned}$$

Either k or l can be 0 and $1 \leq r \leq n$.

By introducing slack variables, a general linear program can be easily transformed into the following two special forms.

$$\begin{aligned} \text{Standard form:} \quad \min \quad & \mathbf{c} \cdot \mathbf{x} \\ s.t. \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

$$\text{Canonical form:} \quad \min \quad \mathbf{c} \cdot \mathbf{x}$$
$$s.t. \quad \mathbf{A}\mathbf{x} \geq \mathbf{b}$$
$$\mathbf{x} \geq 0$$

That is, in the standard form, only equality constraints are allowed, while in the canonical form, only inequality constraints are allowed. The variables in both forms are sign-constrained (i.e., $\mathbf{x} \geq 0$).

The constraints in linear programming define a *feasible solution space*. Although it is generally infinite, linear programming can be solved efficiently because an optimal solution only occurs at *extreme points*. We introduce this theorem starting from the following definitions.

Definition 2.3.1 (Convex Set) *A subset S of $\mathbb{R}^n$ is called* convex *if for any two distinct points $\mathbf{x}_1$ and $\mathbf{x}_2$ in S, any point*

$$\mathbf{x} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2, 0 \leq \lambda \leq 1$$

is also in $\mathbb{R}^n$.

Definition 2.3.2 (Convex Combination) *A point $\mathbf{x} \in \mathbb{R}^n$ is a* convex combination *of the points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r \in \mathbb{R}^n$, if*

$$\mathbf{x} = \sum_{i=1}^{r} c_i \mathbf{x}_i$$

for some real numbers $c_1, c_2, \ldots, c_r$ which satisfy

$$\sum_{i=1}^{r} c_i = 1 \text{ and } c_i \geq 0, 1 \leq i \leq r$$

Theorem 2.3.1 *The set of all convex combinations of a finite set of points in $\mathbb{R}^n$ is a convex set.*

Because the constraints in linear programming are linear, we have the following theorem.

Theorem 2.3.2 *The feasible solution space of linear programming is a convex set.*

Definition 2.3.3 (Extreme Point) *A point* $\mathbf{x}$ *in a convex set* S *is called an* extreme point *(or* vertex*) of* S *if there are no distinct points* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *such that*

$$\mathbf{x} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2, 0 < \lambda < 1.$$

Theorem 2.3.3 *Let* S *be the feasible solution space of a linear programming problem. If* S *is nonempty and bounded, then an optimal solution to the problem exists and occurs at an extreme point.*

The geometric idea of extreme points can be found though simple algebraic operations. Here we investigate their relation. Consider the following linear program in its standard form

$$\text{Standard form:} \quad \min \quad \mathbf{c} \cdot \mathbf{x} \tag{2.1}$$

$$s.t. \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.2}$$

$$\mathbf{x} \geq 0, \tag{2.3}$$

where $\mathbf{A}$ is an $m \times n$ matrix of rank m, $m \leq n$, $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$.

Let the columns of $\mathbf{A}$ be denoted by $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$. We can then write (2.2) as

$$x_1\mathbf{A}_1 + x_2\mathbf{A}_2 + \cdots + x_s\mathbf{A}_s = \mathbf{b} \tag{2.4}$$

Definition 2.3.4 (Basic Solution & Basic Feasible Solution) *Given the above linear program, if columns* $j_1 < j_2 < \cdots < j_m$ *are linearly independent, then the solution* $\mathbf{x}'$ *to (2.2)*

$$x'_{j_1}\mathbf{A}_{j_1} + x'_{j_2}\mathbf{A}_{j_2} + \cdots + x'_{j_m}\mathbf{A}_{j_m} = \mathbf{b}$$

is called a basic solution*, and variables* $x_{j_1}, \ldots, x_{j_m}$ *are called* basic *variables. Any variable* x_l *where* $l \notin \{j_1, \ldots, j_m\}$ *is set to 0 and is called* nonbasic*. A basic solution* $\mathbf{x}'$ *is called a* basic feasible solution *(or* bfs *for short) if it satisfies (2.3) (i.e.* $\mathbf{x}' \geq 0$*).*

A basic solution can be derived as follows. Let $\mathbf{B}$ be the nonsingular submatrix consists of columns $j_1, j_2, \ldots, j_m$ of $\mathbf{A}$. $\mathbf{B}^{-1}\mathbf{b}$ is the solution to the basic variables. Basic solutions are important because of the following theorem.

Theorem 2.3.4 *Let S be the convex set of the feasible solution space defined by (2.2) and (2.3). A point $\mathbf{x}' \in \mathbb{R}^n$ is an extreme point of S if and only if $\mathbf{x}'$ is a basic feasible solution.*

Because there are only finitely many extreme points in the feasible solution space formed by a linear program. An optimal solution can be found by exhaustively checking the value of the cost function on each basic feasible solution.

In fact, the well-known *Simplex Algorithm* designed by G. Dantzig in 1947 is based on this idea. Starting from a *bfs*, the Simplex Algorithm repeatedly checks the neighboring points, where only one basic variable differs. It can be shown that once the algorithm finds a local minimum, it is also a global minimum, and therefore the optimal solution to the linear programming problem.

2.3.2 Integer Linear Programming

Integer linear programming (ILP) or *integer programming* is the problem of linear programming that requires the solution to be integer. Similar to the standard and canonical forms of linear programming, we can have the following forms of integer linear programming.

$$\min\{\mathbf{c} \cdot \mathbf{x} | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} \text{ integer}\} \tag{2.5}$$

$$\min\{\mathbf{c} \cdot \mathbf{x} | \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} \text{ integer}\} \tag{2.6}$$

Integer linear programming in general is NP-hard. However, when the problem size is within hundreds of variables and constraints, it can usually be solved efficiently by commercial numerical packages, such as Xpress-MP (2004) or CPLEX (2003). We introduce some general strategies of solving an integer linear programming problem here.

Linear Programming Relaxation (LPR)

To solve an ILP problem, a natural idea is to *relax* the integer constraints by dropping such constraints, and tries to find the solution using a general linear programming solver.

If LPR returns an integer solution, then it is also the optimal solution to the ILP problem. If the solution is non-integer, then at least it gives an lower bound to the value of the cost function, which can be used to modify the problem in order to get closer to an optimal integer solution. A direct way to handle the non-integer solution is called *rounding*, which finds an integer point that is close to the non-integer solution, and hopefully still maintains the small cost. To find a good rounding method is not always easy, and the cost may deviates from the optimal value a lot. When the cost function is in some specific form and satisfies some conditions, a well designed rounding algorithm can be shown that the rounded solution is a good approximation to the optimal solution (Kleinberg & Tardos, 1999; Chekuri, Khanna, Naor, & Zosin, 2001). Nevertheless, in general, the outcome of the rounding procedure may not even be a legal solution to the problem.

Branch & Bound and Cutting Plane

Branch and bound is the method that divides an ILP problem into several LP subproblems. It uses LPR as a subroutine to generate dual (upper and lower) bounds to reduce the search space, and finds the optimal solution as well. When LPR finds a non-integer solution, it splits the problem on a non-integer variable. For example, suppose variable x_i is fractional in a non-integer solution to the ILP problem $\min\{cx : x \in S, x \in \{0,1\}^n\}$, where S is the linear constraints. The ILP problem can be split into two sub LPR problems, $\min\{cx : x \in S \cap \{x_i = 0\}\}$ and $\min\{cx : x \in S \cap \{x_i = 1\}\}$. Since any feasible solution provides an upper bound and any LPR solution generates a lower bound, the search tree can be effectively cut.

Another strategy of dealing with non-integer points, which is often combined with *branch & bound*, is called *cutting plane*. When a non-integer solution is given by LPR, it adds a new linear constraint that makes the non-integer point infeasible, while still keeps the optimal integer solution

in the feasible region. As a result, the feasible region is closer to the ideal polyhedron, which is the convex hull of feasible integer solutions. The most famous cutting plane algorithm is Gomory's fractional cutting plane method (Wolsey, 1998), which can be shown that only finite number of constraints are needed. In addition, researchers have developed different cutting plane algorithms for different types of ILP problems. One example is (Wang & Regan, 2000), which only focuses on binary ILP problems.

Although in theory, a search based strategy may need several steps to find the optimal solution, LPR often generates integer solutions in our experiments in this dissertation. This phenomenon may link to the theory of *unimodularity*.

Unimodularity

When the coefficient matrix of the linear program, either in its standard or canonical form, satisfies certain properties called *unimodular* or *totally unimodular* respectively, the linear program always has an optimal integer solution (Schrijver, 1986). In other words, LPR is guaranteed to produce an integer solution.

Definition 2.3.5 (Unimodular) *A matrix* $\mathbf{A}$ *of rank* m *is called* unimodular *if all the entries of* $\mathbf{A}$ *are integers, and the determinant of every square submatrix of* $\mathbf{A}$ *of order* m *is in 0,+1,-1.*

Definition 2.3.6 (Totally Unimodular) *A matrix* $\mathbf{A}$ *of rank* m *is called* totally unimodular *if the determinant of every square submatrix of* $\mathbf{A}$ *is in 0,+1,-1.*

Obviously, a totally unimodular matrix is also a unimodular matrix.

Theorem 2.3.5 (Veinott & Dantzig) *Let* $\mathbf{A}$ *be an* (m, n)*-integral matrix with full row rank* m*. Then the polyhedron* $\{\mathbf{x}|\mathbf{x} \geq 0; \mathbf{Ax} = \mathbf{b}\}$ *is integral for each integral vector* $\mathbf{b}$*, if and only if* $\mathbf{A}$ *is unimodular.*

Theorem 2.3.5 indicates that if a linear program is in its standard form, then regardless of the cost function and the integral vector $\mathbf{b}$, the optimal solution is integral if and only if the coefficient

matrix $\mathbf{A}$ is unimodular.

The reason is that the values of the basic variables in a bfs of a linear program always has the following form:

$$\mathbf{B}^{-1} \cdot \mathbf{b} = \frac{adj(\mathbf{B})}{||\mathbf{B}||} \cdot \mathbf{b}, \tag{2.7}$$

where $\mathbf{B}$ is a non-singular square submatrix of $\mathbf{A}$ of order m. Since $||\mathbf{B}||$ is either +1 or -1, and the optimal solution only appears on extreme points (i.e., basic feasible solutions), this linear programming problem can only have integer solutions.

For the canonical form of a linear program, we also have a similar theorem.

Theorem 2.3.6 (Hoffman & Kruskal) *Let $\mathbf{A}$ be an (m, n)-integral matrix with full row rank m. Then the polyhedron $\{\mathbf{x}|\mathbf{x} \geq 0; \mathbf{Ax} \leq \mathbf{b}\}$ is integral for each integral vector $\mathbf{b}$, if and only if $\mathbf{A}$ is totally unimodular.*

Note that even if the coefficient matrix is not unimodular or totally unimodular, an LP-solver can still always return integer solution given a special coefficient matrix $\mathbf{A}$, a specific vector $\mathbf{b}$, or a cost function with certain properties.

Chapter 3

Relational Learning via Propositionalization

This chapter 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 scalable and flexible and allows the use of any propositional algorithm, including probabilistic algorithms, within it.

3.1 Overview

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.

Inductive Logic Programming (ILP) is an active subfield of machine learning that addresses relational learning and is a natural approach to apply to these tasks. In principle, ILP methods allow induction over relational structures and unbounded data structures. However, studies in ILP 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 (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 definition, 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 chapter, 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, but results in rela-

tional 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. As we will see in later chapters, our propositionalization method serves as the main component for feature generation, and extracts numerous features in different learning tasks in the NLP domain. The next subsection provides a high-level view of this learning framework.

3.1.1 The Learning Framework

Our learning framework attempts to learn classifiers for relational learning problems by utilizing propositional learning algorithms. The key to our propositionalization technique is the decoupling of a data driven feature extraction stage and a feature efficient learning stage. The former 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. The latter uses a feature-efficient propositional learning algorithm to learn a function that represents the target property in terms of the extracted features.

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 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 individually quantified as in the predicate $\exists X author\_of(Roth, X) \wedge book(X)$.

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 features 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, or 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

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

Table 3.1: An example of propositionalization in the kinship domain

"focus" elements; therefore, our learning algorithm should be able to learn well in the presence of a large number of irrelevant features.

3.2 Related Work – Propositionalization

Propositionalization is a transformation that maps a relational representation to features in propositional form (Kramer, Lavrac, & Flach, 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 3.1.

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, Dzeroski, & Grobelnik, 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[1] (Krogel, Rawles, Zelezny, Flach, Lavrac, & Wrobel, 2003) takes flattened Prolog clauses (similar to those used in 1BC (Flach & Lachiche, 1999)) as its language. It generates 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.

[1] http://www.cs.bris.ac.uk/home/rawles/sinus/

RSD RSD is a system designed for relational subgroup discovery (Lavrac, Zelezny, & Flach, 2003; Krogel, Rawles, Zelezny, Flach, Lavrac, & Wrobel, 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, Courtine, and Zucker (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 and 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, 2002), 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, Lavrac, & Flach, 2001) and (Krogel, Rawles, Zelezny, Flach, Lavrac, & Wrobel, 2003).

Our propositionalization framework is based on a graphical representation which 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 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, 2002).

3.3 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 (Lavrac, Dzeroski, & Grobelnik, 1991) and (Khardon, Roth, & Valiant, 1999; Cumby & Roth, 2000), but technically is more related to the latter.

3.3.1 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 3.3.1 (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 these might be words, phrases, sentences or documents. Predicates used in $\mathcal{D}$ 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 constructs could be a single sentence, a document, etc.

Definition 3.3.2 *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, Courtine, & Zucker, 2003). Compared to (Bournaud, Courtine, & Zucker, 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 3.3.1 *The instance of a family domain in Figure 3.1 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.*

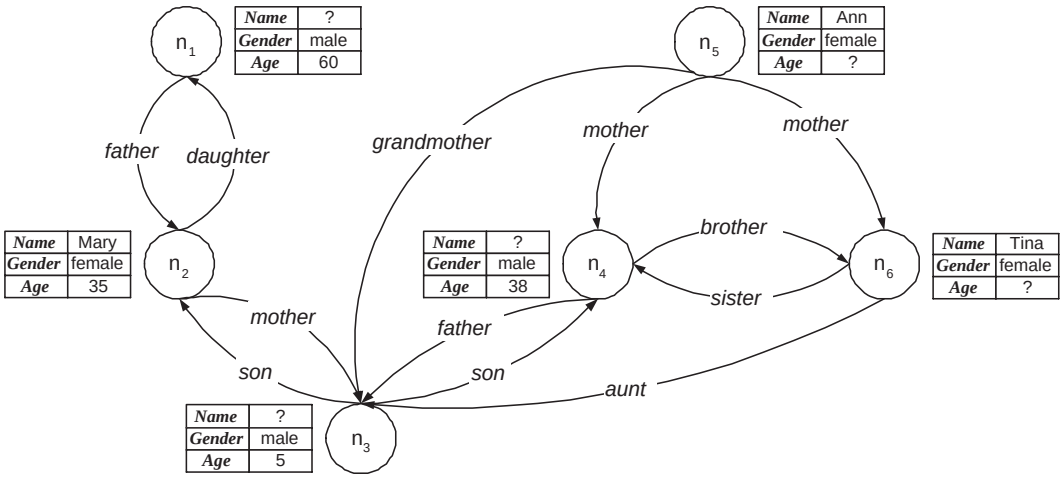

Figure 3.1: An instance of the kinship domain: each node (element) represents a person in this family. The associated attributes are *Name*, *Gender*, and *Age*. Some attribute values of an element may be unknown. The relation of two persons is denoted by the link connecting them.

Example 3.3.2 *Figure 3.2 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.*

3.3.2 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 (Definition 3.3.2). This is done by (1) defining primitive formulae with a limited scope of quantifiers (Definition 3.3.3), and (2) defining general formulae inductively, in terms of primitive formulae, in a restricted way that depends on the relational structures in the domain (Section 3.3.3).

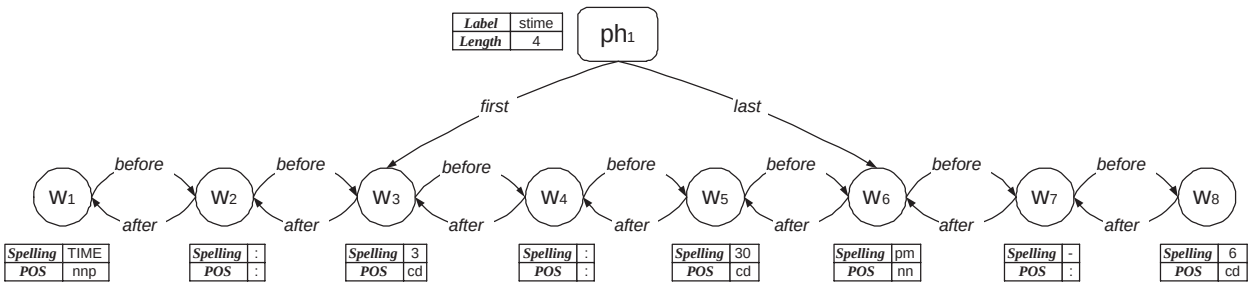

Figure 3.2: An instance in the text domain: there are two types of elements – *words* and *phrases*. Words $w_1, \cdots, w_8$ are connected by their positional relations, *before* and *after*. Each word has attributes *spelling* and *part-of-speech*. Phrase ph_1 has links pointing to its first and last word, and is associated with its label and length.

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 3.3.3 *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 $(\neg F)$, $(F \wedge G)$, $(F \vee G)$.*

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, Roth, & Valiant, 1999). The informal semantics of the quantifiers and connectives is the same as usual.

Definition 3.3.4 (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.

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 in the corresponding atomic formula, the following properties trivially hold. It will be clear that the way we extend the language (Section 3.3.3) maintains these properties.

Proposition 3.3.1 *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 3.3.2 (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 $f(F_1, \ldots F_n)$ on x can be evaluated in time $t_f + \sum_F t_F$, where the sum is over all n formulae that are arguments of f.*

3.3.3 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 evaluated efficiently.

Definition 3.3.5 *A formula in $\mathcal{R}$ maps an instance x to its truth value. It is* active *in x if it has truth value* true *in it. We denote by X the set of all instances, the* instance space. *A formula $F \in \mathcal{R}$ is thus a* relational feature *over X, $F : X \to \{0, 1\}$.*

Example 3.3.3 *Let instance x be the text fragment illustrated in Example 3.3.2. 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 3.3.6 *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 3.3.4 *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 3.3.7 *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 3.3.5 *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 3.3.8 *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 $r[E]$.*

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., to 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 Definition 3.3.3 (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 3.3.9 (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 3.3.6 *When applied with respect to the graph g which represents the linear structure of a sentence, $colloc_g$ generates formulae that correspond to n-grams. For example, 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 3.3.10 (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 3.3.7 *Given the fragment "Dr John Smith", the features generated by the RGF scol-loc(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 Proposition 3.3.1, 3.3.2, with the only additional cost of finding a chain in the graph.

3.3.4 Application Examples

Our propositionalization method has been applied to several classical ILP tasks (Cumby & Roth, 2002), 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.

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 Section 3.3.1, 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 in 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.

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.

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. Figure 3.3 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 in paths between two atoms. The relational features consist of the attributes of

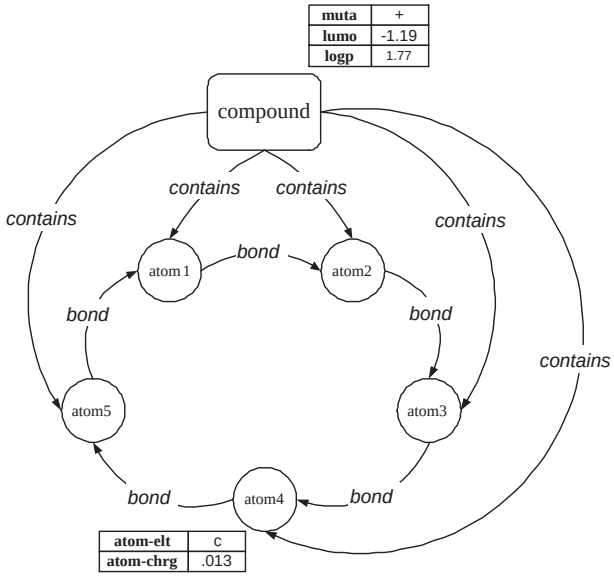

Figure 3.3: The graphical representation of a mutagenesis domain (taken and revised from (Cumby & Roth, 2003)): each domain has one *compound* element that has relations *contains* pointing to each *atom*; *atoms* have *bond* relations among them; each element has its own attributes.

the nodes extracted by predefined sensors and the bond-types labeled on the edges as well.

The system based on our framework achieves an accuracy of 88.6%, which is comparable to the results achieved in (Srinivasan, Muggleton, King, & Sternberg, 1996) using the Progol (Muggleton, 1992) system, and a little below the best results on this data set (Sebag & Rouveirol, 1997), which made use of richer features.

3.4 Discussion

In this chapter, we suggest a new propositionalization approach for relational learning, which allows for the representation and learning of relational information using propositional means. The method presented here is flexible and allows the use of any propositional algorithm within it, including probabilistic approaches. Consequently, this method can be used on a variety of relational learning problems, and plays the role of feature extractor in all the NLP applications in this dissertation. Below we discuss some subtle issues in the comparison between our method and other

propositionalization methods or traditional ILP approaches.

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 SNoW or naive Bayes are used, appropriate weights are learned for the features "in parallel", which makes the learning process more efficient.

A nice property of ILP is the better comprehensibility of the FOL rules it learns. This 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 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. 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 a logic program represents a disjunction over these, a linear threshold function over these relational features is more expressive. In this way, it may allow learning smaller programs. The following example illustrates the representational issues:

Example 3.4.1 *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:*

$$\texttt{speaker}(target) \leftarrow \texttt{before\_targ}(2, \text{“}\textit{Speaker}\text{”}) \wedge$$
$$\texttt{contains}(\textit{target}, \text{“}\textit{Dr}\text{”}) \wedge$$
$$\texttt{after\_targ}(1, \text{line-feed})$$

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.

Chapter 4

Extracting Entities

We first apply our learning framework proposed in Chapter 3 to a basic information extraction (IE) task. As introduced in Chapter 1, IE is a text processing task that attempts to extract items with specific semantic or functional meaning from unrestricted and unstructured text. In this chapter, we focus on extracting specifically defined entities from different sets of similar documents. Examples of this task include identifying speaker names in a seminar announcement and locating the title of a job posting. 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.

4.1 Overview

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). In this chapter, 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. 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) have demonstrated the success of ILP methods in this domain. Therefore, it is an interesting task to which our relational learning framework proposed in Chapter 3 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.

The rest of this chapter is organized as follows. We first describe our information extraction systems, including the data and templates, target representation, feature generation, and the system architecture in Section 4.2. The experimental results are summarized in Section 4.3, and some interesting issues are discussed in Section 4.4.

4.2 Task Description and System Design

4.2.1 Data and Templates

Under the same framework, we build two IE systems for different sets of documents – *seminar announcements* and *computer-related job postings*. The data and some target slots in the templates are described here.

Seminar Announcements

The first data set consists of 485 CMU seminar announcements[1]. 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 does not contain the slot at all).

Computer-related Job Postings

The second data set is a set of 300 computer-related job postings from the University of Texas at Austin[2]. 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.

4.2.2 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.

[1]The data set was originally collected from newsgroups and annotated by Dayne Freitag; it is available at (RISE, 1998).

[2]The data set was originally collected from newsgroups and annotated by Mary Califf; it is also available at (RISE, 1998).

$\cdots$ the [talk given by:] [Dr FName LName] [*line-feed* Topic : A] novel $\cdots$

Figure 4.1: Three regions for feature extraction

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 $1 \leq i \leq j \leq n$, constitutes 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 4.1 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 Figure 3.2. 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 4.2.4 and Figure 4.3.

4.2.3 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, Rosen, & Roth, 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 need to be generated in order to represent all possible fragments in a document. Among them, negative examples (irrelevant fragments) vastly outnumber positive examples (fragments that represent legitimate slots). In the seminar announcements data set, for example, the average length of an article is 200 words. If we limit the maximum length of fragments to 14 words, then each article would generate about 3300 candidate fragments, out of which about 10 fragments (i.e. 0.3%) represent legitimate slots. To handle this, we consider several methods to eliminate many of the negative training examples[3]. 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.

4.2.4 Two-stage Architecture

In order to efficiently eliminate negative examples without degrading accuracy, we propose a two-stage learning framework for the IE task. 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

[3]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 (i.e., only fragments of interest) will significantly degrade recognition performance.

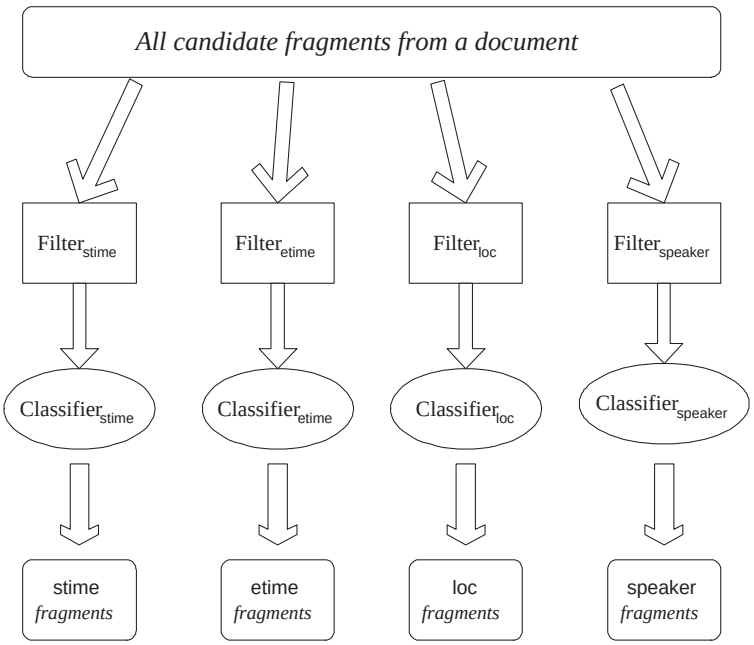

Figure 4.2: The two-stage architecture for seminar announcements data

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 4.2 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, which is an important issue given the small data set.

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.

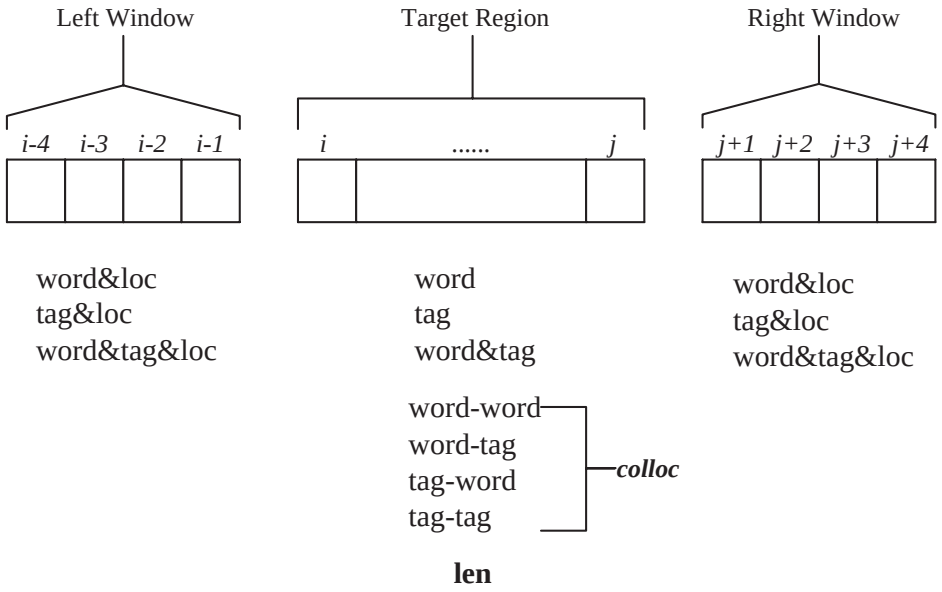

Figure 4.3: Graphical Illustration of a set of RGFs

- Left & Right window: *word&loc*, *tag&loc*, and *word&tag&loc*, where *loc* extracts the position of the word in the window.

The use of these RGFs to extract features that represent a fragment is demonstrated in Figure 4.3.

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 added for the *seminar announcements* 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] is the target region, and [1,4] means 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 POS 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 POS tag in the target, and the first tag in the right window.

The RGFs added for the *job postings* 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. In addition, colloc(word,word,word)[0,0] extracts all the trigrams in the target region, and colloc(word,word)[0,0] extracts the bigrams.

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.

4.3 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; Califf & Mooney, 2003), SRV (Freitag, 2000), and WHISK (Soderland, 1999)). We compare our results to theirs in Section 4.3.1. We also report the experimental results on the effects of pruning features and examples in Section 4.3.2.

4.3.1 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_1 (the harmonic mean of precision and recall). We estimate the 95% confidence intervals of

System	2-SNoW-IE			1-SNoW-IE			1-NB-IE		
	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_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_1	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_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 4.1: Results for *seminar announcements*

the F_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_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 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 examples) that were generated using our propositionalization framework for 1-SNoW-IE, and in exactly the same way. We discuss the experiments in more details below.

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 4.1) 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

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

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 the 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 outperforms the existing rule-based IE systems on all the four slots. To clarify, we note that the output representation of our system makes use of similar types of relational features as do the ILP-based systems, only that instead of a collection of conjunctive rules over these, it is represented as a linear function.

Tables 4.2 and 4.3 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 Section 3.3.3. 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:

- Tr : in the target region
- Before: one word is right before the other

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

Table 4.3: Some dominant features for slot `speaker` in seminar announcement data

- 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

Tables 4.2 and 4.3 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 announcements* task and 240,574 for the *job postings* 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.

	% 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 4.4: Filtering efficiency on *seminar announcements* data

Table 4.1 also exhibits the enhancement in performance due to the two state architecture of the SNoW-IE system. As a side note, the ability to use this architecture provides another indication of the flexibility of this approach and the advantage of learning with propositional means. Table 4.4 gives some insight of the examples fed 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, make 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. However, note 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 the naive Bayes 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_1	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_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 4.5: Results for *computer-related job postings*

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 4.5 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.

In terms of F_1, 2-SNoW-IE outperforms RAPIER in slots like `title`, `recruiter`, `state`, `city`, and `req_yrs_exp`, and has no statistical difference on others. As a result, the average F_1 is slightly, but statistically significantly, better (despite not using the semantic class sensor). 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. While they still achieve an equal or higher level of performance on some slots, they perform worse on others, and therefore the overall F_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 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 output only slots with high activation values, which indicate the confidence of the predictions.

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 does not allow exact comparisons. However, we believe the numbers reported here at least provide some feeling for the significant computational difference.

4.3.2 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.

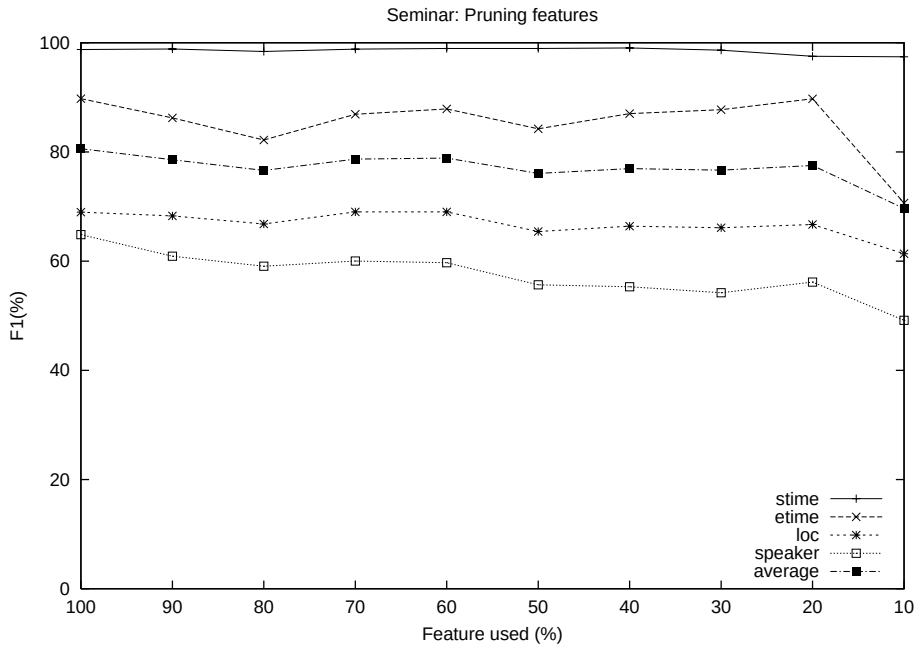

Figure 4.4: Eliminate infrequent features in *seminar announcements* experiments (10-fold cross validation)

Figure 4.4 presents the effect of pruning features. During 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 Figure 4.4 represents the percentage of the tail of this histogram that was eliminated. That is, all features that occur less than j times are eliminated, as long as the total number of features eliminated do not exceed $x\%$ of the features. Training continues for one more round over all the examples after features are eliminated (see details in (Carlson, Rosen, & Roth, 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.

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 extremely 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

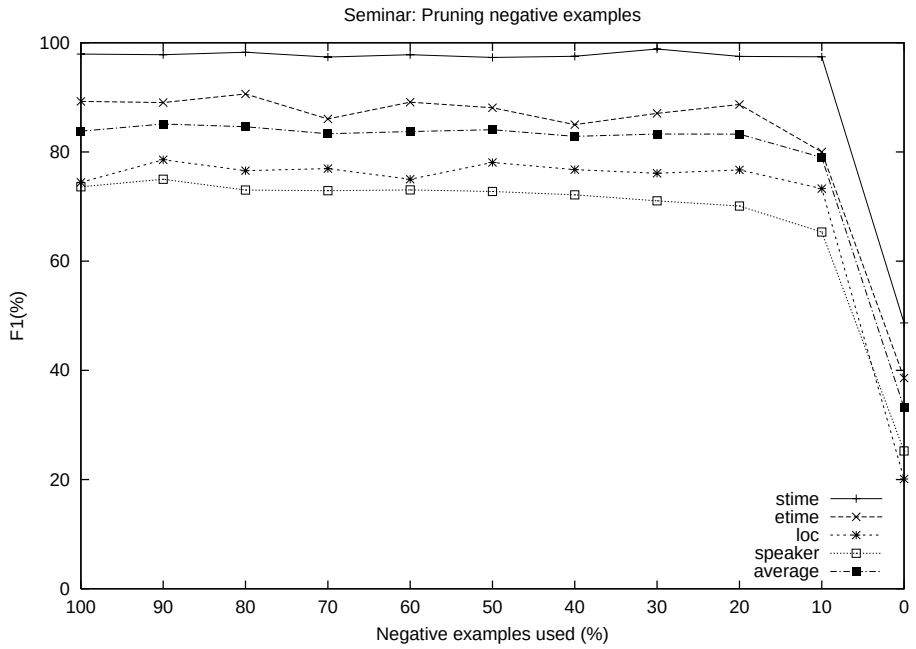

Figure 4.5: Eliminate negative examples randomly in *seminar announcements* experiments (10-fold cross validation)

function is able to make predictions that take 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 4.5 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 Section 4.2.3, 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_1 score 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.

4.4 Discussion

In this chapter, we have demonstrated how to apply the learning framework developed in Chapter 3 to build a basic information extraction system that identify certain phrases according to predefined templates. Specifically, we concentrated on the *seminar announcement* and *job posting* domains and have shown that our systems outperform existing ILP-based or -inspired methods, while being more efficient.

The information extraction problem tackled in this chapter is similar to other types of phrase labeling problem in natural language processing. Examples include *chunking* (Tjong Kim Sang & Buchholz, 2000), which recognizes the atomic phrases in sentences, and *named entity recognition* (Tjong Kim Sang & De Meulder, 2003), which identifies the types of phrases such as *person*, *location*, or *organization*. The IE problem in this chapter can be treated as a special entity recognition problem, and therefore the same technology can be applied on these NLP tasks.

In addition to recognizing entities, identifying the relations among entities is also an important problem for general information extraction. For example, whether the fields extracted in either the job posting or seminar announcement domain belong to the same table asks for the relation among these phrases. Although phrases from the same document are assumed to belong to the same table, and therefore the relation recognition problem is simplified in this chapter, this is usually not the case in general. In the next chapter, we extend the scope of information extraction to both entity

and relation recognition.

Chapter 5

Entity & Relation Recognition

This chapter develops a general framework for recognizing relations and entities in sentences, while taking mutual dependencies among them into account. For example, the *kill (KFJ, Oswald)* relation in: "`J. V. Oswald was murdered at JFK after his assassin, R. U. KFJ...`" depends on identifying Oswald and KFJ as *people*, JFK being identified as a *location*, and the *kill* relation between Oswald and KFJ; this, in turn, enforces that Oswald and KJF are *people*.

In our framework, classifiers that identify entities and relations among them are first learned from local information in the sentence; this information, along with constraints induced among entity types and relations, is used to perform global inference that accounts for the mutual dependencies among the entities and relations.

We develop two inference approaches based on Bayesian network and integer linear programming for this problem and evaluate them in the context of simultaneously learning named entities and relations. Our approaches allow us to efficiently incorporate domain and task specific constraints at decision time, resulting in significant improvements in the accuracy and the "human-like" quality of the inference.

5.1 Overview & Related Work

Recognizing and classifying entities and relations in text data is a key task in many NLP problems, such as information extraction (IE) (Califf & Mooney, 1999; Freitag, 2000; Roth & Yih, 2001), question answering (QA) (Voorhees, 2000) and story comprehension (Hirschman, Light, Breck, & Burger, 1999). In a typical IE application of constructing a job database, the system has to extract meaningful entities like *title* and *salary*, and it needs to determine whether the entities are associated with the same position. In a QA system, many questions ask for specific entities involved in some relations. For example, the question "Where was Poe born?" in the TREC-9 question answering track (Voorhees, 2000) asks for the *location* entity in which Poe was *born*. The question "Who killed Lee Harvey Oswald?" seeks a *person* entity that has the relation *kill* with the *person* Lee Harvey Oswald.

In all cases we know of, the tasks of identifying entities and relations are treated as separate problems. The common procedure is to first identify and classify entities using a named entity recognizer and only then determine the relations between the entities. However, this approach has several problems. First, if the named entity recognizer is not perfect, the errors it makes will propagate to the relation classifier. For example, if "Boston" is mislabeled as a person, it will never be classified as the location of Poe's birthplace. Second, relation information is sometimes crucial to resolving ambiguous named entity recognition. For instance, if the information that entity "JFK" is the victim of the assassination is given, the named entity recognizer is unlikely to misclassify it as a location (e.g., the JFK airport).

As opposed to the common approach, we want to consider these local problems together, and make a global prediction with respect to the constraints among the output. Informally, for the problem of recognizing the *kill (KFJ, Oswald)* relation in the sentence "`J. V. Oswald was murdered at JFK after his assassin, R. U. KFJ...`", it requires making several local decisions, such as identifying named entities in the sentence, in order to support the relation identification. For instance, it may be useful to identify that Oswald and KFJ are *people*,

and JFK is a *location*. This, in turn, may help to identify that the *kill* action is described in the sentence. At the same time, the relation *kill* constrains its arguments to be *people* (or at least, not to be *location*s) and helps to enforce that Oswald and KFJ are likely to be *people*, while JFK is not.

In our model, we first learn a collection of "local" predictors such as entity and relation identifiers. Their output is used to represent a conditional distribution for each entity and relation, given the observed data. At decision time, given a sentence, we produce a global decision that optimizes over the suggestions of the classifiers that are active in the sentence, known constraints among them and, potentially, domain or tasks specific constraints relevant to the current decision.

We test two inference approaches following this proposed framework. The first one creates a Bayesian network and incorporates the constraints in the conditional probability tables (CPTs). The second one relies on the integer linear programming optimization technique, which allows the representation of both hard and soft constraints.

The rest of this chapter is organized as follows. Section 5.2 defines the problem in a formal way. Sections 5.3 and 5.4 describe two inference approaches to this problem. Section 5.5 records the experiments we did and exhibits some promising results. Finally, Section 5.6 discusses some of the open problems and future work in this framework.

5.2 Global Inference of Entities/Relations

The problem at hand is that of producing a coherent labeling of entities and relations in a given sentence. Conceptually, the entities and relations can be viewed, taking into account the mutual dependencies, as the graph in Figure 5.1, where the nodes represent entities (e.g., phrases) and the links denote the binary relations between the entities. Each entity and relation has several properties. Some of the properties, such as words inside the entities and POS tags of words in the context of the sentence, are easy to acquire. However, other properties like the semantic types (i.e., class labels, such as "people", "locations") of phrases are difficult. Identifying the labels of entities and relations is treated here as a learning problem. In particular, we learn these target properties as

functions of all other properties of the sentence.

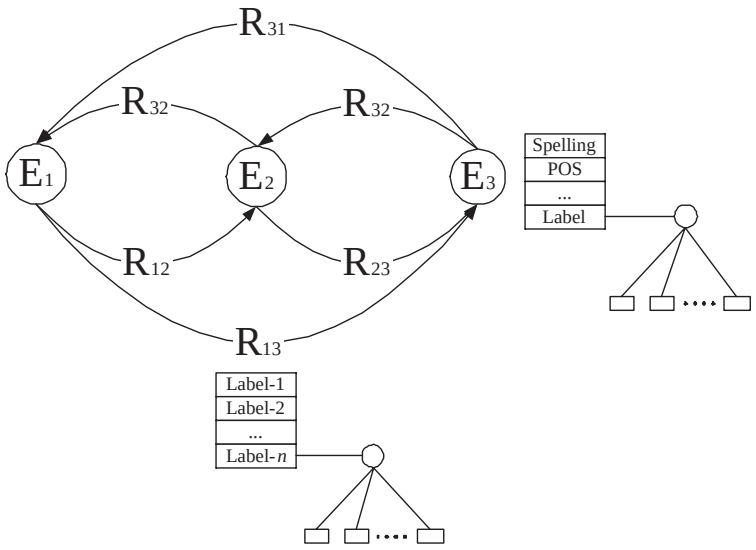

Figure 5.1: Conceptual view of entities and relations

To describe the problem in a formal way, we first define sentences and entities as follows.

Definition 5.2.1 (Sentence & Entity) *A sentence S is a linked list which consists of words w and entities $\mathcal{E}$. An entity can be a single word or a set of consecutive words with a predefined boundary. Entities in a sentence are labeled as $\mathcal{E} = \{E_1, E_2, \cdots, E_n\}$ according to their order, and they take values (i.e., labels) that range over a set of entity types $\mathcal{L}_\mathcal{E}$. The value assigned to $E_i \in \mathcal{E}$ is denoted $f_{E_i} \in \mathcal{L}_\mathcal{E}$.*

Notice that determining the entity boundaries is also a difficult problem – the *segmentation* (or *phrase detection*) problem (Abney, 1991; Punyakanok & Roth, 2001). Here we assume it is solved and given to us as input; thus we only concentrate on classification.

Dole 's wife , Elizabeth , is a native of Salisbury , N.C.
E1 **E2** **E3**

Figure 5.2: A sentence that has three entities

Example 5.2.1 *The sentence in Figure 5.2 has three entities: E_1 = "Dole", E_2 = "Elizabeth", and E_3 = "Salisbury, N.C."*

A relation is defined by the entities that are involved in it (its arguments). Note that we only discuss binary relations.

Definition 5.2.2 (Relation) *A (binary) relation $R_{ij} = (E_i, E_j)$ represents the relation between E_i and E_j, where E_i is the first argument and E_j is the second. In addition, R_{ij} can range over a set of entity types $\mathcal{L}_\mathcal{R}$. We use $\mathcal{R} = \{R_{ij}\}_{\{1 \leq i,j \leq n; i \neq j\}}$ as the set of binary relations on the entities $\mathcal{E}$ in a sentence. Two special functions $\mathcal{N}^1$ and $\mathcal{N}^2$ are used to indicate the argument entities of a relation R_{ij}. Specifically, $E_i = \mathcal{N}^1(R_{ij})$ and $E_j = \mathcal{N}^2(R_{ij})$.*

Example 5.2.2 *In the sentence given in Figure 5.2, there are six relations between the entities: R_{12} = ("Dole", "Elizabeth"), R_{21} = ("Elizabeth", "Dole"), R_{13} = ("Dole", "Salisbury, N.C."), R_{31} = ("Salisbury, N.C.", "Dole"), R_{23} = ("Elizabeth", "Salisbury, N.C."), and R_{32} = ("Salisbury, N.C.", "Elizabeth")*

We define the types (i.e., classes) of relations and entities as follows.

Definition 5.2.3 (Classes) *We denote the set of predefined entity classes and relation classes as $\mathcal{L}_\mathcal{E}$ and $\mathcal{L}_\mathcal{R}$ respectively. $\mathcal{L}_\mathcal{E}$ has one special element* other_ent*, which represents any unlisted entity class. Similarly, $\mathcal{L}_\mathcal{R}$ also has one special element* other_rel*, which means the involved entities are irrelevant or the relation class is undefined.*

When clear from the context, we use E_i and R_{ij} to refer to the entity and relation, as well as their types (class labels).

Example 5.2.3 *Suppose $\mathcal{L}_\mathcal{E}$ =* { other_ent, person, location } *and $\mathcal{L}_\mathcal{R}$ =* { other_rel, born_in, spouse_of }*. For the entities in Figure 5.2, E_1 and E_2 belong to* person *and E_3 belongs to* location*. In addition, relation R_{23} is* born_in*, R_{12} and R_{21} are* spouse_of*. Other relations are* other_rel*.*

The class label of a single entity or relation depends not only on its local properties, but also on properties of other entities and relations. The classification task is somewhat difficult since the predictions of entity labels and relation labels are mutually dependent. For instance, the class label of E_1 depends on the class label of R_{12} and the class label of R_{12} also depends on the class label of E_1 and E_2. While we can assume that all the data is annotated for training purposes, this cannot be assumed at evaluation time. We may presume that some local properties such as the words or POS tags are given, but none of the class labels for entities or relations is.

To simplify the complexity of the interaction within the graph but still preserve the characteristic of mutual dependency, we abstract this classification problem in the following probabilistic framework. First, the classifiers are trained independently and used to estimate the probabilities of assigning different labels given the observation (that is, the easily classified properties in it). Then, the output of the classifiers is used as a conditional distribution for each entity and relation, given the observation. This information, along with the constraints among the relations and entities, is used to make global inferences.

The class labels of entities and relations in a sentence must satisfy some constraints. For example, if E_1 (the first argument of R_{12}) is a *location*, then R_{12} cannot be *born_in* because the first argument of relation *born_in* has to be a *person*. We define constraints as follows.

Definition 5.2.4 (Constraint) *A constraint is a function that maps a relation label and an entity label to either 0 or 1 (dissatisfy or satisfy the constraint). Specifically,* $\mathcal{C}^1 : \mathcal{L}_\mathcal{R} \times \mathcal{L}_\mathcal{E} \rightarrow \{0, 1\}$ *constrains values of the first argument of a relation.* $\mathcal{C}^2$ *is defined similarly and constrains the second argument that a relation can take.*

The constraint function can be treated as a set, which consists of pairs of relation and entity labels that satisfy the constraint. For example, (*born_in*, *person*) is in $\mathcal{C}^1$ but not in $\mathcal{C}^2$ because the first entity of relation *born_in* has to be a *person* and the second entity can only be a *location* instead of a *person*.

We seek an inference algorithm that can produce a coherent labeling of entities and relations in

a given sentence. Furthermore, it follows, as best as possible the recommendation of the entity and relation classifiers, but also satisfies natural constraints that exist on whether specific entities can be the argument of specific relations, whether two relations can occur together at the same time, or any other information that might be available at the inference time (e.g., suppose it is known that entities A and B represent the same location; one may like to incorporate an additional constraint that prevents an inference of the type: "C lives in A; C does not live in B").

We note that a large number of problems can be modeled this way. Examples include problems such as chunking sentences (Punyakanok & Roth, 2001), coreference resolution and sequencing problems in computational biology. In fact, each of the components of our problem here, namely the separate task of recognizing named entities in sentences and the task of recognizing semantic relations between phrases, can be modeled this way. However, our goal is specifically to consider interacting problems at different levels, resulting in more complex constraints among them, and exhibit the power of our method.

5.3 Bayesian Network Inference

Each nontrivial property of the entities and relations, such as the class label, depends on a very large number of variables. In order to predict the most suitable coherent labels, we would like to make inferences on several variables. However, when modeling the interaction between the target properties, it is crucial to avoid accounting for dependencies among the huge set of variables on which these properties depend. This is because incorporating these dependencies into our inference is unnecessary and will make the inference intractable. Instead, we can abstract these dependencies in a way by learning the probability of each property conditioned upon the observation. The number of features on which this learning problem depends could be huge, and they can be of different granularity and based on previous learned predicates (e.g., POS), as abstracted into the triangles in Figure 5.1. Inference is then made based on the probabilities. This approach is similar to (Punyakanok & Roth, 2001; Lafferty, McCallum, & Pereira, 2001) only that there it is restricted

to sequential constraint structures.

The following subsections describe the details of these two stages. Section 5.3.1 explains the feature extraction method and learning algorithm we used. Section 5.3.2 introduces the idea of using a Bayesian network in search of the best global class labeling and the applied inference algorithm.

5.3.1 Learning Basic Classifiers

Although the labels of entities and relations from a sentence mutually depend on each other, two basic classifiers for entities and relations are first learned, in which a multi-class classifier for E(R) is learned as a function of all other "known" properties of the observation. The classifier for entities is a named entity classifier, in which the boundary of an entity is predefined. On the other hand, the relation classifier is given a pair of entities, which denote the two arguments of the target relation. Accurate predictions of these two classifiers seem to rely on complicated syntax analysis and semantics related information of the whole sentence. However, we derive weak classifiers by treating these two learning tasks as shallow text processing problems. This strategy has been successfully applied on several NLP tasks, such as information extraction (Califf & Mooney, 1999; Freitag, 2000; Roth & Yih, 2001) and chunking (i.e. shallow paring) (Muñoz, Punyakanok, Roth, & Zimak, 1999). It assumes that the class labels can be decided by local properties, such as the information provided by the words around or inside the target and their POS tags. Examples include the spelling of a word, part-of-speech, and semantic related attributes acquired from external resources such as WordNet (Miller, Beckwith, Fellbaum, Gross, & Miller, 1990).

However, raw features may be too simple and unable to represent sophisticated concepts. Researcher have argued that relation learning is needed for complicated learning tasks such as NLP problems. The traditional approach of relational learning is to apply ILP methods, which has several disadvantages. For instance, the learning algorithm may need modifying to adapt different problems and it is not trivial to transform classification results to probabilities, which is an impor-

tant issue in our framework. Alternatively, we handle this relational learning problem using the strategy suggested in Chapter 3: creating relational features first, and then applying propositional learning algorithms to learn on them.

The propositional learner we use is SNoW. Based on its learned activation values, we use it as a probabilistic classifier. More details can be found in Section 2.1.

5.3.2 Bayesian Inference Model

We construct a Bayesian network (see Section 2.2 for the introduction of Bayesian network) for each sentence that represents the constraints existing among R's and E's. Then, we use the classifiers from section 5.3.1 to compute the probabilities $P(E|observations)$ and $P(R|observations)$, and use the Bayesian network to compute the most probable global predictions of the class labels.

The structure of our Bayesian network, which represents the constraints is a two-layer graph. In particular, the variable E's and R's are the nodes in the network, where the E nodes are in one layer, and the R nodes are in the other. Since the label of a relation depends on the entity classes of its arguments, the links in the network connect the entity nodes and the relation nodes that have these entities as arguments. For instance, node R_{ij} has two incoming links from nodes E_i and E_j. As an illustration, Figure 5.3 shows a Bayesian network that consists of 3 entity nodes and 6 relation nodes. Property 5.3.1 describes some conditional probabilities $P(R_{ij}|E_i, E_j)$ that encodes the constraints in Definition 5.2.4.

Property 5.3.1 *The probability of the label of relation R_{ij} given the labels of its arguments E_i and E_j has the following properties.*

- $P(R_{ij} = \text{other\_rel}|E_i = e^1, E_j = e^2) = 1$, *if there exists no $r \in \mathcal{L}_\mathcal{R}$, such that $(r, e^1) \in \mathcal{C}^1$ and $(r, e^2) \in \mathcal{C}^2$.*

- $P(R_{ij} = r|E_i = e^1, E_j = e^2) = 0$, *if $(r, e^1) \notin \mathcal{C}^1$ or $(r, e^2) \notin \mathcal{C}^2$.*

Note that the conditional probabilities do not need to be specified manually. In fact, they can

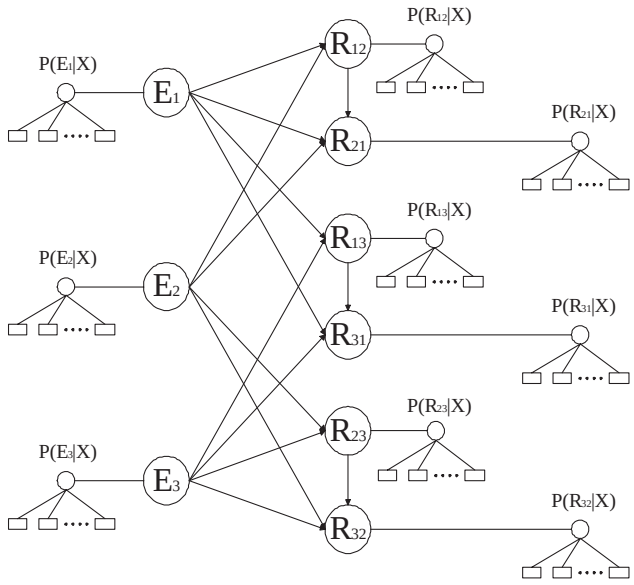

Figure 5.3: Bayesian network of 3 entity nodes and 6 relation nodes

be easily learned from an annotated training dataset. For those CPT entries that are not governed by Property 5.3.1, we decide the values by counting the frequencies in the training data.

Finding a most probable class assignment to the entities and relations is equivalent to finding the assignment of all the variables in the Bayesian network that maximizes the joint probability. In other words, the global prediction $e_1, e_2, ..., e_n, r_{12}, r_{21}, ..., r_{n(n-1)}$ satisfies the following equation.

$$(e_1, ..., e_n, r_{12}, ..., r_{n(n-1)}) = \underset{e_i, r_{jk}}{\operatorname{argmax}} Prob(E_1, ..., E_n, R_{12}, ..., R_{n(n-1)}).$$

However, this most-probable-explanation (MPE) inference problem is intractable if the network contains loops (undirected cycles) (Roth, 1996), which is exactly the case in our network. Therefore, we resort to the following approximation method instead.

Recently, researchers have achieved great success in solving the problem of decoding messages through a noisy channel with the help of Bayesian networks (Gallager, 1962; MacKay, 1999). The network structure used in their problem is similar to the network used here, namely a loopy bipartite DAG (directed acyclic graph). The inference algorithm they used is Pearl's belief propagation algorithm (Pearl, 1988), which outputs exact posteriors in linear time if the network is singly

connected (i.e. without loops) but does not guarantee to converge for loopy networks. However, researchers have empirically demonstrate that by iterating the belief propagation algorithm several times, the output values often converge to the right posteriors (Murphy, Weiss, & Jordan, 1999). Due to the existence of loops, we also apply belief propagation algorithm iteratively as our inference procedure.

5.4 Integer Linear Programming Inference

While it is fairly easy to use, the Bayesian network model may not be expressive enough since it allows no cycles. To fully model the problem, cycles may be needed. For example, the class labels of R_{12} and R_{21} actually depend on each other (e.g., if R_{12} is *born_in*, then R_{21} will not be *born_in* or *kill*). Similarly, the class labels of E_1 and E_2 can depend on the label of R_{12}. To fully represent the mutual dependencies, we would like to explore other probabilistic models that are more expressive than the Bayesian network.

The most direct way to formalize our inference problem is via the formalism of Markov Random Field (MRF) theory (Li, 2001). Rather than doing that, for computational reasons, we first use a fairly standard transformation of MRF to a discrete optimization problem (see (Kleinberg & Tardos, 1999) for details). Specifically, under weak assumptions we can view the inference problem as the following optimization problem, which aims to minimize the objective function that is the sum of the following two cost functions.

Assignment cost: the cost of deviating from the assignment of the variables $\mathcal{V}$ given by the classifiers. The specific cost function we use is defined as follows: Let l be the label assigned to variable $u \in \mathcal{V}$. If the marginal probability estimation is $p = P(f_u = l)$, then the assignment cost $c_u(l)$ is $-\log p$.

Constraint cost: the cost imposed by breaking constraints between neighboring nodes. The specific cost function we use is defined as follows: Consider two entity nodes E_i, E_j and its cor-

responding relation node R_{ij}; that is, $E_i = \mathcal{N}^1(R_{ij})$ and $E_j = \mathcal{N}^2(R_{ij})$. The constraint cost indicates whether the labels are consistent with the constraints. In particular, we use: $d^1(f_{E_i}, f_{R_{ij}})$ is 0 if $(f_{R_{ij}}, f_{E_i}) \in \mathcal{C}^1$; otherwise, $d^1(f_{E_i}, f_{R_{ij}})$ is ∞ [1]. Similarly, we use d^2 to force the consistency of the second argument of a relation.

Since we are seeking the most probable global assignment that satisfies the constraints, therefore, the overall cost function we optimize, for a global labeling f of all variables is:

$$C(f) = \sum_{u \in \mathcal{V}} c_u(f_u) + \sum_{R_{ij} \in \mathcal{R}} \left[d^1(f_{R_{ij}}, f_{E_i}) + d^2(f_{R_{ij}}, f_{E_j}) \right] \tag{5.1}$$

Unfortunately, it is not hard to see that the combinatorial problem (Equation 5.1) is computationally intractable even when placing assumptions on the cost function (Kleinberg & Tardos, 1999). The computational approach we adopt is to develop a *linear programming* (LP) formulation of the problem, and then solve the corresponding *integer linear programming* (ILP) problem. Our LP formulation is based on the method proposed by Chekuri, Khanna, Naor, and Zosin (2001). Since the objective function (Equation 5.1) is not a linear function in terms of the labels, we introduce new binary variables to represent different possible assignments to each original variable; we then represent the objective function as a linear function of these binary variables.

Let $x_{\{u,i\}}$ be an indicator variable, defined to be 1 if and only if variable u is labeled i and 0 otherwise, where $u \in \mathcal{E}, i \in \mathcal{L}_\mathcal{E}$ or $u \in \mathcal{R}, i \in \mathcal{L}_\mathcal{R}$. For example, $x_{\{E_1,2\}} = 1$ when the label of entity E_1 is 2; $x_{\{R_{23},3\}} = 0$ when the label of relation R_{23} is not 3. Let $x_{\{R_{ij},r,E_i,e_1\}}$ be an indicator variable representing whether relation R_{ij} is assigned label r and its first argument, E_i, is assigned label e_1. For instance, $x_{\{R_{12},1,E_1,2\}} = 1$ means the label of relation R_{12} is 1 *and* the label of its first argument, E_1, is 2. Similarly, $x_{\{R_{ij},r,E_j,e_2\}} = 1$ indicates that R_{ij} is assigned label r and its second argument, E_j, is assigned label e_2. With these definitions, the optimization problem can be represented as the following integer linear program.

[1] In practice, we use a very large number (e.g., 9^{15}).

$$\min \sum_{E\in\mathcal{E}} \sum_{e\in\mathcal{L}_\mathcal{E}} c_E(e)\cdot x_{\{E,e\}} + \sum_{R\in\mathcal{R}} \sum_{r\in\mathcal{L}_\mathcal{R}} c_R(r)\cdot x_{\{R,r\}}$$

$$+ \sum_{\substack{E_i,E_j\in\mathcal{E}\\ E_i\neq E_j}} \left[\sum_{r\in\mathcal{L}_\mathcal{R}} \sum_{e_1\in\mathcal{L}_\mathcal{E}} d^1(r,e_1)\cdot x_{\{R_{ij},r,E_i,e_1\}} + \sum_{r\in\mathcal{L}_\mathcal{R}} \sum_{e_2\in\mathcal{L}_\mathcal{E}} d^2(r,e_2)\cdot x_{\{R_{ij},r,E_j,e_2\}} \right]$$

subject to:

$$\sum_{e\in\mathcal{L}_\mathcal{E}} x_{\{E,e\}} = 1 \qquad \forall E\in\mathcal{E} \tag{5.2}$$

$$\sum_{r\in\mathcal{L}_\mathcal{R}} x_{\{R,r\}} = 1 \qquad \forall R\in\mathcal{R} \tag{5.3}$$

$$x_{\{E,e\}} = \sum_{r\in\mathcal{L}_\mathcal{R}} x_{\{R,r,E,e\}} \qquad \forall E\in\mathcal{E} \quad \text{and}$$
$$\forall R\in\{R : E=\mathcal{N}^1(R) \text{ or } E=\mathcal{N}^2(R)\} \tag{5.4}$$

$$x_{\{R,r\}} = \sum_{e\in\mathcal{L}_\mathcal{E}} x_{\{R,r,E,e\}} \qquad \forall R\in\mathcal{R} \quad \text{and}$$
$$\forall E=\mathcal{N}^1(R) \text{ or } E=\mathcal{N}^2(R) \tag{5.5}$$

$$x_{\{E,e\}} \in \{0,1\} \qquad \forall E\in\mathcal{E}, e\in\mathcal{L}_\mathcal{E} \tag{5.6}$$

$$x_{\{R,r\}} \in \{0,1\} \qquad \forall R\in\mathcal{R}, r\in\mathcal{L}_\mathcal{R} \tag{5.7}$$

$$x_{\{R,r,E,e\}} \in \{0,1\} \qquad \forall R\in\mathcal{R}, r\in\mathcal{L}_\mathcal{R},$$
$$E\in\mathcal{E}, e\in\mathcal{L}_\mathcal{E} \tag{5.8}$$

Equations (5.2) and (5.3) require that each entity or relation variable can only be assigned one label. Equations (5.4) and (5.5) assure that the assignment to each entity or relation variable is consistent with the assignment to its neighboring variables. Equations (5.6), (5.7), and (5.8) are the integral constraints on these binary variables.

There are several advantages of representing the problem in an LP formulation. First of all, linear (in)equalities are fairly general and are able to represent many types of constraints (e.g., the decision time constraint in the experiment in Section 5.5.2). More importantly, an ILP problem at

this scale can be solved very quickly using current numerical packages, such as Xpress-MP (2004) or CPLEX (2003). General strategies of solving an ILP problem are introduced in Section 2.3.2.

5.5 Experiments

The following subsections describe how we conduct experiments for the Bayesian inference model and the integer linear programming based inference procedure.

5.5.1 Experiments for the Bayesian Network Based Inference

Data Preparation

In order to build different datasets, we first collected sentences from TREC documents, which are mostly daily news such as Wall Street Journal, Associated Press, and San Jose Mercury News. Among the collected sentences, 245 sentences contain relation *kill* (i.e., two entities that have the *murder-victim* relation); 179 sentences contain relation *born_in* (i.e., a pair of entities where the second is the birthplace of the first). In addition to the above sentences, we also collected 502 sentences that contain no relations.[2]

Entities in these sentences are segmented by the simple rule: consecutive proper nouns and commas are combined and treated as an entity. Predefined entity class labels include *other_ent*, *person*, and *location*. Moreover, relations are defined by every pair of entities in a sentence, and the relation class labels defined are *other_rel*, *kill*, and *birthplace*.

Three datasets are constructed using the collected sentences. Dataset "kill" has all the 245 sentences of relation *kill*. Dataset "born_in" has all the 179 sentences of relation *born_in*. The third dataset "all" mixes all the sentences.

[2]The dataset is available at http://l2r.cs.uiuc.edu/∼cogcomp/Data/ER/

Tested Approaches

We compare three approaches in the experiments: *basic*, *omniscient*, and *BN*. The first approach, *basic*, tests our baseline – the performance of the basic classifiers. As described in Section 5.3.1, these classifiers are learned independently using local features and make predictions on entities and relations separately, without taking global interactions into account. The features extracted are described as follows. For the entity classifier, features from the words around each entity are: words, tags, conjunctions of words and tags, bigrams and trigrams of words and tags. Features from the entity itself include the number of words it contains, bigrams of words in it, and some attributes of the words inside such as the prefix and suffix. In addition, whether the entity has some strings that match the names of famous people and places is also used as a feature. For the relation classifier, features are extracted from words around and between the two entity arguments. The types of features include bigrams, trigrams, words, tags, and words related to "kill" and "birth" retrieved from WordNet.

The second approach, *omniscient*, is similar to *basic*. The only difference here is the labels of entities are revealed to the R classifier and so are the labels of relations to the E classifier. It is certainly impossible to know the true entity and relation labels in advance. However, this experiment may give us some idea about how much the performance of the entity classifier can be enhanced by knowing whether the target is involved in some relations, and also how much the relation classifier can be benefited from knowing the entity labels of its arguments. In addition, it also provides a comparison to see how well the Bayesian network inference model can improve the results.

The third approach, *BN*, tests the ability of making global inferences in our Bayesian network inference framework. We use the Bayes Net Toolbox for Matlab by Kevin Murphy [3] to implement the network and set the maximum number of the iteration of belief propagation algorithm as 20. Given the probabilities estimated by basic classifiers, the network reasons for the labels of the entities and relations globally in a sentence. Compared to the first two approaches, where some

[3]available at http://www.cs.ubc.ca/∼murphyk/Software/BNT/bnt.html

predictions may violate the constraints, the Bayesian network model incorporates the constraints between entities and relations, thus all the predictions it makes will be coherent.

All the experiments of these approaches are done in 5-fold validation. In other words, these datasets are randomly separated into 5 disjoint subsets, and experiments are done 5 times by iteratively using 4 of them as training data and the rest as testing.

Results

The experimental results in terms of recall, precision,and F_1 for datasets "kill", "born_in", and "all" are given in Table 5.1, Table 5.2, and Table 5.3 respectively. We discuss two interesting facts of the results as follows.

First, the Bayesian network approach tends to decrease recall in a small degree but increase precision significantly. This phenomenon is especially clear on the classification results of some relations. As a result, the F_1 value of the relation classification results is still enhanced to the extent that is near or even higher than the results of the *Omniscient* approach. This may be explained by the fact that if the label of a relation is predicted as positive (i.e. not *other_rel*), the types of its entity arguments must satisfy the constraints. This inference process reduces the number of false positive predictions, and thus enhances the precision.

Second, knowing the class labels of relations does not seem to help the entity classifier much. In all three datasets, the difference of *Basic* and *Omniscient* approaches is usually less than 3% in terms of F_1, which is not very significant given the size of our datasets. This phenomenon may be due to the fact that only a few entities in a sentence are involved in some relations. Therefore, it is unlikely that the entity classifier can use the relation information to correct its prediction.

Approach	person			location			kill		
	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
Basic	96.6	92.3	94.4	76.3	91.9	83.1	61.8	57.2	58.6
BN	89.0	96.1	92.4	78.8	86.3	82.1	49.8	85.4	62.2
Omniscient	96.4	92.6	94.5	75.4	90.2	81.9	67.7	63.6	64.8

Table 5.1: Results for dataset "kill"

Approach	person			location			born_in		
	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
Basic	85.5	90.7	87.8	89.5	93.2	91.1	81.4	63.4	70.9
BN	87.0	90.9	88.8	87.5	93.4	90.3	87.6	70.7	78.0
Omniscient	90.6	93.4	91.7	90.7	96.5	93.4	86.9	71.8	78.0

Table 5.2: Results for dataset "born_in"

Approach	person			location		
	Rec	Prec	F_1	Rec	Prec	F_1
Basic	92.1	87.0	89.4	83.2	81.1	82.0
BN	78.8	94.7	86.0	83.0	81.3	82.1
Omniscient	93.4	87.3	90.2	83.5	83.1	83.2
Approach	kill			born_in		
	Rec	Prec	F_1	Rec	Prec	F_1
Basic	43.8	78.6	55.0	69.0	72.9	70.5
BN	47.2	86.8	60.7	68.4	87.5	76.6
Omniscient	52.8	79.5	62.1	76.1	71.3	73.2

Table 5.3: Results for dataset "all"

5.5.2 Experiments for Integer Linear Programming Inference

We describe below two experiments on the problem of simultaneously recognizing entities and relations. In the first, we view the task as a knowledge acquisition task – we let the system read sentences and identify entities and relations among them. Given that this is a difficult task which may require quite often information beyond the sentence, we consider also a "forced decision" task, in which we simulate a question answering situation – we ask the system, say, "who killed whom" and evaluate it on identifying correctly the relation and its arguments, given that it is known that somewhere in this sentence this relation is active. In addition, this evaluation exhibits the ability of our approach to incorporate task specific constraints at decision time.

We annotated the named entities and relations in some sentences from the TREC documents. In order to effectively observe the interaction between relations and entities, we picked 1,437

sentences that have at least one active relation. Among those sentences, there are 5,336 entities, and 19,048 pairs of entities (binary relations). Entity labels include 1,685 *persons*, 1,968 *locations*, 978 *organizations* and 705 *other_ent*. Relation labels include 406 *located_in*, 394 *work_for*, 451 *orgBased_in*, 521 *live_in*, 268 *kill*, and 17,007 *other_rel*. Note that most pairs of entities have no active relations at all. Therefore, relation *other_rel* significantly outnumbers others. Examples of each relation label and the constraints between a relation variable and its two entity arguments are shown in Table 5.4.

Relation	Entity1	Entity2	Example
located_in	loc	loc	(New York, US)
work_for	per	org	(Bill Gates, Microsoft)
orgBased_in	org	loc	(HP, Palo Alto)
live_in	per	loc	(Bush, US)
kill	per	per	(Oswald, JFK)

Table 5.4: Relations of interest and the corresponding constraints

In order to focus on the evaluation of our inference procedure, we assume the problem of *segmentation* (or *phrase detection*) (Abney, 1991; Punyakanok & Roth, 2001) is solved, and the entity boundaries are given to us as input; thus we only concentrate on their classifications.

We evaluate our LP based inference procedure by observing its effect in different approaches of combining the entity and relation classifiers. The first approach is to train entity and relation classifiers *separately*. In particular, the relation classifier does not know the labels of its entity arguments, and the entity classifier does not know the labels of relations in the sentence, either. For the entity classifier, one set of features are extracted from words within a size 4 window around the target phrase. They are: (1) words, part-of-speech tags, and conjunctions of them; (2) bigrams and trigrams of the mixture of words and tags. In addition, some other features are extracted from the target phrase, which are listed in Table 5.5.

For the relation classifier, there are three sets of features:

1. features similar to those used in the entity classification are extracted from the two argument

[4]We collected names of famous places, people and popular titles from other data sources in advance.

symbol	explanation
icap	the first character of a word is capitalized
acap	all characters of a word are capitalized
incap	some characters of a word are capitalized
suffix	the suffix of a word is “ing”, “ment”, etc.
bigram	bigram of words in the target phrase
len	number of words in the target phrase
place[4]	the phrase is/has a known place’s name
prof[4]	the phrase is/has a professional title (e.g., Lt.)
name[4]	the phrase is/has a known person’s name

Table 5.5: Some features extracted from the target phrase

Pattern	Example
arg_1 , arg_2	San Jose, CA
arg_1 , $\cdots$ a $\cdots$ arg_2 *prof*	John Smith, a Starbucks manager $\cdots$
in/at arg_1 in/at/, arg_2	Officials in Perugia in Umbria province said $\cdots$
arg_2 *prof* arg_1	CNN reporter David McKinley $\cdots$
arg_1 $\cdots$ native of $\cdots$ arg_2	Elizabeth Dole is a native of Salisbury, N.C.
arg_1 $\cdots$ based in/at arg_2	$\cdots$ a manager for K mart based in Troy, Mich. said $\cdots$

Table 5.6: Some patterns used in relation classification

entities of the relation;

2. conjunctions of the features from the two arguments;

3. some patterns extracted from the sentence or between the two arguments.

Some features in category 3 are “the number of words between arg_1 and arg_2 ”, “whether arg_1 and arg_2 are the same word”, or “arg_1 is the beginning of the sentence and has words that consist of all capitalized characters”, where arg_1 and arg_2 represent the first and second argument entities respectively. Table 5.6 presents some patterns we use.

In addition to the *separate* approach, we also test several pipeline models, including $E \rightarrow R$, $R \rightarrow E$ and $E \leftrightarrow R$. $E \rightarrow R$ first trains an entity classifier, and its predictions on the two entity arguments of a relation are used conjunctively as additional features in learning the relation classifier. Similarly, $R \rightarrow E$ first trains the relation classifier, and its output is used as additional features in the entity classifier. The $E \leftrightarrow R$ model is the combination of the above two. It takes the entity classifier in the $R \rightarrow E$ model and the relation classifier in the $E \rightarrow R$ as its final classifiers.

Although the true labels of entities and relations are known during training, only the *predicted* labels are available in testing. Learning on the predictions of the previous stage classifier presumably makes the current stage classifier more tolerant to the mistakes. In fact, we also observe this phenomenon empirically. For example, when the relation classifier is trained using the true entity labels, the performance is much worse than using the predicted entity labels.

The last approach, *omniscient*, tests the conceptual upper bound of this entity/relation classification problem. It also trains the two classifiers separately. However, it assumes that the entity classifier knows the *correct* relation labels, and similarly the relation classifier knows the *right* entity labels as well. This additional information is then used as features in training and testing. Note that this assumption is totally unrealistic. Nevertheless, it may give us a hint on how accurate the classifiers with global inference can achieve. Finally, we apply the LP based inference procedure to the above 5 models, and observe how it improves the performance.

5.5.3 Results

Tables 5.7 and 5.8 show the performance of each approach in F_1. The results show that the inference procedure consistently improves the performance of the 5 models, both in entities and relations. One interesting observation is that the *omniscient* classifiers, which know the correct

Approach	person			location			organization		
	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
Separate	89.5	89.8	89.4	87.0	91.5	89.0	67.6	91.3	77.0
Separate w/ Inf	90.5	90.6	90.4	88.6	91.8	90.1	71.0	91.2	79.4
E $\rightarrow$ R	89.5	89.8	89.4	87.0	91.5	89.0	67.6	91.3	77.0
E $\rightarrow$ R w/ Inf	89.7	90.1	89.7	87.0	91.7	89.1	69.0	91.2	78.0
R $\rightarrow$ E	89.1	88.7	88.6	88.1	89.8	88.9	71.4	89.3	78.7
R $\rightarrow$ E w/ Inf	88.6	88.6	88.3	88.2	89.4	88.7	72.1	88.5	79.0
E $\leftrightarrow$ R	89.1	88.7	88.6	88.1	89.8	88.9	71.4	89.3	78.7
E $\leftrightarrow$ R w/ Inf	89.5	89.1	89.0	88.7	89.7	89.1	72.0	89.5	79.2
Omniscient	94.9	93.7	94.2	92.4	96.6	94.4	88.1	93.5	90.7
Omniscient w/ Inf	96.1	94.2	95.1	94.0	97.0	95.4	88.7	94.9	91.7

Table 5.7: Results of long entity classification

Approach	located_in			work_for			orgBased_in		
	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
Separate	53.0	43.3	45.2	41.9	55.1	46.3	35.6	85.4	50.0
Separate w/ Inf	51.6	56.3	50.5	40.1	74.1	51.2	35.7	90.8	50.8
E $\rightarrow$ R	56.4	52.5	50.7	44.4	60.8	51.2	42.1	77.8	54.3
E $\rightarrow$ R w/ Inf	55.7	53.2	50.9	42.9	72.1	53.5	42.3	78.0	54.5
R $\rightarrow$ E	53.0	43.3	45.2	41.9	55.1	46.3	35.6	85.4	50.0
R $\rightarrow$ E w/ Inf	53.0	49.8	49.1	41.6	67.5	50.4	36.6	87.1	51.2
E $\leftrightarrow$ R	56.4	52.5	50.7	44.4	60.8	51.2	42.1	77.8	54.3
E $\leftrightarrow$ R w/ Inf	55.7	53.9	51.3	42.3	72.0	53.1	41.6	79.8	54.3
Omniscient	62.9	59.5	57.5	50.3	69.4	58.2	50.3	77.9	60.9
Omniscient w/ Inf	62.9	61.9	59.1	50.3	79.2	61.4	50.9	81.7	62.5

Approach	live_in			kill		
	Rec	Prec	F_1	Rec	Prec	F_1
Separate	39.7	61.7	48.0	81.5	75.3	77.6
Separate w/ Inf	41.7	68.2	51.4	80.8	82.7	81.4
E $\rightarrow$ R	50.0	58.9	53.5	81.5	73.0	76.5
E $\rightarrow$ R w/ Inf	50.0	57.7	53.0	80.6	77.2	78.3
R $\rightarrow$ E	39.7	61.7	48.0	81.5	75.3	77.6
R $\rightarrow$ E w/ Inf	40.6	64.1	49.4	81.5	79.7	80.1
E $\leftrightarrow$ R	50.0	58.9	53.5	81.5	73.0	76.5
E $\leftrightarrow$ R w/ Inf	49.0	59.1	53.0	81.5	77.5	79.0
Omniscient	56.1	61.7	58.2	81.4	76.4	77.9
Omniscient w/ Inf	57.3	63.9	59.9	81.4	79.9	79.9

Table 5.8: Results of relation classification

entity or relation labels, can still be improved by the inference procedure. This demonstrates the effectiveness of incorporating constraints, even when the learning algorithm may be able to learn them from the data.

One of the more significant results in our experiments, we believe, is the improvement in the *quality* of the decisions. As mentioned in Section 5.1, incorporating constraints helps to avoid inconsistency in classification. It is interesting to investigate how often such mistakes happen without global inference, and see how effectively the global inference enhances this.

For this purpose, we define the *quality* of the decision as follows. For an active relation of which the label is classified correctly, if both its argument entities are also predicted correctly, we count it as a *coherent* prediction. *Quality* is then the number of *coherent* predictions divided

by the sum of *coherent* and *incoherent* predictions. When the inference procedure is not applied, 5% to 25% of the predictions are incoherent. Therefore, the quality is not always good. On the other hand, our global inference procedure takes the natural constraints into account, so it never generates incoherent predictions. If the relation classifier has the correct entity labels as features, a good learner should learn the constraints as well. As a result, the quality of *omniscient* is almost as good as *omniscient with inference*.

Another experiment we did is the *forced decision* test, which boosts the F_1 score of the "kill" relation to 86.2%. Here we consider only sentences in which the "kill" relation is active. We force the system to determine which of the possible relations in a sentence (i.e., which pair of entities) has this relation by adding a new linear inequality. This is a realistic situation (e.g., in the context of question answering) in that it adds an external constraint, not present at the time of learning the classifiers and it evaluates the ability of our inference algorithm to cope with it. The results exhibit that our expectations are correct.

5.6 Discussion

We presented two different inference approaches based on Bayesian network and integer linear programming for global inference where decisions depend on the outcomes of several different but mutually dependent classifiers. Although the Bayesian network approach has shown its feasibility in our preliminary experiments, the DAG structure may not provide enough expressivity for problems that involve more components and constraints. On the other hand, the integer linear programming formalism is able to incorporate a fairly general constraint structure that may deviate from the sequential nature typically studied, and can find the optimal solution efficiently. Interested readers may refer to Appendix A for more detailed examples of using integer linear programming for global inference, as well as transforming logic rules to linear constraints.

Contrary to general search schemes (e.g., beam search), which do not guarantee optimality, the linear programming approach provides an efficient way to finding the optimal solution. The

key advantage of the linear programming formulation is its generality and flexibility; in particular, it supports the ability to incorporate classifiers learned in other contexts, "hints" supplied and decision time constraints, and reason with all these for the best global prediction. In sharp contrast with the typically used pipeline framework, our global inference approaches do not blindly trust the results of some classifiers, and therefore are able to overcome mistakes made by classifiers with the help of constraints.

Our experiments have demonstrated these advantages by considering the interaction between entity and relation classifiers. In fact, more classifiers can be added and used within the same integer linear programming framework. For example, if coreference resolution is available, it is possible to incorporate it in the form of constraints that force the labels of the co-referred entities to be the same (but, of course, allowing the global solution to reject the suggestion of these classifiers). Consequently, this may enhance the performance of entity/relation recognition and, at the same time, correct possible coreference resolution errors. Another example is to use chunking information for better relation identification; suppose, for example, that we have available chunking information that identifies Subj+Verb and Verb+Object phrases. Given a sentence that has the verb "murder", we may conclude that the subject and object of this verb are in a "kill" relation. Since the chunking information is used in the global inference procedure, this information will contribute to enhancing its performance and robustness, relying on having more constraints and overcoming possible mistakes by some of the classifiers. Moreover, in an interactive environment where a user can supply new constraints (e.g., a question answering situation) this framework is able to make use of the new information and enhance the performance at decision time, without retraining the classifiers.

In addition to the aforementioned research directions, we would also like to explore the application of our framework in a boot-strapping manner. The main difficulty in applying learning on NLP problems is not lack of text corpus, but lack of *labeled* data. Boot-strapping, applying the classifiers to autonomously annotate the data and using the new data to train and improve existing classifiers, is a promising approach. Since the precision of our framework is pretty high, it seems

possible to use the global inference to annotate new data. Based on this property, we can derive an EM-like approach for labeling and inferring the types of entities and relations simultaneously. The basic idea is to use the global inference output as a mean to annotate entities and relations. The new annotated data can then be used to train classifiers, and the whole process is repeated again.

As we have shown, our formulation supports not only improved accuracy, but also improves the ‘human-like” quality of the decisions. We believe that it has the potential to be a powerful way for supporting natural language inferences.

Chapter 6

Semantic Role Labeling

Semantic role labeling is a task that identifies the arguments of each verb in a sentence. It can be treated as an extended version of relation recognition, where the verb with its correct sense represents the relation, and the arguments can be viewed as the entities that have the relation.

In this chapter, we present a general framework for semantic role labeling. It combines a machine learning technique with an integer linear programming based inference procedure, which incorporates linguistic and structural constraints into the decision process. The system achieves very competitive results, and is one of the best systems in the CoNLL-2004 shared task on semantic role labeling. In addition, we study experimentally the necessity of syntactic parsing for semantic role labeling. Inspired by the conclusion, we develop a state-of-the-art semantic role labeling system by combining several systems based on different syntactic parsers.

6.1 Overview and Related Work

Semantic parsing of sentences is believed to be an important task toward natural language understanding, and has immediate applications in tasks such as information extraction and question answering. We study *semantic role labeling* (SRL) in which for each verb in a sentence, the goal is to identify all constituents that fill a semantic role, and to determine their roles, such as Agent, Patient or Instrument, and their adjuncts, such as Locative, Temporal or Manner.

The PropBank project (Kingsbury & Palmer, 2002), which provides a large human-annotated

corpus of semantic verb-argument relations, has enabled researchers to apply machine learning techniques to improve SRL systems (Gildea & Palmer, 2002; Chen & Rambow, 2003; Gildea & Hockenmaier, 2003; Pradhan, Hacioglu, Ward, Martin, & Jurafsky, 2003; Surdeanu, Harabagiu, Williams, & Aarseth, 2003; Pradhan, Ward, Hacioglu, Martin, & Jurafsky, 2004; Xue & Palmer, 2004). However, most systems rely heavily on the full syntactic parse trees. Therefore, the overall performance of the system is largely determined by the quality of the automatic syntactic parsers of which state of the art (Collins, 1999; Charniak, 2001) is still far from perfect.

Alternatively *shallow* syntactic parsers (i.e., chunkers and clausers), although not providing as much information as a full syntactic parser, have been shown to be more robust in their specific task (Li & Roth, 2001). This raises the very natural and interesting question of quantifying the necessity of the full parse information to semantic parsing and whether it is possible to use only shallow syntactic information to build an outstanding SRL system.

Although PropBank is built by adding semantic annotation to the constituents on syntactic parse trees in Penn Treebank, it is not clear how important syntactic parsing is for building an SRL system. To the best of our knowledge, this problem was first addressed by Gildea and Palmer (2002). In their attempt of using limited syntactic information, the parser was *very shallow* – clauses were not available and only chunks were used. Moreover, the pruning stage in (Gildea & Palmer, 2002) was too strict since only chunks are considered as argument candidates, meaning that over 60% of the arguments were not treated as candidates. As a result, the overall recall in their approach was very low. As we will demonstrate later, high recall of the pruning stage is in fact essential to a quality SRL system.

Using only the shallow parse information in an SRL system has largely been ignored until the recent CoNLL-04 shared task competition (Carreras & Màrquez, 2004a). In this competition, participants were restricted to only shallow parse information for their SRL systems. As a result, it became clear that the performance of the best shallow parse based system (Hacioglu, Pradhan, Ward, Martin, & Jurafsky, 2004) is only 10% in F_1 below the best system that uses full parse information (Pradhan, Ward, Hacioglu, Martin, & Jurafsky, 2004). In addition, there has not been

a true quantitative comparison with shallow parsing. First, the CoNLL-04 shared task used only a subset of the data for training. Furthermore, its evaluation treats the continued and referential tags differently, which makes the performance metric stricter and the results worse. Second, an SRL system is usually complicated and consists of several stages. It is still unknown how much and where precisely the syntactic information helps the most.

The goal of this chapter is twofold. First, we make a fair comparison between SRL systems which use full parse trees and those exclusively using shallow syntactic information. Our shallow parsing based SRL system achieves very competitive results and has been evaluated in the CoNLL-04 shared task competition. This comparison brings forward a better analysis on the necessity of full parsing in the SRL task. Second, to relieve the dependency of the SRL system on the quality of automatic parsers, we improve semantic role labeling significantly by combining several SRL systems based on different state-of-art full parsers.

To make our conclusions applicable to general SRL systems, we adhere to a widely used two step system architecture. In the first step, the system is trained to identify argument candidates for a given verb predicate. In the second step, the system classifies the argument candidates into their types. In addition, it is also common to use a simple procedure to prune obvious non-candidates before the first step, and to use post-processing inference to fix inconsistent predictions after the second step. We also employ these two additional steps.

In our comparison between the systems using shallow and full syntactic information, we found the most interesting result is that while each step of the system using shallow information exhibits very good performance, the overall performance is significantly inferior to the system that uses full information. This necessity of full parse information is especially noticeable at the pruning stage. In addition, we produce a state-of-the-art SRL system by combining of different SRL systems based on two (potentially noisy) automatic full parsers (Collins, 1999; Charniak, 2001).

The rest of this chapter is organized as follows. Section 6.2 introduces the task of semantic role labeling in more detail. Section 6.3 describes the 4-stage architecture of our SRL system, which includes pruning, argument identification, argument classification, and inference. In addition, the

features used are also explained. Section 6.4 presents the evaluation results of our system in the CoNLL-04 shared task competition. Section 6.5 explains why and where the full parsing information contributes to SRL by conducting a series of carefully designed experiments. Inspired by the result, we propose an approach that combines different SRL systems based on joint inference in Section 6.6. Finally, Section 6.7 concludes this chapter.

6.2 Semantic Role Labeling (SRL) Task

The goal of the semantic-role labeling task is to discover the verb-argument structure for a given input sentence. For example, given a sentence " I *left* my pearls to my daughter-in-law in my will", the goal is to identify different arguments of the verb *left* which yields the output:

$[_{\text{A0}}$ I$]$ $[_{\text{V}}$ *left* $]$ $[_{\text{A1}}$ my pearls$]$ $[_{\text{A2}}$ to my daughter-in-law$]$ $[_{\text{AM-LOC}}$ in my will$]$.

Here A0 represents the *leaver*, A1 represents the *thing left*, A2 represents the *benefactor*, AM-LOC is an adjunct indicating the location of the action, and V determines the verb. In addition, each argument can be mapped to a constituent in its corresponding syntactic full parse tree.

Following the definition of the PropBank and CoNLL-2004 shared task, there are six different types of arguments labeled as A0-A5 and AA. These labels have different semantics for each verb as specified in the PropBank Frame files. In addition, there are also 13 types of adjuncts labeled as AM-*adj* where *adj* specifies the adjunct type. In some cases, an argument may span over different parts of a sentence, the label C-*arg* is used to specify the continuity of the arguments, as shown in the example below.

$[_{\text{A1}}$ The pearls$]$, $[_{\text{A0}}$ I$]$ $[_{\text{V}}$ *said*$]$, $[_{\text{C-A1}}$ were left to my daughter-in-law$]$.

In some other cases, an argument might be a relative pronoun that in fact refers to the actual agent outside the clause. In this case, the actual agent is labeled as the appropriate argument type, *arg*, while the relative pronoun is instead labeled as R-*arg*. For example,

$[_{\text{A1}}$ The pearls$]$ $[_{\text{R-A1}}$ which$]$ $[_{\text{A0}}$ I$]$ $[_{\text{V}}$ *left*$]$, $[_{\text{A2}}$ to my daughter-in-law$]$ are fake.

The distribution of these argument labels is fairly unbalanced. In the official release of PropBank I, core arguments (A0–A5 and AA) occupy 71.26%, where the largest parts are A0 (25.39%) and A1 (35.19%). The rest portion is mostly the adjunct arguments (24.90%). The continued (C-*arg*) and referential (R-*arg*) arguments are relatively fewer, occupying 1.22% and 2.63% respectively. For more definitions of PropBank and the semantic role labeling task, readers can refer to (Kingsbury & Palmer, 2002) and (Carreras & Màrquez, 2004a).

6.3 SRL System Architecture

Our SRL system consists of four stages: *pruning*, *argument identification*, *argument classification*, and *inference*. In particular, the goal of pruning and argument identification is to identify argument candidates for a given verb predicate. The system only classifies the argument candidates into their types in the stage of argument classification. Linguistic and structural constraints are incorporated in the inference stage to resolve inconsistent global predictions. This section describes how we build these four stages, including the features used in training the classifiers.

6.3.1 Pruning

When the full parse tree of a sentence is available, only the constituents in the parse tree are considered as argument candidates. In addition, our system exploits the heuristic rules introduced by Xue and Palmer (2004) to filter out simple constituents that are very unlikely to be arguments. The heuristic is a recursive process starting from the verb of which arguments to be identified. It first returns the siblings of the verb as candidates; then it moves to the parent of the verb, and collects the siblings again. The process goes on until it reaches the root. In addition, if a constituent is a `PP` (propositional phrase), its children are also collected.

6.3.2 Argument Identification

The argument identification stage utilizes binary classification to identify whether a candidate is an argument or not. When full parsing is available, we train and apply the binary classifiers on the constituents supplied by the pruning stage. When only shallow parsing is available, the system does not have the pruning stage, and also does not have constituents to begin with. Therefore, conceptually the system has to consider all possible subsequences (i.e., consecutive words) in a sentence as potential argument candidates. We avoid this by using a learning scheme by first training two classifiers, one to predict the beginnings of possible arguments, and the other the ends. The predictions are combined to form argument candidates that do not violate the following constraints.

1. Arguments cannot cover the predicate (i.e., the active verb).
2. Arguments cannot overlap with the clauses (they can be embedded in one another).
3. If a predicate is outside a clause, its arguments cannot be embedded in that clause.

The features used in the full parsing and shallow parsing settings are described as follows.

Features when full parsing is available

Most of the features used in our system are standard features which include

- **Predicate and POS tag of predicate** features indicate the lemma of the predicate verb and its POS tag.
- **Voice** feature indicates passive/active voice of the predicate.
- **Phrase type** feature provides the phrase type of the constituent.
- **Head word and POS tag of the head word** feature provides the head word and its POS tag of the constituent. We use rules introduced by Collins (1999) to extract this feature.

- **Position** feature describes if the constituent is before or after the predicate relative to the position in the sentence.
- **Path** records the traversal path in the parse tree from the predicate to the constituent.
- **Subcategorization** feature describes the phrase structure around the predicate's parent. It records the immediate structure in the parse tree that expands to its parent.

We also use the following additional features.

- **Verb class** feature is the class of the predicate described in PropBank Frames.
- **Lengths** of the target constituent, in the numbers of words and chunks separately.
- **Chunk** tells if the target argument is, embeds, overlaps, or is embedded in a chunk with its type.
- **Chunk pattern** encodes the sequence of chunks from the current words to the predicate.
- **Chunk pattern length** feature counts the number of chunks in the argument.
- **Clause relative position** feature is the position of the target word relative to the predicate in the pseudo-parse tree constructed only from clause constituents. There are four configurations—target constituent and predicate share the same parent, target constituent parent is an ancestor of predicate, predicate parent is an ancestor of target word, or otherwise.
- **Clause coverage** describes how much of the local clause (from the predicate) is covered by the target argument.
- **NEG** feature is active if the target verb chunk has `not` or `n't`.
- **MOD** feature is active when there is a modal verb in the verb chunk. The rules of the **NEG** and **MOD** features are used in a baseline SRL system developed by Erik Tjong Kim Sang (Carreras & Màrquez, 2004a).

Features when only shallow parsing is available

Features used are similar to those used by the system with full parsing except those that need full parse trees to generate. For these types of features, we either try to mimic the features with some heuristics rules or discard them. The details of these features are as follows.

- **Phrase type** uses a simple heuristics to identify only `VP`, `PP`, and `NP`.
- **Head word and POS tag of the head word** are the rightmost word for `NP`, and the leftmost word for `VP` and `PP`.
- **Shallow-Path** records the traversal path in the pseudo-parse tree constructed only from the clause structure and chunks.
- **Shallow-Subcategorization** feature describes the chunk and clause structure around the predicate's parent in the pseudo-parse tree.
- **Syntactic frame** features are discarded.

6.3.3 Argument Classification

This stage assigns the final argument labels to the argument candidates supplied from the previous stage. A multi-class classifier is trained to classify the types of the argument candidates. In addition, to reduce the excessive candidates mistakenly output by the previous stage, the classifier can also classify the argument as *null* (meaning "not an argument") to discard the argument.

The features used here are the same as those used in the argument identification stage. However, when full parsing are available, an additional feature introduced by Xue and Palmer (2004) is used.

- **Syntactic frame** describes the sequential pattern of the noun phrases and the predicate in the sentence.

6.3.4 Inference

The purpose of this stage is to incorporate some prior linguistic and structural knowledge, such as "arguments do not overlap" or "each verb takes at most one argument of each type." This knowledge is used to resolve any inconsistencies of argument classification in order to generate final legitimate predictions. We design an inference procedure based on integer linear programming (ILP). It takes as input the confidence scores over each type of the arguments supplied by the argument classifier. The output is the optimal solution that maximizes the linear sum of the confidence scores (e.g., the conditional probabilities estimated by the argument classifier), subject to the constraints that encode the domain knowledge.

In this subsection we first introduce the constraints and the inference problem in the semantic role labeling task. We then demonstrate how we apply integer linear programming (ILP) to generate the global label assignment.

Constraints over Argument Labeling

Formally, the argument classifier attempts to assign labels to a set of arguments, $S^{1:M}$, indexed from 1 to M. Each argument S^i can take any label from a set of argument labels, $\mathcal{P}$, and the indexed set of arguments can take a set of labels, $c^{1:M} \in \mathcal{P}^M$. If we assume that the classifier returns a score, $\text{score}(S^i = c^i)$, corresponding to the likelihood of seeing label c^i for argument S^i, then, given a sentence, the unaltered inference task is solved by maximizing the overall score of the arguments,

$$\hat{c}^{1:M} = \underset{c^{1:M} \in \mathcal{P}^M}{\text{argmax}} \, \text{score}(S^{1:M} = c^{1:M}) = \underset{c^{1:M} \in \mathcal{P}^M}{\text{argmax}} \sum_{i=1}^{M} \text{score}(S^i = c^i). \tag{6.1}$$

In the presence of global constraints derived from linguistic information and structural considerations, our system seeks for a *legitimate* labeling that maximizes the score. Specifically, it can be viewed as the solution space is limited through the use of a filter function, $\mathcal{F}$, that eliminates many argument labelings from consideration. It is interesting to contrast this with previous work

that filters individual phrases (Carreras & Màrquez, 2004b). Here, we are concerned with global constraints as well as constraints on the arguments. Therefore, the final labeling becomes

$$\hat{c}^{1:M} = \underset{c^{1:M} \in \mathcal{F}(\mathcal{P}^M)}{\operatorname{argmax}} \sum_{i=1}^{M} \operatorname{score}(S^i = c^i) \tag{6.2}$$

The filter function used considers the following constraints:

1. Arguments cannot cover the predicate except those that contain only the verb or the verb and the following word.
2. Arguments cannot overlap with the clauses (they can be embedded in one another).
3. If a predicate is outside a clause, its arguments cannot be embedded in that clause.
4. No overlapping or embedding arguments.
5. No duplicate argument classes for core arguments, such as A0–A5 and AA.
6. Exactly one V argument per proposition (i.e., a sentence given the active verb).
7. If there is a C-V argument, then there should be a sequence of consecutive V, A1, and C-V pattern. For example, when *split* is the verb in "split it up", the A1 argument is "it" and C-V argument is "up".
8. If there is an R-*arg* argument, then there has to be an *arg* argument. That is, if an argument is a reference to some other argument *arg*, then this referenced argument must exist in the sentence.
9. If there is a C-*arg* argument, then there has to be an *arg* argument; in addition, the C-*arg* argument must occur after *arg*. This is stricter than the previous rule because the order of appearance also needs to be considered.
10. Given the predicate, some argument classes are illegal (e.g. predicate 'stalk' can take only A0 or A1). This linguistic information can be found in *PropBank Frames*.

We reformulate the constraints as linear (in)equalities by introducing indicator variables. The optimization problem (Eq. 6.2) is solved using ILP.

Using Integer Linear Programming

As discussed previously, a collection of potential arguments is not necessarily a valid semantic labeling since it must satisfy all of the constraints. In this context, inference is the process of finding the *best* (according to Equation 6.1) valid semantic labels that satisfy all of the specified constraints. We take a similar approach that has been previously used for entity/relation recognition (See Chapter 5), and model this inference procedure as solving an ILP problem.

An *integer linear program* is basically the same as a *linear program*. The cost function and the (in)equality constraints are all linear in terms of the variables. The only difference in an integer linear program is the variables can only take integers as their values. In our inference problem, the variables are in fact binary. A general binary integer linear programming problem can be stated as follows.

Given a cost vector $\mathbf{p} \in \Re^d$, a set of variables, $\mathbf{u} = (u_1, \ldots, u_d)$ and cost matrices $\mathbf{C}_1 \in \Re^{c_1} \times \Re^d, \mathbf{C}_2 \in \Re^{c_2} \times \Re^d$, where c_1 and c_2 are the numbers of inequality and equality constraints and d is the number of binary variables. The ILP solution $\mathbf{u}^*$ is the vector that maximizes the cost function,

$$\mathbf{u}^* = \underset{\mathbf{u} \in \{0,1\}^d}{\operatorname{argmax}} \mathbf{p} \cdot \mathbf{u},$$

subject to

$$\mathbf{C}_1\mathbf{u} \geq \mathbf{b}_1, \text{ and } \mathbf{C}_2\mathbf{u} = \mathbf{b}_2,$$

where $\mathbf{b}_1, \mathbf{b}_2 \in \Re^d$, and for all $u \in \mathbf{u}$, $u \in \{0, 1\}$.

To solve the problem of Equation 6.2 in this setting, we first reformulate the original cost function $\sum_{i=1}^{M} \text{score}(S^i = c^i)$ as a linear function over several binary variables, and then represent the filter function $\mathcal{F}$ using linear inequalities and equalities.

We set up a bijection from the semantic labeling to the variable set $\mathbf{u}$. This is done by setting $\mathbf{u}$ to a set of indicator variables. Specifically, let $u_{ic} = [S^i = c]$ be the indicator variable that

represents whether or not the argument type c is assigned to S^i, and let $p_{ic} = \text{score}(S^i = c)$. Equation 6.1 can then be written as an ILP cost function as

$$\underset{\mathbf{u}\in\{0,1\}^d}{\text{argmax}} \sum_{i=1}^{M} \sum_{c=1}^{|\mathcal{P}|} p_{ic} u_{ic},$$

subject to

$$\sum_{c=1}^{|\mathcal{P}|} u_{ic} = 1 \qquad \forall u_{ic} \in \mathbf{u},$$

which means that each argument can take only one type. Note that this new constraint comes from the variable transformation, and is not one of the constraints used in the filter function $\mathcal{F}$.

Constraints 1 through 3 can be evaluated on a per-argument basis – for the sake of efficiency, arguments that violate these constraints are eliminated even before given to the argument classifier. Next, we show how to transform the constraints in the filter function into the form of linear (in)equalities over $\mathbf{u}$, and use them in this ILP setting.

Constraint 4: No overlapping or embedding If arguments $S^{j_1}, \ldots, S^{j_k}$ occupy the same word in a sentence, then this constraint restricts only one of the arguments to be assigned to an argument type. In other words, $k-1$ arguments will be the special class *null*, which means the argument candidate is not a legitimate argument. If the special class *null* is represented by the symbol ϕ, then for every set of such arguments, the following linear equality represents this constraint.

$$\sum_{i=1}^{k} u_{j_i \phi} = k - 1$$

Constraint 5: No duplicate argument classes Within the same sentence, several types of arguments cannot appear more than once. For example, a predicate can only take one A0. This constraint can be represented using the following inequality.

$$\sum_{i=1}^{M} u_{i\text{A0}} \leq 1$$

Constraint 6: Exactly one V argument For each verb, there is one and has to be one V argument, which represents the active verb. Similarly, this constraint can be represented by the following equality.

$$\sum_{i=1}^{M} u_{i\text{V}} = 1$$

Constraint 7: V–A1–C-V pattern This constraint is only useful when there are three consecutive candidate arguments in a sentence. Suppose arguments $S^{j_1}, S^{j_2}, S^{j_3}$ are consecutive. If S^{j_3} is C-V, then S^{j_1} and S^{j_2} have to be V and A1, respectively. This if-then constraint can be represented by the following two linear inequalities.

$$u_{j_3\text{C-V}} \geq u_{j_1\text{V}}, \text{ and } u_{j_3\text{C-V}} \geq u_{j_2\text{A1}}$$

Constraint 8: R-*arg* arguments Suppose the referenced argument type is A0 and the referential type is R-A0. The linear inequalities that represent this constraint are:

$$\forall m \in \{1, \ldots, M\} : \sum_{i=1}^{M} u_{i\text{A0}} \geq u_{m\text{R-A0}}$$

If there are γ reference argument pairs, then the total number of inequalities needed is γM.

Constraint 9: C-*arg* arguments This constraint is similar to the reference argument constraints. The difference is that the continued argument *arg* has to occur before C-*arg*. Assume that the argument pair is A0 and C-A0, and argument S_{j_i} appears before S_{j_k} if $i \leq k$. The linear inequalities that represent this constraint are:

$$\forall m \in \{2, \ldots, M\} : \sum_{i=1}^{m-1} u_{j_i\text{A0}} \geq u_{m\text{C-A0}}$$

Constraint 10: Illegal argument types Given a specific verb, some argument types should never occur. For example, most verbs don't have arguments A5. This constraint is represented by

summing all the corresponding indicator variables to be 0.

$$\sum_{i=1}^{M} u_{i\text{A5}} = 0$$

Using ILP to solve this inference problem enjoys several advantages. Linear constraints are very general, and are able to represent many types of constraints. Previous approaches usually rely on dynamic programming to resolve non overlapping/embedding constraints (i.e., Constraint 4) when the constraint structure is sequential, but are unable to handle other constraints. The ILP approach is flexible enough to handle more expressive constraints. Although solving an ILP problem is NP-hard, with the help of today's commercial numerical packages, this problem can usually be solved very fast in practice. For instance, it only takes about 10 minutes to solve the inference problem for 4305 sentences on a Pentium-III 800 MHz machine in our experiments. Note that ordinary search methods (e.g., beam search) are not necessarily faster than solving an ILP problem and do not guarantee the optimal solution.

6.4 Experimental Results in CoNLL-04

We begin the evaluation of our system in the setting of CoNLL-04. In this competition, only shallow parse information is allowed; therefore, our system does not have the pruning stage. The data provided in the CoNLL-2004 semantic-role labeling shared task consists of a portion of PropBank corpus. The training set is extracted from TreeBank (Marcus, Santorini, & Marcinkiewicz, 1993) section 15–18, the development set, used in tuning parameters of the system, from section 20, and the test set from section 21.

We first compare this system using only the scoring function from the argument identification stage, with only the non-overlapping/embedding constraints. In this simplified version of our SRL system, the two word based classifiers, *start* and *end*, are trained to be multi-class classifiers that predict the first and last word of an argument and its type. In addition, we evaluate the effectiveness

	Prec.	Rec.	F_1
ST classifiers, non-overlap	70.54	61.50	65.71
ST classifiers, all constraints	70.97	60.74	65.46
Argument classifier, non-overlap	69.69	64.75	67.13
Argument classifier, all constraints	71.96	64.93	68.26

Table 6.1: Summary of the experimental results on the development set. *ST classifiers* are the simplified SRL system based on the original argument identification stage only. *Argument classifier* is the full system. All numbers are for overall performance.

	Prec.	Rec.	F_1
Without Inference	86.95	87.24	87.10
With Inference	88.03	88.23	88.13

Table 6.2: Results of the SRL system when argument boundaries are known. Inference improves performance by correcting the labels that violate the linguistic constraints, rather than restricting structural properties since the correct boundaries are given. All numbers are for overall performance on the development set.

of using only this constraint versus all constraints, as described in Section 6.3.4.

Table 6.1 shows how additional constraints over the standard non-overlapping constraints improve performance on the development set. The argument scoring is chosen from either the ST classifiers or the argument classifier and each is evaluated by considering simply the non-overlapping/embedding constraint or the full set of constraints. To make a fair comparison, parameters were set separately to optimize performance when using the argument identification results. In general, using all constraints increases F_1 by about 1%, but slightly decreases the performance when only the ST classifiers are used. Also, using the complete system architecture improves both precision and recall, and the enhancement reflected in F_1 is about 2.5%.

It is interesting to find out how well the argument classifier can perform given perfectly segmented arguments. This evaluates the quality of the argument classifier, and also provides a conceptual upper bound. Table 6.2 first shows the results without using inference (i.e. $\mathcal{F}(\mathcal{P}^M) = \mathcal{P}^M$). The second row shows adding inference to the phrase classification can further improve F_1 by 1%.

Finally, the overall result on the official test set is given in Table 6.3. Note that the result here is not comparable with the best in this domain (Pradhan, Ward, Hacioglu, Martin, & Jurafsky, 2004) where the full parse tree is assumed given. For a fair comparison, our system was among the best at

CoNLL-04, where the best system (Hacioglu, Pradhan, Ward, Martin, & Jurafsky, 2004) achieves a 69.49 F_1 score.

	Dist.	Prec.	Rec.	F_1
Overall	100.00	70.07	63.07	66.39
A0	26.87	81.13	77.70	79.38
A1	35.73	74.21	63.02	68.16
A2	7.44	54.16	41.04	46.69
A3	1.56	47.06	26.67	34.04
A4	0.52	71.43	60.00	65.22
AM-ADV	3.20	39.36	36.16	37.69
AM-CAU	0.51	45.95	34.69	39.53
AM-DIR	0.52	42.50	34.00	37.78
AM-DIS	2.22	52.00	67.14	58.61
AM-EXT	0.15	46.67	50.00	48.28
AM-LOC	2.38	33.47	34.65	34.05
AM-MNR	2.66	45.19	36.86	40.60
AM-MOD	3.51	92.49	94.96	93.70
AM-NEG	1.32	85.92	96.06	90.71
AM-PNC	0.89	32.79	23.53	27.40
AM-TMP	7.78	59.77	56.89	58.30
R-A0	1.66	81.33	76.73	78.96
R-A1	0.73	58.82	57.14	57.97
R-A2	0.09	100.00	22.22	36.36
R-AM-TMP	0.15	54.55	42.86	48.00

Table 6.3: CoNLL-2004 shared task result on the test set

6.5 The Necessity of Syntactic Parsing

We study the necessity of syntactic parsing experimentally by observing the effects of using full parsing and shallow parsing at each stage of an SRL system. In Section 6.5.1, we first describe how we prepare the data. The comparison of full parsing and shallow parsing on the three stages (excluding the inference stage) is presented in the reversed order (Sections 6.5.2, 6.5.3, 6.5.4).

6.5.1 Experimental Setting

We use PropBank sections 02 through 21 as training data, and section 23 as testing. In order to apply the standard CoNLL-04 evaluation script, our system conforms to both the input and output format defined in the shared task.

The CoNLL-04 evaluation metric is slightly more restricted since an argument prediction is only considered correct when all its *continued* arguments (C-*arg*) are correct and *referential* arguments (R-*arg*) are included – these requirements are often absent in previous SRL systems, given that they only occupy a very small percentage of the data. To provide a fair comparison, we also report the performance when discarding continued and referential arguments. Following the notation used in (Xue & Palmer, 2004), this evaluation metric is referred as "argM+", which considers all the core arguments and adjunct arguments.

The goal of the experiments in this section is to understand the effective contribution of full parsing versus shallow parsing using only the part-of-speech tags, chunks, and clauses. In addition, we also compare performance when using the correct (gold standard) versus using automatic parse data. The automatic full parse trees are derived using Charniak's parser (Charniak, 2001) (version 0.4). In automatic shallow parsing, the information is generated by a state-of-the-art POS tagger (Even-Zohar & Roth, 2001), chunker (Punyakanok & Roth, 2001), and clauser (Carreras & Màrquez, 2004b).

6.5.2 Argument Classification

To evaluate the performance gap between full parsing and shallow parsing in argument classification, we assume the argument boundaries are known, and only train classifiers to classify the labels of these arguments. In this stage, the only difference between *full parsing* and *shallow parsing* is the construction of three full parsing features: *path*, *subcategorization* and *syntactic frame*. As described in Section 6.3, *path* and *subcategorization* can be approximated by *shallow-path* and *shallow-subcategorization* using chunks and clauses. However, it is unclear how to mimic the syn-

tactic frame feature since it relies on the internal structure of a full parse tree. Therefore, it does not have a corresponding feature in the shallow parsing case.

Table 6.4 reports the experimental results of argument classification when argument boundaries are known. Although full parsing features seem to help when using the gold standard data, the difference in F_1 is only 0.32% and 0.29% for the CoNLL-04 and ArgM+ evaluation respectively. When the automatic (full and shallow) parsers are used, the gap is smaller.

	Full Parsing	Shallow Parsing
Gold (CoNLL-04)	91.32	91.00
Auto (CoNLL-04)	90.93	90.69
Gold (ArgM+)	91.83	91.54
Auto (ArgM+)	91.10	90.93

Table 6.4: The accuracy of argument classification when argument boundaries are known

Lesson When the argument boundaries are known, the performance of the full paring systems is about the same as the shallow parsing system.

6.5.3 Argument Identification

Argument identification is an important stage that effectively reduces the number of argument candidates after pruning. Given an argument candidate, an argument identifier is a binary classifier that decides whether or not the candidate should be considered as an argument. To evaluate the influence of full parsing in this stage, the candidate list used here is the pruning results on the gold standard parse trees.

Similar to the argument classification stage, the only difference between full parsing and shallow parsing is the use of *path* and *subcategorization* features. Again, we replace them with *shallow-path* and *shallow-subcategorization* when the binary classifier is trained using the shallow parsing information.

Table 6.5 reports the performance of the argument identifier on the test set using the direct predictions of the trained binary classifier. The recall and precision of the full parsing system are

around 2 to 3 percents higher than the shallow parsing system on the gold standard data. As a result, the F_1 score is 2.5% higher. The performance on automatic parse data is unsurprisingly lower, but the difference between full parsing and shallow parsing is relatively the same. In terms of filtering efficiency, around 25% of the examples are predicted as positive. In other words, both argument identifiers filter out around 75% of the argument candidates after pruning.

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	96.53	93.57	95.03	93.66	91.72	92.68
Auto	94.68	90.60	92.59	92.31	88.36	90.29

Table 6.5: The performance of argument identification after pruning (based on the gold standard full parse trees)

Since the recall in argument identification sets the upper bound of the recall in argument classification, in practice, the threshold that predicts examples as positive is usually lowered to allow more positive predictions. That is, a candidate is predicted as positive when its probability estimation is larger than the threshold. Table 6.6 shows the performance of the argument identifiers when the threshold is 0.1.

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	92.13	95.62	93.84	88.54	94.81	91.57
Auto	89.48	94.14	91.75	86.14	93.21	89.54

Table 6.6: The performance of argument identification after pruning (based on the gold standard full parse trees) and with threshold=0.1

Since argument identification is just an intermediate step of a complete system, a more realistic evaluation method is to see how each final system performs. Table 6.7 and Table 6.8 report the final results in recall, precision, and F_1 in CoNLL-04 and ArgM+ metrics. The F_1 difference is about 4.5% when using the gold standard data. However, when automatic parsers are used, shallow parsing is in fact slightly better. This may be due to the fact that shallow parsers are more accurate in chunk or clause predictions compared to a regular full parser (Li & Roth, 2001).

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	88.81	89.35	89.08	84.19	85.03	84.61
Auto	84.21	85.04	84.63	86.17	84.02	85.08

Table 6.7: The CoNLL-04 evaluation of the overall system performance when pruning (using the gold standard full parse trees) is available

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	89.02	89.57	89.29	84.46	85.31	84.88
Auto	84.38	85.38	84.87	86.37	84.36	85.35

Table 6.8: ArgM+ performance of the overall system when pruning (using the gold standard full parse trees) is available

Lesson Full parsing helps in argument identification. However, when the automatic shallow parser is more accurate than the full parser, using the full parsing information may not have advantages over shallow parsing.

6.5.4 Pruning

As shown in the previous two subsections, the performance difference of full parsing and shallow parsing is not large when the pruning information is given. We conclude that the main contribution of the full parse is in the pruning stage. Since the shallow parsing system does not have enough information for the pruning heuristics, we train two word based classifiers to replace the pruning stage (similar to the simplified SRL system in Section 6.4). One classifier is trained to predict whether a given word is the start (S) of an argument; the other classifier is to predict the end (E) of an argument. If the product of probabilities of a pair of S and E predictions is larger than a predefined threshold, then this pair is considered as an argument candidate. The pruning comparison of using the classifiers and heuristics is shown in Table 6.9.

Amazingly, the classifier pruning strategy seems better than the heuristics. With about the same recall, the classifiers achieve higher precision. However, to really compare systems using full parsing and shallow parsing, we still need to see the overall performance. We build two

	Full Parsing			Classifier th=0.04		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	25.94	97.27	40.96	29.58	97.18	45.35
Auto	22.79	86.08	36.04	24.68	94.80	39.17

Table 6.9: The performance of pruning

semantic role systems based on full parsing and shallow parsing. The full parsing system follows the pruning, argument identification, argument classification, and inference stages, as described earlier. For the shallow parsing system, pruning is replaced by the word-based pruning classifiers, and the rest stages are designed only to use shallow parsing as described in previous sections. Table 6.10 and Table 6.11 show the overall performance in the two evaluation metrics.

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	88.81	89.35	89.08	75.34	75.28	75.31
Auto	77.09	75.51	76.29	75.48	67.13	71.06

Table 6.10: The CoNLL-04 evaluation of the overall system performance

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	89.02	89.57	89.29	75.35	75.20	75.27
Auto	77.09	75.57	76.32	75.54	67.14	71.09

Table 6.11: ArgM+ performance of the overall system

As indicated in the tables, the gap in F_1 between the full parsing and shallow parsing systems enlarges to more than 13% on the gold standard data. At first glance, this result seems to contradict our conclusion in Section 6.5.3. After all, if the pruning stage of the shallow parsing SRL system performs equally well or even better, the overall performance gap in F_1 should be small.

After we carefully examine the output of the word-based classifier pruning, we realize that it in fact filters out "easy" candidates, and leaves examples that are difficult to the later stages. To be specific, these argument candidates often overlap and differ only with one or two words. On the other hand, the pruning heuristics based on full parsing never outputs overlapping candidates. The following argument identification stage can be thought of as good in discriminating different types

of candidates.

Lesson The most crucial contribution of full parsing is in pruning. The internal tree structure helps significantly in discriminating argument candidates, which makes the work easy to the following stages.

6.6 Joint Inference

The empirical study in Section 6.5 indicates the performance of an SRL system primarily depends on the very first stage – pruning, which is derived directly from the full parse trees. This also means that in practice the quality of the syntactic parser is decisive to the quality of the SRL system. To improve semantic role labeling, one possible way is to combine different SRL systems through a joint inference stage, given that the systems are derived using different full parse trees.

To test this idea, we first build two SRL systems that use Collins' parser (Collins, 1999)[1] and Charniak's parser (Charniak, 2001) respectively. In fact, these two parsers have noticeably different output. Applying pruning heuristics on the output of Collins' parser produces a list of candidates with 81.05% recall. Although this number is significantly lower that 86.08% recall produced by Charniak's parser, the union of the two candidate lists still significantly improves recall to 91.37%. We construct the two systems by implementing the first three stages, namely pruning, argument identification, and argument classification. When a testing sentence is given, a joint inference stage is used to resolve the inconsistency of the output of argument classification in these two systems.

We first briefly review the objective function used in the inference procedure introduced in Section 6.3.4. Formally speaking, the argument classifier attempts to assign labels to a set of arguments, $S^{1:M}$, indexed from 1 to M. Each argument S^i can take any label from a set of argument labels, $\mathcal{P}$, and the indexed set of arguments can take a set of labels, $c^{1:M} \in \mathcal{P}^M$. If we assume that the argument classifier returns an estimated conditional probability distribution, $Prob(S^i = c^i)$, then, given a sentence, the inference procedure seeks a global assignment that maximizes the

[1] We use the Collins' parser implemented by Bikel (2004).

..., traders say, unable to **cool** the selling panic in both stocks and futures.

Figure 6.1: The output of two SRL systems: system A has two candidates, a_1 = "traders" and a_4 = "the selling panic in both stocks and futures"; system B has three argument candidates, b_1 = "traders", b_2 = "the selling panic", and b_3 = "in both stocks and futures". In addition, we create two phantom candidates a_2 and a_3 for system A that correspond to b_2 and b_3 respectively, and b_4 for system B that corresponds to a_4.

objective function denoted by Equation 6.2, which can be rewritten as follows.

$$\hat{c}^{1:M} = \underset{c^{1:M} \in \mathcal{F}(\mathcal{P}^M)}{\operatorname{argmax}} \sum_{i=1}^{M} Prob(S^i = c^i), \tag{6.3}$$

where the linguistic and structural constraints are represented by the filter $\mathcal{F}$. In other words, this objective function reflects the expected number of correct argument predictions, subject to the constraints.

When there are two or more argument classifiers from different SRL systems, a joint inference procedure can take the output estimated probabilities for these candidates as input, although some candidates may refer to the same phrases in the sentence. For example, Figure 6.1 shows the two candidate sets for a fragment of a sentence, "*..., traders say, unable to* ***cool*** *the selling panic in both stocks and futures.*" In this example, system A has two argument candidates, a_1 = "traders" and a_4 = "the selling panic in both stocks and futures"; system B has three argument candidates, b_1 = "traders", b_2 = "the selling panic", and b_3 = "in both stocks and futures".

If we throw all these variables together into the inference procedure, then the final prediction

will be more likely dominated by the system that has more candidates, which is system B in this example. The reason is because our objective function is the sum of the probabilities of all the candidate assignments.

This bias can be corrected by the following observation. Although system A only has two candidates, a_1 and a_4, it can be treated as it also has two additional *phantom* candidates, a_2 and a_3, where a_2 and b_2 refer to the same phrase, and so do a_3 and b_3. Similarly, system B has a phantom candidate b_4 that corresponds to a_4. Because system A does not really generate a_2 and a_3, we can assume that these two phantom candidates are predicted as *null* (i.e., not an argument). We assign the same prior distribution to each phantom candidate. In particular, the probability of the *null* class is set to be 0.55 based on empirical tests, and the probabilities of the rest classes are set based on their occurrence frequencies in the training data.

Tables 6.12 and 6.13 report the performance of individual systems, as well as the joint system. The joint system based on this straightforward strategy significantly improves the performance compared to the two original SRL systems in both recall and precision, and thus achieves a much higher F_1.

	Prec	Rec	F_1
Collins' Parse	75.92	71.45	73.62
Charniak's Parse	77.09	75.51	76.29
Combined Result	80.53	76.94	78.69

Table 6.12: The performance in CoNLL-04's evaluation of individual and combined SRL systems

	Prec	Rec	F_1
Collins' Parse	75.87	71.36	73.54
Charniak's Parse	77.09	75.57	76.32
Combined Result	80.56	76.99	78.73

Table 6.13: The performance in argM+'s evaluation of individual and combined SRL systems

6.7 Discussion

In this chapter, we show that linguistic information is useful for semantic role labeling, both in extracting features and deriving hard constraints on the output. We also demonstrate that it is possible to use integer linear programming to perform inference that incorporates a wide variety of hard constraints, which would be difficult to incorporate using existing methods.

In addition to building a state-of-the-art SRL system in the CoNLL-04 shared task competition, we make a fair comparison between the SRL systems using full parse tree information and using only shallow syntactic information. What we found confirms the necessity of full-parsing for the SRL problem. In particular, this information is the most crucial in the pruning stage of the system, and relatively less important to the following stages.

Inspired by this observation, we proposed an effective and simple approach that combines different SRL systems through a joint inference stage. The combined system significantly improves the performance compared to the original systems.

Chapter 7

Conclusions

In the preceding chapters, we proposed a novel propositionalization approach for relational feature generation. In particular, we defined a language that allows users to generate relational features in a data-driven way. This approach has served as the fundamental step, feature extraction, in all of the machine learning based information extraction systems described in this dissertation.

This suggests that the relational learning framework is more flexible and allows the use of any propositional algorithm within it, including probabilistic approaches. As such, it addresses the problem of using relational representations within a probabilistic framework in a general and natural way. It is important when the learned systems are used as individual components in a complex system, where the probabilities provide crucial information for global inference.

We also proposed two inference procedures: one based on Bayesian networks and the other using integer linear programming. Although the preliminary result of the Bayesian network approach is promising, our focus is on the linear programming approach because of its generality and flexibility; in particular, it supports the ability to incorporate classifiers learned in other contexts and constraints known at training or decision time; then it reasons with all these for the best global prediction. This approach is very general, as linear equalities and inequalities are very powerful, as they are able to encode any Boolean constraints. Our experiments have demonstrated these advantages by considering the interaction between entity and relation classifiers, and by encoding complicated linguistic and structural constraints in semantic role labeling.

The success of our approaches have pushed forward the fundamental technologies of infor-

mation extraction. In addition, it also indicates several directions for long term research. As for relational feature generation, the most interesting direction is to investigate how the advantages of our approach can be used to move our learning framework more toward a "real" relational learning method. In order to reduce the burden of defining the "types" of features, we can further study automatic ways of abstracting features and learning the RGFs needed for a given problem.

As for the global inference framework, a promising and important direction is to scale up the approach to allow more variables and constraints. For example, if coreference resolution is available in the entity/relation recognition task, it is possible to incorporate it in the form of constraints that force the labels of the co-referred entities to be the same (but, of course, allowing the global solution to reject the suggestion of these classifiers). Consequently, this may enhance the performance of entity/relation recognition and, at the same time, correct possible coreference resolution errors. Another example is to use chunking information for better relation identification; suppose, for example that we have available chunking information that identifies Subj+Verb and Verb+Object phrases. Given a sentence that has the verb "murder", we may conclude that the subject and object of this verb are in a "kill" relation.

Finally, the general inference procedure based on integer linear programming has also facilitated the study of the interaction between learning and inference for structured output. For example, the trade-off of different learning strategies has been investigated in (Punyakanok, Roth, Yih, & Zimak, 2005). In the future, we hope to develop a more theoretical understanding along this research direction.

Appendix A

Global Inference Using Integer Linear Programming

This appendix gives a simple but complete step-by-step case study, which demonstrates how we apply integer linear programming (ILP) to solve a global inference problem in natural language processing. This framework first transforms an optimization problem into an integer linear program. The program can then be solved using existing numerical packages.

The goal here is to provide readers an easy-to-follow example to model their own problems in this framework. The goal of this appendix is twofold. Section A.1 describe a problem of labeling entities and relations simultaneously as our inference task. It then discusses the constraints among the labels and shows how the objective function and constraints are transformed to an integer linear program. Although transforming the constraints to their linear forms is not difficult in this entity and relation example, sometimes it can be tricky, especially when more variables are involved. Therefore, we discuss how to handle different types of constraints in Section A.2.

A.1 Labeling Entities and Relations

Given a sentence, the task is to assign labels to the entities in this sentence, and identify the relation of each pair of these entities. Each entity is a phrase and we assume the boundaries of these entities are given.

Figure A.1 gives an example of the sentence “Dole’s wife, Elizabeth, is a native of N.C.” In this

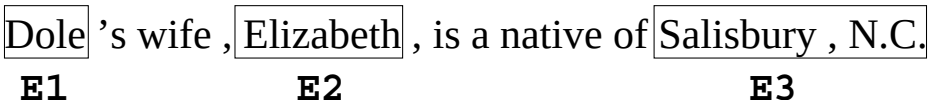

Figure A.1: A sentence that has 3 entities

sentence, there are three entities, *Dole*, *Elizabeth*, and *N.C.* We use $E_1, E_2,$ and E_3 to represent their entity labels. In this example, possible classes include ***other***, ***person***, and ***location***. In addition, we would like to know the relation between each pair of the entities. For a pair of two entities E_i and E_j, the relation is denoted by R_{ij}. In this example, there will be 6 relation variables – $R_{12}, R_{21}, R_{13}, R_{31}, R_{23}, R_{32}$. Since most entities have no interesting relation, the values of most relation variables should be ***irrelevant***. Besides this special label, the relations of interest in this example are ***spouse_of*** and ***born_in***.

variable	other	person	location
E_1	0.05	0.85	0.10
E_2	0.10	0.60	0.30
E_3	0.05	0.50	0.45

variable	irrelevant	spouse_of	born_in
R_{12}	0.05	0.45	0.50
R_{21}	0.75	0.10	0.15
R_{13}	0.85	0.05	0.10
R_{31}	0.80	0.05	0.15
R_{23}	0.10	0.05	0.85
R_{32}	0.65	0.20	0.15

Table A.1: The confidence scores on the labels of each variable

Assume that magically some local classifiers have already provided the *confidence* scores on possible labels, as shown in Table A.1. If we want to choose the labels that maximize the sum of those confidence scores, it's the same as choosing the label that has the highest score for each variable. The global labeling then becomes Table A.2:

At this point, the problem seems to have been solved by the local classifiers. However, after a second look at this labeling, we can easily find the inconsistency between entity and relation labels. For example, R_{12} cannot be *born_in* if both entities E_1 and E_2 are *persons*. Indeed, there exists

variable	label	score
E_1	person	0.85
E_2	person	0.60
E_3	person	0.50
R_{12}	born_in	0.50
R_{21}	irrelevant	0.75
R_{13}	irrelevant	0.85
R_{31}	irrelevant	0.80
R_{23}	born_in	0.85
R_{32}	irrelevant	0.65
	sum	6.35

Table A.2: The global labeling when the individual highest scores are picked

some natural constraints between the labels of entity and relation variables that the local classifiers may not know or respect. In our example, we know the global labeling also satisfies the following two constraints.

- if R_{ij} = spouse_of, then E_i = person AND E_j = person
- if R_{ij} = born_in, then E_i = person AND E_j = location

In summary, the problem we want to solve here really is to find the best legitimate global labeling, which is the one that maximizes the sum of the confidence scores, subject to the constraints.

Note that although exhaustive search seems plausible in this toy problem, it soon becomes intractable when the number of variables or the number of possible labels grows. In the rest of this section, we are going to show that how we transfer this problem to an integer linear program, and let the numerical packages help us find the answer.

A.1.1 Indicator Variables

In order to apply (integer) linear programming, both the objective function and constraints have to be linear. Since the confidence score could be any real number, the original function is not linear. In addition, the logic constraints we have are not linear as well.

To overcome this difficulty, the first step of the transformation is to introduce several *indicator* (binary) variables, which represent the assignment of the original variables. For each entity or relation variable a and each legitimate label k, we introduce a binary variable $x_{a,k}$. When the original variable a is assigned label k, $x_{a,k}$ is set to 1. Otherwise, $x_{a,k}$ is 0. In our toy example, we then have 27 such indicator variables:

$$
\begin{array}{ccc}
x_{E_1,\text{other}}, & x_{E_1,\text{person}}, & x_{E_1,\text{location}}, \\
x_{E_2,\text{other}}, & x_{E_2,\text{person}}, & x_{E_2,\text{location}}, \\
x_{E_3,\text{other}}, & x_{E_3,\text{person}}, & x_{E_3,\text{location}}, \\
x_{R_{12},\text{irrelevant}}, & x_{R_{12},\text{spouse\_of}}, & x_{R_{12},\text{born\_in}}, \\
x_{R_{21},\text{irrelevant}}, & x_{R_{21},\text{spouse\_of}}, & x_{R_{21},\text{born\_in}}, \\
x_{R_{13},\text{irrelevant}}, & x_{R_{13},\text{spouse\_of}}, & x_{R_{13},\text{born\_in}}, \\
x_{R_{31},\text{irrelevant}}, & x_{R_{31},\text{spouse\_of}}, & x_{R_{31},\text{born\_in}}, \\
x_{R_{23},\text{irrelevant}}, & x_{R_{23},\text{spouse\_of}}, & x_{R_{23},\text{born\_in}}, \\
x_{R_{32},\text{irrelevant}}, & x_{R_{32},\text{spouse\_of}}, & x_{R_{32},\text{born\_in}}.
\end{array}
$$

To simplify the notation, let $L_E = \{\text{other}, \text{person}, \text{location}\}$ and $L_R = \{\text{irrelevant}, \text{spouse\_of}, \text{born\_in}\}$ represent the sets of entity and relation labels, respectively. Assume $n = 3$ means the number of entities we have in the sentence. The indicator variables we introduce are:

$$
\begin{aligned}
&x_{E_i,l_e}, \quad \text{where } 1 \le i \le n \text{ and } l_e \in L_E \\
&x_{R_{ij},l_r}, \quad \text{where } 1 \le i, j \le n, i \ne j, \text{ and } l_r \in L_R
\end{aligned}
$$

A.1.2 Objective Function

Suppose c_{E_i,l_e} represents the confidence score of E_i being l_e, where $1 \le i \le n$ and $l_e \in L_E$, and c_{R_{ij},l_r} represents the confidence score of R_{ij} being l_r, where $1 \le i, j \le n, i \ne j$ and $l_r \in L_R$. The

objective function f (i.e., the sum of confidence scores) can be represented by

$$f = \sum_{1 \le i \le n, l_e \in L_E} c_{E_i,l_e} x_{E_i,l_e} + \sum_{1 \le i,j \le n, i \ne j, l_r \in L_R} c_{R_{ij},l_r} x_{R_{ij},l_r}$$

If we plug in the numbers in Table A.1, the function f is:

$$f = 0.05 \cdot x_{E_1,\text{other}} + 0.85 \cdot x_{E_1,\text{person}} + \cdots + 0.65 \cdot x_{R_{32},\text{irrelevant}} + 0.20 \cdot x_{R_{32},\text{spouse\_of}} + 0.15 \cdot x_{R_{32},\text{born\_in}}$$

Inevitably, this transformation also brings new constraints, which come from the fact that one entity/relation variable can only have one label, and must have one label. For example, only exact one of the labels *other*, *person*, *location* can be assigned to E_1. As a result, only one of the indicator variables $x_{E_1,other}$, $x_{E_1,person}$, $x_{E_1,location}$ can and must be 1. This restriction can be easily written as the following linear equations.

$$\begin{aligned} \textstyle\sum_{l_e \in L_E} x_{E_i,l_e} &= 1 \quad \forall 1 \le i \le n \\ \textstyle\sum_{l_r \in L_R} x_{R_{ij},l_r} &= 1 \quad \forall 1 \le i,j \le n, i \ne j \end{aligned}$$

A.1.3 Logic Constraints

The other reason of introducing indicator variables is to handle the real constraints we have – the logic constraints between entity and relation labels. In our example, they are:

- if R_{ij} = spouse_of, then E_i = person AND E_j = person, where $1 \le i,j \le n$ and $i \ne j$
- if R_{ij} = born_in, then E_i = person AND E_j = location, where $1 \le i,j \le n$ and $i \ne j$

If we treat the indicator variables as boolean variables, where 1 means *true* and 0 means f*alse*, the constraints can be rephrased as:

$$\begin{aligned} x_{R_{ij},\text{spouse\_of}} &\rightarrow x_{E_i,person} \wedge x_{E_j,person} \quad 1 \le i,j \le n, \text{ and } i \ne j \\ x_{R_{ij},born\_in} &\rightarrow x_{E_i,person} \wedge x_{E_j,location} \quad 1 \le i,j \le n, \text{ and } i \ne j \end{aligned}$$

$$\max \quad \sum_{1\le i\le n, l_e\in L_E} c_{E_i,l_e} x_{E_i,l_e} + \sum_{1\le i,j\le n, i\neq j, l_r\in L_R} c_{R_{ij},l_r} x_{R_{ij},l_r}$$

subject to:

$$x_{E_i,l_e} \in \{0,1\} \qquad \forall 1 \le i \le n \qquad \text{(A.1)}$$

$$x_{R_{ij},l_r} \in \{0,1\} \qquad \forall 1 \le i,j \le n, i \neq j \qquad \text{(A.2)}$$

$$\sum_{l_e\in L_E} x_{E_i,l_e} = 1 \qquad \forall 1 \le i \le n \qquad \text{(A.3)}$$

$$\sum_{l_r\in L_R} x_{R_{ij},l_r} = 1 \qquad \forall 1 \le i,j \le n, i \neq j \qquad \text{(A.4)}$$

$$2 \cdot x_{R_{ij},\text{spouse\_of}} \le x_{E_i,\text{person}} + x_{E_j,\text{person}} \quad 1 \le i,j \le n, \text{ and } i \neq j \qquad \text{(A.5)}$$

$$2 \cdot x_{R_{ij},\text{born\_in}} \le x_{E_i,\text{person}} + x_{E_j,\text{location}} \quad 1 \le i,j \le n, \text{ and } i \neq j \qquad \text{(A.6)}$$

Figure A.2: The complete integer linear program

In fact, these two boolean constraints can be modeled by the following two linear inequalities.

$$2 \cdot x_{R_{ij},\text{spouse\_of}} \le x_{E_i,\text{person}} + x_{E_j,\text{person}} \quad 1 \le i,j \le n, \text{ and } i \neq j$$

$$2 \cdot x_{R_{ij},\text{born\_in}} \le x_{E_i,\text{person}} + x_{E_j,\text{location}} \quad 1 \le i,j \le n, \text{ and } i \neq j$$

When $x_{R_{ij},\text{spouse\_of}}$ is 0 (false), $x_{E_i,\text{person}}$ and $x_{E_j,\text{person}}$ can be either 0 or 1, and the inequality holds. However, when $x_{R_{ij},\text{spouse\_of}}$ is 1 (true), both $x_{E_i,\text{person}}$ and $x_{E_j,\text{person}}$ have to be 1 (true). Therefore, the original logic rule can be represented by this linear inequality.

Transforming the logic constraints into linear forms is the key of this framework. It is not difficult, but may be tricky sometimes. We will talk more about transforming other types of logic constraints in Section A.2 later.

A.1.4 Solving the Integer Linear Program Using Xpress-MP

Figure A.2 shows the complete integer linear program. Now, all we need to do now is to apply some numeric packages, such as Xpress-MP (Xpress-MP, 2004), CPLEX (CPLEX, 2003), or the LP solver in R (The R Project for Statistical Computing, 2004), to solve it. Transferring the solution back to the global labeling we want is straightforward – just find those indicator variables that have

the value 1. In this section, I will demonstrate how to apply Xpress-MP to do the job.

The syntax in Xpress-MP is fairly easy and straightforward. Here I simply list the source code with some comments, which are the lines beginning with the "!" symbol.

```
model "Entity Relation Inference"
 uses "mmxprs"

parameters
  DATAFILE = "er.dat"
  Num_Entities = 3;
end-parameters

declarations
  ENTITIES = 1..Num_Entities
  ENT_CLASSES = {"Other", "Person", "Location"}
  REL_CLASSES = {"Irrelevant", "SpouseOf", "BornIn"}

  scoreEnt: array(ENTITIES, ENT_CLASSES) of real
  scoreRel: array(ENTITIES, ENTITIES, REL_CLASSES) of real
end-declarations

! DATAFILE stores the confidence scores from the local classifiers.
initializations from DATAFILE
  scoreEnt   scoreRel
end-initializations

! These are the indicator variables.
declarations
  ent : array(ENTITIES, ENT_CLASSES) of mpvar
  rel : array(ENTITIES, ENTITIES, REL_CLASSES) of mpvar
end-declarations

! The objective function: sum of confidence scores
```

```
Obj := sum(u in ENTITIES, e in ENT_CLASSES) scoreEnt(u,e)*ent(u,e)
  + sum(u,v in ENTITIES, r in REL_CLASSES|u <> v) scoreRel(u,v,r)*rel(u,v,r)

! Constraints (A.1) and (A.2): the variables take only binary values
forall(u in ENTITIES, e in ENT_CLASSES)
  ent(u,e) is_binary
forall(e1,e2 in ENTITIES, r in REL_CLASSES | e1 <> e2)
  rel(e1,e2,r) is_binary

! Constraints (A.3) and (A.4): sum = 1
forall(u in ENTITIES) sum(e in ENT_CLASSES)
  ent(u,e) = 1
forall(u,v in ENTITIES | u <> v) sum(r in REL_CLASSES)
  rel(u,v,r) = 1

! Constraints (A.5) and (A.6): logic constraints on entities and relations
forall(e1,e2 in ENTITIES | e1 <> e2)
  2*rel(e1,e2,"SpouseOf") <= ent(e1,"Person") + ent(e2,"Person")
forall(e1,e2 in ENTITIES | e1 <> e2)
  2*rel(e1,e2,"BornIn") <= ent(e1,"Person") + ent(e2,"Location")

! Solve the problem
maximize(Obj)

! Output the indicator variables that are 1
forall(u in ENTITIES, e in ENT_CLASSES | getsol(ent(u,e)) >= 1)
  writeln(u, " ", e)
forall(e1,e2 in ENTITIES, r in REL_CLASSES |
           e1 <> e2 and getsol(rel(e1,e2,r)) >= 1)
  writeln(e1, " ", e2, " ", r)

end-model
```

A.2 Transforming Logic Constraints into Linear Forms

This section summarizes and revises some rules of transforming logic constraints to linear inequalities and equalities described in (Guéret, Prins, & Sevaux, 2002). To simplify the illustration, symbols a, b, c and $x_1, x_2, \cdots, x_n$ are used to represent indicator variables, which are treated as boolean variables and binary variables at the same time. As usual, the values $0, 1$ represent the truth values *false* and *true*, respectively.

A.2.1 Choice Among Several Possibilities

In our entity and relation example, we have already processed the constraint "exactly k variables among $x_1, x_2, \cdots, x_n$ are true", where $k = 1$. The general form of this linear equation is:

$$x_1 + x_2 + \cdots + x_n = k$$

Another constraint, "at most k variables among $x_1, x_2, \cdots, x_n$ can be true", can be represented in a similar inequality.

$$x_1 + x_2 + \cdots + x_n \leq k$$

Uninterestingly, "k or more variables among $x_1, x_2, \cdots, x_n$ must be true" will be

$$x_1 + x_2 + \cdots + x_n \geq k$$

A.2.2 Implications

Implications are usually the logic constraints we encounter. While handling two or three variables may be trivial, extending it to more variables may be tricky. Here we illustrate how to develop the ideas from the simplest case to complicated constraints.

Two variables Suppose there are only two indicator variables a, b in the implication. The constraint, $a \rightarrow b$, can be represented as $a \leq b$. This can be easily verified by the following truth table.

$a \leq b$	$b = 0$	$b = 1$
$a = 0$	true	true
$a = 1$	false	true

What if we need to deal with something like $a \rightarrow \bar{b}$? The value of the compliment of b is exactly $1 - b$. Therefore, the corresponding linear constraint is $a \leq 1 - b$, or $a + b \leq 1$.

The relation "if and only if" is straightforward too. $a \leftrightarrow b$ is identical to $a \rightarrow b$ and $b \rightarrow a$. The corresponding linear constraints are $a \leq b$ and $b \leq a$, which is in fact $a = b$.

Three variables We can generalize the implication a little bit to cover three variables. Since $a \rightarrow b \wedge c$ can be separated as $a \rightarrow b$ and $a \rightarrow c$, the straightforward transformation is to put two linear inequalities $a \leq b$ and $a \leq c$. Alternatively, the transformation in our entity and relation example "$2a \leq b + c$" also suffice, which is easy to check using a truth table.

Another implication, $a \rightarrow b \vee c$, can be modeled by $a \leq b + c$. This is because when $a = 1$, at least one of b and c has to be 1 to make the inequality correct.

The inverse of the above two implications can be derived using the compliment and DeMorgan's Theorem. $b \wedge c \rightarrow a$ is equivalent to $\bar{a} \rightarrow \overline{b \wedge c}$, which is $\bar{a} \rightarrow \bar{b} \vee \bar{c}$. Use the above rule and the the compliment, it can be modeled by $(1-a) \leq (1-b)+(1-c)$, or equivalently $a \geq b+c-1$. $b \vee c \rightarrow a$ is equivalent to $b \rightarrow a$ and $c \rightarrow a$, so it can be modeled by two inequalities $b \leq a$ and $c \leq a$. Alternatively, this can be transformed to $\bar{a} \rightarrow \overline{b \vee c}$, which is $\bar{a} \rightarrow \bar{b} \wedge \bar{c}$. Therefore, it can be modeled by $2(1 - a) \leq (1 - b) + (1 - c)$, or $\frac{b+c}{2} \leq a$.

More variables A logic constraint that has more variables can be complicated. Therefore, we only discuss some common cases here. Suppose we want to model "if a, then k or more variables among $x_1, x_2, \cdots, x_n$ are true." We can extend the transformation of $a \rightarrow b \vee c$, and use the

following linear inequality.

$$a \leq \frac{x_1 + x_2 + \cdots + x_n}{k}$$

This transforation is certainly valid for $k = 1$. It is also easy to verify for other cases. If $a = 0$, then the right-hand-side (RHS) is always larger or equal to 0, and the inequality is satisfied. However, when $a = 1$, it forces at least k of the x variables are true, which is exactly what we want.

The next case we would like to try is the inverse, which is "if k or more variables among $x_1, x_2, \cdots, x_n$ are true, then a is true." This might be somewhat tricker than others. Our first guess might be:

$$(x_1 + x_2 + \cdots + x_n) - (k - 1) \leq a$$

Although this may seem correct at the first glance, we observe that the left-hand-side (LHS) will be larger than 1 when more than k of the x variables are 1. Because a can be either 0 or 1, this constraint will be infeasible. In fact, what we really need is to *squash* the LHS to less than 1. Currently, the largest possible value of the LHS is $n - (k - 1)$. Therefore, dividing the LHS by $n - (k - 1)$ should suffice.

$$\frac{(x_1 + x_2 + \cdots + x_n) - (k - 1)}{n - (k - 1)} \leq a$$

We can examine two special cases of this transformation to see if they are correct. Note that $b \vee c \rightarrow a$ is indeed one of these cases, given that $n = 2$ and $k = 1$. The linear inequality $\frac{b+c}{2} \leq a$ is exactly the same as what we derived previously. The other special case is "$x_1 \wedge x_2 \wedge \cdots \wedge x_n \rightarrow a$", which is equivalent to say $k = n$ here. Obviously, $a \geq x_1 + x_2 + \cdots + x_n - (n-1)$ is correct. One interesting observation is that the conjunction of a set of boolean variables is the same as the product of the corresponding binary variables. Therefore, the nonlinear constraint $a = x_1 \cdot x_2 \cdots x_n$ is the same as $a = x_1 \wedge x_2 \wedge \cdots \wedge x_n$. Its linear transformation is therefore $a \geq x_1 + x_2 + \cdots + x_n - (n - 1)$ and $a \leq \frac{x_1 + x_2 + \cdots + x_n}{n}$.

Table A.3 summarizes all the transformations we have discussed in this section.

Original constraint	Linear form
Exactly k of $x_1, x_2, \cdots, x_n$	$x_1 + x_2 + \cdots + x_n = k$
At most k of $x_1, x_2, \cdots, x_n$	$x_1 + x_2 + \cdots + x_n \leq k$
At least k of $x_1, x_2, \cdots, x_n$	$x_1 + x_2 + \cdots + x_n \geq k$
$a \rightarrow b$	$a \leq b$
$a = \bar{b}$	$a = 1 - b$
$a \rightarrow \bar{b}$	$a + b \leq 1$
$\bar{a} \rightarrow b$	$a + b \geq 1$
$a \leftrightarrow b$	$a = b$
$a \rightarrow b \wedge c$	$a \leq b$ and $a \leq c$ or, $a \leq \frac{b+c}{2}$
$a \rightarrow b \vee c$	$a \leq b + c$
$b \wedge c \rightarrow a$	$a \geq b + c - 1$
$b \vee c \rightarrow a$	$a \geq \frac{b+c}{2}$
if a then at least k of $x_1, x_2, \cdots, x_n$	$a \leq \frac{x_1+x_2+\cdots+x_n}{k}$
if at least k of $x_1, x_2, \cdots, x_n$ then a	$a \geq \frac{x_1+x_2+\cdots+w_n-(k-1)}{n-(k-1)}$
$a = x_1 \cdot x_2 \cdots x_n$	$a \leq \frac{x_1+x_2+\cdots+x_n}{n}$ and $a \geq x_1 + x_2 + \cdots + x_n - (n-1)$

Table A.3: Rules of mapping constraints to linear (in)equalities

A.3 Conclusions

Thanks to the theoretical development of integer linear programming in the last two decades and the tremendous improvement on hardware and software technologies, numerical packages these days are able to solve many large integer linear programming problems within very short time, even though ILP is in general NP-hard.

In this appendix, we have provided an entity and relation problem as example, and discussed several cases for transforming boolean constraints. We hope these illustrations can help the readers to also apply integer linear programming in their inference problems.

Bibliography

Abney, S. P. (1991). Parsing by chunks. In R. C. Berwick, S. P. A., & Tenny, C. (Eds.), *Principle-based parsing: Computation and Psycholinguistics* (pp. 257–278). Dordrecht: Kluwer.

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

Bikel, D. (2004, December). Intricacies of Collins' parsing model. *Computational Linguistics, 30*(4), 479–511.

Bishop, C. (1995). *Neural networks for pattern recognition* (Chapter 6.4: Modelling conditional distributions, pp. 215). Oxford University Press.

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, August). *Relational learning techniques for natural language information extraction*. Doctoral dissertation, 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). AAAI.

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.

Carleson, A., Cumby, C., Rosen, J., & Roth, D. (1999, May). *The SNoW learning architecture* (Technical Report UIUCDCS-R-99-2101). UIUC Computer Science Department.

Carlson, A., Cumby, C., Rosen, J., Rizzolo, N., & Roth, D. (2004, August). SNoW user manual. http://l2r.cs.uiuc.edu/∼cogcomp/software/snow-userguide/.

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). AAAI.

Carreras, X., & Màrquez, L. (2004a). Introduction to the CoNLL-2004 shared tasks: Semantic role labeling. In *Proc. of CoNLL-2004* (pp. 89–97). Boston, MA, USA.

Carreras, X., & Màrquez, L. (2004b). Online learning via global feedback for phrase recognition. In *Proc. of NIPS 2003*. MIT Press.

Charniak, E. (2001). Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics* (pp. 116–123). Toulouse, France.

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

Chekuri, C., Khanna, S., Naor, J., & Zosin, L. (2001). Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *Symposium on Discrete Algorithms* (pp. 109–118).

Chen, J., & Rambow, O. (2003). Use of deep linguistic features for the recognition and labeling of semantic arguments. In *Proc. of EMNLP-2003* (pp. 41–48). Sapporo, Japan.

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.

Collins, M. (1999). *Head-driven statistical models for natural language parsing.* Doctoral dissertation, Computer Science Department, University of Pennsylvenia, Philadelphia.

CPLEX (2003). ILOG, Inc. CPLEX. http://www.ilog.com/products/cplex/.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., & Slattery, S. (1998). Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence* (pp. 509–516). Madison, US: AAAI Press, Menlo Park, US.

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 *Proceedings of the Conference on Principiles of Knowledge Representation and Reasoning (KR-00)* (pp. 425–434).

Cumby, C., & Roth, D. (2002). Learning with feature description logics. In *Proceedings of the 12th International Conference on Inductive Logic Programming (ILP-02)* (pp. 32–47).

Cumby, C. M., & Roth, D. (2003, August). 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)* (pp. 1–8).

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 *Proc. of the Conference on Empirical Methods for Natural Language Processing (EMNLP)* (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)*, Volume 1634 of *LNAI* (pp. 92–103). Springer-Verlag.

Flach, P., & Lachiche, N. (2001, January). 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).

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*(3), 277–296.

Gallager, R. (1962, Jan). Low density parity check codes. *IRE Trans. Info. Theory*, *IT-8*, 21–28.

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

Gildea, D., & Hockenmaier, J. (2003). Identifying semantic roles using combinatory categorial grammar. In *Proc. of the EMNLP-2003*. Sapporo, Japan.

Gildea, D., & Palmer, M. (2002). The necessity of parsing for predicate argument recognition. In *Proc. of ACL 2002* (pp. 239–246). Philadelphia, PA.

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.

Guéret, C., Prins, C., & Sevaux, M. (2002). *Applications of optimization with xpress-mp*. Dash Optimization. Translated and revised by Susanne Heipcke.

Hacioglu, K., Pradhan, S., Ward, W., Martin, J. H., & Jurafsky, D. (2004). Semantic role labeling by tagging syntactic chunks. In *Proc. of CoNLL-04* (pp. 110–113).

Har-Peled, S., Roth, D., & Zimak, D. (2002). Constraint classification: A new approach to multiclass classification and ranking. In *Neural Information Processing Systems* (pp. 809–816). MIT Press.

Hinton, G. (1986, August). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (pp. 1–12). Amherst, Mass.

Hirschman, L., Light, M., Breck, E., & Burger, J. (1999). Deep read: A reading comprehension system. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics* (pp. 325–348).

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.

Kingsbury, P., & Palmer, M. (2002). From Treebank to PropBank. In *Proc. of LREC-2002*. Spain.

Kleinberg, J., & Tardos, E. (1999). Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. In *IEEE Symposium on Foundations of Computer Science* (pp. 14–23).

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)* (pp. 258–265).

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, September). 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.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of the International Conference on Machine Learning* (pp. 282–289).

Lavrac, N., & Dzeroski, D. (1994). *Inductive logic programming: Techniques and applications*. London: Ellis Horwood.

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.

Li, S. (2001). *Markov random field modeling in image analisys*. Springer-Verlag.

Li, X., & Roth, D. (2001). Exploring evidence for shallow parsing. In *Proc. of the Annual Conference on Computational Natural Language Learning* (pp. 107–110).

Li, Y., Zaragoza, H., Herbrich, R., Shawe-Taylor, J., & Kandola, J. (2002). The perceptron algorithm with uneven margins. In *Proc. of the International Conference on Machine Learning* (pp. 379–386).

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.

MacKay, D. (1999). Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory, 45*(2), 399–431.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. (1993, June). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics, 19*(2), 313–330.

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

Miller, G., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. (1990). Wordnet: An on-line lexical database. *International Journal of Lexicography, 3(4)*, 235–312.

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.

Muñoz, M., Punyakanok, V., Roth, D., & Zimak, D. (1999, June). A learning approach to shallow parsing. In *EMNLP-VLC'99, the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora* (pp. 168–178).

Murphy, K., Weiss, Y., & Jordan, M. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proc. of Uncertainty in AI* (pp. 467–475).

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.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann.

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).

Pradhan, S., Hacioglu, K., Ward, W., Martin, J., & Jurafsky, D. (2003). Semantic role parsing adding semantic structure to unstructured text. In *Proc. of ICDM-2003* (pp. 629–632).

Pradhan, S., Ward, W., Hacioglu, K., Martin, J. H., & Jurafsky, D. (2004). Shallow semantic parsing using support vector machines. In *Proc. of NAACL-HLT 2004* (pp. 233–240).

Punyakanok, V., & Roth, D. (2001). The use of classifiers in sequential inference. In *NIPS-13; The 2000 Conference on Advances in Neural Information Processing Systems* (pp. 995–1001). MIT Press.

Punyakanok, V., Roth, D., Yih, W., & Zimak, D. (2005). Learning and inference over constrained output. In *National Conference on Artificial Intelligence*.

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*. New York: Spartan.

Roth, D. (1996, April). On the hardness of approximate reasoning. *Artificial Intelligence*, *82*(1-2), 273–302.

Roth, D. (1998). Learning to resolve natural language ambiguities: A unified approach. In *Proc. National Conference on Artificial Intelligence* (pp. 806–813).

Roth, D., Yang, M., & Ahuja, N. (2000). Learning to recognize objects. In *2000 Conference on Computer Vision and Pattern Recognition* (pp. 1724–1731).

Roth, D., Yang, M.-H., & Ahuja, N. (2002). Learning to recognize objects. *Neural Computation*, *14*(5), 1071–1104.

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).

Schrijver, A. (1986, December). *Theory of linear and integer programming*. Wiley Interscience series in discrete mathmatics. John Wiley & Sons.

Sebag, M., & Rouveirol, C. (1997). Tractable induction and classification in FOL. In *Proc. 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.

Surdeanu, M., Harabagiu, S., Williams, J., & Aarseth, P. (2003). Using predicate-argument structures for information extraction. In *Proc. of ACL 2003* (pp. 8–15).

The R Project for Statistical Computing (2004). http://www.r-project.org/.

Tjong Kim Sang, E. F., & Buchholz, S. (2000). Introduction to the CoNLL-2000 shared task: Chunking. In *Proc. of the CoNLL-2000 and LLL-2000* (pp. 127–132).

Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proc. of CoNLL-2003* (pp. 142–147).

Voorhees, E. (2000). Overview of the trec-9 question answering track. In *The Ninth Text Retrieval Conference (TREC-9)* (pp. 71–80). NIST SP 500-249.

Wang, X., & Regan, A. (2000). *A cutting plane method for integer programming problems with binary variables* (Technical Report UCI-ITS-WP-00-12). University of California, Irvine.

Wolsey, L. (1998). *Integer programming*. John Wiley & Sons, Inc.

Xpress-MP (2004). Dash Optimization. Xpress-MP. http://www.dashoptimization.com/products.html.

Xue, N., & Palmer, M. (2004). Calibrating features for semantic role labeling. In *Proc. of the EMNLP-2004* (pp. 88–94). Barcelona, Spain.

Author's Biography

Wen-tau Yih was born in Yunlin, Taiwan, on January 12, 1973. He graduated from the National Taiwan University in 1995 with a Bachelor of Science in Engineering. In 1997, he received his Master of Science in Computer Science from the National Taiwan University. Before relocating to Champaign, Illinois, USA to pursue further graduate study in Computer Science, Yih joined the Armed Forces Taoyuan General Hospital as an IT officer with a rank of Second Lieutenant to fulfill his military obligation. During his study in the Department of Computer Science at the University of Illinois at Urbana-Champaign, Yih focused his research on Machine Learning and Natural Language Processing, and has published several academic papers in various highly selective conferences. Following the completion of his Ph.D., Yih will continue his research career as a Post-Doc Researcher at Microsoft Research in Redmond, Washington, USA.